

# **Mathematical Programming and Operations Research**

**Modeling, Algorithms, and Complexity  
Examples in Excel and Python  
(Work in progress)**

Edited by: Robert Hildebrand

Contributors: Robert Hildebrand, Laurent Poirrier, Douglas Bish, Diego Moran

Version Compilation date: July 1, 2022



# Preface

---

This entire book is a working manuscript. The first draft of the book is yet to be completed.

This book is being written and compiled using a number of open source materials. We will strive to properly cite all resources used and give references on where to find these resources. Although the material used in this book comes from a variety of licences, everything used here will be CC-BY-SA 4.0 compatible, and hence, the entire book will fall under a CC-BY-SA 4.0 license.

## MAJOR ACKNOWLEDGEMENTS

I would like to acknowledge that substantial parts of this book were borrowed under a CC-BY-SA license. These substantial pieces include:

- "A First Course in Linear Algebra" by Lyryx Learning (based on original text by Ken Kuttler). A majority of their formatting was used along with selected sections that make up the appendix sections on linear algebra. We are extremely grateful to Lyryx for sharing their files with us. They do an amazing job compiling their books and the templates and formatting that we have borrowed here clearly took a lot of work to set up. Thank you for sharing all of this material to make structuring and formating this book much easier! See subsequent page for list of contributors.
- "Foundations of Applied Mathematics" with many contributors. See <https://github.com/Foundations-of-Applied-Mathematics>. Several sections from these notes were used along with some formatting. Some of this content has been edited or rearranged to suit the needs of this book. This content comes with some great references to code and nice formatting to present code within the book. See subsequent page with list of contributors.
- "Linear Inequalities and Linear Programming" by Kevin Cheung. See <https://github.com/dataopt/lineqlpbook>. These notes are posted on GitHub in a ".Rmd" format for nice reading online. This content was converted to L<sup>A</sup>T<sub>E</sub>X using Pandoc. These notes make up a substantial section of the Linear Programming part of this book.
- Linear Programming notes by Douglas Bish. These notes also make up a substantial section of the Linear Programming part of this book.

I would also like to acknowledge Laurent Porrier and Diego Moran for contributing various notes on linear and integer programming.

I would also like to thank Jamie Fravel for helping to edit this book and for contributing chapters, examples, and code.



# Contents

---

<b>Contents</b>	<b>3</b>
<b>1 Resources and Notation</b>	<b>1</b>
<b>2 Mathematical Programming</b>	<b>5</b>
2.1 Linear Programming (LP) . . . . .	5
2.2 Mixed-Integer Linear Programming (MILP) . . . . .	7
2.3 Non-Linear Programming (NLP) . . . . .	8
2.3.1 Convex Programming . . . . .	9
2.3.2 Non-Convex Non-linear Programming . . . . .	9
2.4 Mixed-Integer Non-Linear Programming (MINLP) . . . . .	10
2.4.1 Convex Mixed-Integer Non-Linear Programming . . . . .	10
2.4.2 Non-Convex Mixed-Integer Non-Linear Programming . . . . .	10
<b>I Linear Programming</b>	<b>11</b>
<b>3 Linear Programming</b>	<b>13</b>
3.1 Modeling Assumptions in Linear Programming . . . . .	15
3.2 Examples . . . . .	17
3.2.1 Knapsack Problem . . . . .	23
3.2.2 Work Scheduling . . . . .	23
3.2.3 Assignment Problem . . . . .	23
3.2.4 Multi period Models . . . . .	25
3.2.4.1 Production Planning . . . . .	25
3.2.4.2 Crop Planning . . . . .	25
3.2.5 Mixing Problems . . . . .	25
3.2.6 Financial Planning . . . . .	25
3.2.7 Network Flow . . . . .	25
3.2.7.1 Graphs . . . . .	25
3.2.7.2 Maximum Flow Problem . . . . .	26
3.2.7.3 Minimum Cost Network Flow . . . . .	27
3.2.8 Multi-Commodity Network Flow . . . . .	29
3.3 Modeling Tricks . . . . .	30
3.3.1 Maximizing a minimum . . . . .	30
3.4 Other examples . . . . .	30
<b>4 Graphically Solving Linear Programs</b>	<b>31</b>

## 6 ■ CONTENTS

4.1	Nonempty and Bounded Problem . . . . .	31
4.2	Infinitely Many Optimal Solutions . . . . .	35
4.3	Problems with No Solution . . . . .	38
4.4	Problems with Unbounded Feasible Regions . . . . .	40
4.5	Formal Mathematical Statements . . . . .	45
<b>5</b>	<b>Software - Excel</b>	<b>51</b>
5.0.1	Excel Solver . . . . .	51
5.0.2	Videos . . . . .	51
5.0.3	Links . . . . .	51
<b>6</b>	<b>Software - Python</b>	<b>53</b>
6.1	Installing and Managing Python . . . . .	53
6.2	NumPy Visual Guide . . . . .	57
6.3	Plot Customization and Matplotlib Syntax Guide . . . . .	61
6.4	Networkx - A Python Graph Algorithms Package . . . . .	67
6.5	PuLP - An Optimization Modeling Tool for Python . . . . .	68
6.5.1	Installation . . . . .	68
6.5.2	Example Problem . . . . .	69
6.5.2.1	Product Mix Problem . . . . .	69
6.5.3	Things we can do . . . . .	70
6.5.3.1	Exploring the variables . . . . .	71
6.5.3.2	Other things you can do . . . . .	71
6.5.4	Common issue . . . . .	72
6.5.4.1	Transportation Problem . . . . .	72
6.5.4.2	Optimization with PuLP . . . . .	73
6.5.4.3	Optimization with PuLP: Round 2! . . . . .	74
6.5.5	Changing details of the problem . . . . .	76
6.5.6	Changing Constraint Coefficients . . . . .	78
6.6	Multi Objective Optimization with PuLP . . . . .	78
6.6.0.1	Transportation Problem . . . . .	78
6.6.0.2	Initial Optimization with PuLP . . . . .	79
6.6.1	Creating the Pareto Efficient Frontier . . . . .	81
6.7	Comments . . . . .	83
6.8	Jupyter Notebooks . . . . .	83
6.9	Reading and Writing . . . . .	84
6.10	Python Crash Course . . . . .	84
6.11	Gurobi . . . . .	84
6.12	Plots, Pandas, and Geopandas . . . . .	84
6.12.1	Geopandas . . . . .	84
6.13	Linear Optimization . . . . .	85
6.13.1	Problem Formulation . . . . .	85

<b>II OLD LP Stuff</b>	<b>93</b>
<b>7 LP Notes from Foundations of Applied Mathematics</b>	<b>95</b>
7.1 The Simplex Method . . . . .	101
7.1.1 Pivoting . . . . .	105
7.1.2 Termination and Reading the Dictionary . . . . .	107
7.2 The Product Mix Problem . . . . .	108
<b>8 Linear Programming Notes - Hildebrand</b>	<b>111</b>
8.0.0.1 Simplex Tableau Pivoter . . . . .	111
8.0.0.2 Videos . . . . .	111
8.1 Linear Programming Forms . . . . .	112
8.2 Linear Programming Dual . . . . .	112
8.3 Weak and Strong Duality . . . . .	112
8.3.1 Reduced Costs . . . . .	113
8.3.2 Tableau Based Pivoting . . . . .	114
<b>9 Linear Programming Book - Cheung</b>	<b>117</b>
<b>Preface</b>	<b>117</b>
<b>Notation</b>	<b>117</b>
9.1 Graphical example . . . . .	118
Exercises . . . . .	120
Solutions . . . . .	121
9.2 Definitions . . . . .	122
Exercises . . . . .	124
Solutions . . . . .	125
9.3 Farkas' Lemma . . . . .	125
9.4 Fundamental Theorem of Linear Programming . . . . .	126
Exercises . . . . .	127
Solutions . . . . .	128
9.5 Linear programming duality . . . . .	129
9.5.1 The dual problem . . . . .	132
Exercises . . . . .	134
Solutions . . . . .	135
9.6 Complementary slackness . . . . .	136
Exercises . . . . .	138
Solutions . . . . .	139
9.7 Basic feasible solution . . . . .	141
Exercises . . . . .	143
Solutions . . . . .	143
<b>10 LP Notes from ISE 5405</b>	<b>145</b>
10.1 Introduction to Optimization . . . . .	145

10.1.1 Notation . . . . .	146
10.2 Linear Optimization . . . . .	147
10.2.1 Problem Formulation . . . . .	147
10.2.2 Linear Algebra Review . . . . .	154
10.2.3 Linear Optimization Theory . . . . .	167
10.2.3.1 Optimality Conditions . . . . .	169
10.2.4 Solution Algorithms . . . . .	175
10.2.4.1 The Simplex Algorithm . . . . .	176
10.2.4.2 Dual Simplex Algorithm . . . . .	185
10.2.4.3 Primal-Dual Algorithm . . . . .	186
10.2.5 Sensitivity Analysis . . . . .	189
10.2.6 Theory Applications . . . . .	200
10.3 Other material for Integer Linear Programming . . . . .	202
Exercises . . . . .	206
Solutions . . . . .	206
<b>11 Simplex Method</b>	<b>207</b>
11.1 Simplex Method . . . . .	210
11.2 Finding Feasible Basis . . . . .	210
<b>12 Duality</b>	<b>215</b>
12.1 The Dual of Linear Program . . . . .	216
12.2 Linear programming duality . . . . .	220
12.2.1 The dual problem . . . . .	223
Exercises . . . . .	225
Solutions . . . . .	226
<b>13 Sensitivity Analysis</b>	<b>229</b>
<b>14 Multi-Objective Optimization</b>	<b>231</b>
14.1 Multi Objective Optimization and The Pareto Frontier . . . . .	231
14.2 What points will the Scalarization method find if we vary $\lambda$ ? . . . . .	237
14.3 Political Redistricting [3] . . . . .	237
14.4 Portfolio Optimization [5] . . . . .	237
14.5 Simulated Portfolio Optimization based on Efficient Frontier . . . . .	238
14.6 Aircraft Design [1] . . . . .	238
14.7 Vehicle Dynamics [4] . . . . .	239
14.8 Sustainable Constriction [2] . . . . .	239
14.9 References . . . . .	239
<b>III Discrete Algorithms</b>	<b>241</b>
<b>15 Graph Algorithms</b>	<b>243</b>
15.1 Graph Theory and Network Flows . . . . .	243

15.2 Graphs . . . . .	243
15.2.1 Drawing Graphs . . . . .	243
15.3 Definitions . . . . .	246
15.4 Shortest Path . . . . .	249
15.5 Spanning Trees . . . . .	258
15.6 Exercise Answers . . . . .	262
15.7 Additional Exercises . . . . .	264
15.7.1 Notes . . . . .	272
<b>IV Integer Programming</b>	<b>273</b>
<b>16 Integer Programming Formulations</b>	<b>275</b>
16.1 Knapsack Problem . . . . .	275
16.2 Capital Budgeting . . . . .	278
16.3 Set Covering . . . . .	280
16.3.1 Covering (Generalizing Set Cover) . . . . .	284
16.4 Assignment Problem . . . . .	284
16.5 Facility Location . . . . .	285
16.5.1 Capacitated Facility Location . . . . .	286
16.5.2 Uncapacitated Facility Location . . . . .	287
16.6 Basic Modeling Tricks - Using Binary Variables . . . . .	287
16.6.1 Connecting to continuous variables . . . . .	289
16.6.2 Exact absolute value . . . . .	290
16.6.2.1 Exact 1-norm . . . . .	290
16.6.2.2 Maximum . . . . .	291
16.7 Network Flow . . . . .	291
16.7.1 Example - Multicommodity Flow . . . . .	291
16.7.2 Corresponding optimization problems . . . . .	292
16.7.3 Relation to other problems . . . . .	292
16.7.4 Usage . . . . .	293
16.8 Transportation Problem . . . . .	293
16.9 Other examples . . . . .	293
16.10 Notes from AIMMS modeling book . . . . .	293
16.10.1 Further Topics . . . . .	294
16.11 MIP Solvers and Modeling Tools . . . . .	294
<b>17 Algorithms and Complexity</b>	<b>295</b>
17.1 Big-O Notation . . . . .	295
17.2 Algorithms - Example with Bubble Sort . . . . .	299
17.2.1 Sorting . . . . .	299
17.3 Problem, instance, size . . . . .	304
17.3.1 Problem, instance . . . . .	304
17.3.2 Format and examples of problems/instances . . . . .	304

17.3.3 Size of an instance . . . . .	304
<b>17.4 Complexity Classes . . . . .</b>	<b>305</b>
17.4.1 P . . . . .	306
17.4.2 NP . . . . .	307
17.4.3 Problem Reductions . . . . .	308
17.4.4 NP-Hard . . . . .	308
17.4.5 NP-Complete . . . . .	309
<b>17.5 Problems and Algorithms . . . . .</b>	<b>310</b>
17.5.1 Matching Problem . . . . .	311
17.5.1.1 Greedy Algorithm for Maximal Matching . . . . .	312
17.5.1.2 Other algorithms to look at . . . . .	312
17.5.2 Minimum Spanning Tree . . . . .	312
17.5.3 Kruskal's algorithm . . . . .	313
17.5.3.1 Prim's Algorithm . . . . .	314
17.5.4 Traveling Salesman Problem . . . . .	314
17.5.4.1 Nearest Neighbor - Construction Heuristic . . . . .	314
17.5.4.2 Double Spanning Tree - 2-Apx . . . . .	315
17.5.4.3 Christofides - Approximation Algorithm - (3/2)-Apx . . . . .	316
<b>18 Introduction to computational complexity</b>	<b>317</b>
18.1 Introduction . . . . .	317
18.2 Problem, instance, size . . . . .	317
18.2.1 Problem, instance . . . . .	317
18.2.2 Format and examples of problems/instances . . . . .	318
18.2.3 Size of an instance . . . . .	318
18.3 Algorithms, running time, Big-O notation . . . . .	318
18.3.1 Basics . . . . .	318
18.3.2 Worst-time complexity . . . . .	319
18.3.3 Big-O notation . . . . .	319
18.3.4 Examples . . . . .	319
18.4 Basics . . . . .	319
18.5 Complexity classes . . . . .	320
18.5.1 Polynomial time problems . . . . .	320
18.5.2 Non-deterministic polynomial time problems . . . . .	320
18.5.3 Complements of problems in NP . . . . .	321
18.6 Relationship between the classes . . . . .	321
18.6.1 A basic result . . . . .	321
18.6.2 An \$1,000,000 open question . . . . .	321
18.7 Comparing problems, Polynomial time reductions . . . . .	321
18.8 Comparing problems, Polynomial time reductions . . . . .	322
18.8.1 Definition . . . . .	322
18.8.2 Basic properties . . . . .	323
18.9 NP-Completeness . . . . .	323
18.9.1 The basics . . . . .	323

18.9.2	Do NP-complete problems exist? . . . . .	324
18.10	NP-Hardness . . . . .	324
18.11	Exercises . . . . .	324
<b>19</b>	<b>Exponential Size Formulations</b>	<b>331</b>
19.1	Cutting Stock . . . . .	331
19.1.1	Pattern formulation . . . . .	333
19.1.2	Column Generation . . . . .	335
19.1.3	Cutting Stock - Multiple widths . . . . .	336
19.2	Spanning Trees . . . . .	337
19.3	Traveling Salesman Problem . . . . .	337
19.3.1	Miller Tucker Zemlin (MTZ) Model . . . . .	339
19.3.2	Dantzig-Fulkerson-Johnson (DFJ) Model . . . . .	344
19.3.3	Traveling Salesman Problem - Branching Solution . . . . .	347
19.3.4	Traveling Salesman Problem Variants . . . . .	347
19.3.4.1	Many salespersons (m-TSP) . . . . .	347
19.3.4.2	TSP with order variants . . . . .	350
19.4	Vehicle Routing Problem (VRP) . . . . .	350
19.4.1	Case Study: Bus Routing in Boston . . . . .	350
19.5	Steiner Tree Problem . . . . .	351
19.6	Literature and other notes . . . . .	351
19.6.1	Google maps data . . . . .	352
19.6.2	TSP In Excel . . . . .	352
<b>20</b>	<b>Algorithms to Solve Integer Programs</b>	<b>353</b>
20.1	LP to solve IP . . . . .	353
20.1.1	Rounding LP Solution can be bad! . . . . .	354
20.1.2	Rounding LP solution can be infeasible! . . . . .	354
20.1.3	Fractional Knapsack . . . . .	354
20.2	Branch and Bound . . . . .	354
20.2.1	Algorithm . . . . .	355
20.2.2	Knapsack Problem and 0/1 branching . . . . .	357
20.2.3	Traveling Salesman Problem solution via Branching . . . . .	359
20.3	Cutting Planes . . . . .	359
20.3.1	Chvátal Cuts . . . . .	361
20.3.2	Gomory Cuts . . . . .	362
20.3.3	Cover Inequalities . . . . .	364
20.4	Branching Rules . . . . .	365
20.5	Lagrangian Relaxation for Branch and Bound . . . . .	365
20.6	Benders Decomposition . . . . .	365
20.7	Literature . . . . .	366
20.8	Other material for Integer Linear Programming . . . . .	366
Exercises	. . . . .	370
Solutions	. . . . .	370

20.8.1 Other discrete problems . . . . .	371
20.8.2 Assignment Problem and the Hungarian Algorithm . . . . .	371
20.8.3 History of Computation in Combinatorial Optimization . . . . .	371
<b>21 Heuristics for TSP</b>	<b>373</b>
21.1 Construction Heuristics . . . . .	373
21.1.1 Random Solution . . . . .	373
21.1.2 Nearest Neighbor . . . . .	374
21.1.3 Insertion Method . . . . .	374
21.2 Improvement Heuristics . . . . .	374
21.2.1 2-Opt (Subtour Reversal) . . . . .	375
21.2.2 3-Opt . . . . .	376
21.2.3 $k$ -Opt . . . . .	376
21.3 Meta-Heuristics . . . . .	376
21.3.1 Hill Climbing (2-Opt for TSP) . . . . .	376
21.3.2 Simulated Annealing . . . . .	378
21.3.3 Tabu Search . . . . .	379
21.3.4 Genetic Algorithms . . . . .	380
21.3.5 Greedy randomized adaptive search procedure (GRASP) . . . . .	380
21.3.6 Ant Colony Optimization . . . . .	380
21.4 Computational Comparisons . . . . .	380
<b>V Nonlinear Programming</b>	<b>383</b>
<b>22 Non-linear Programming (NLP)</b>	<b>385</b>
22.1 Convex Sets . . . . .	386
22.2 Convex Functions . . . . .	387
22.2.1 Proving Convexity - Characterizations . . . . .	390
22.2.2 Proving Convexity - Composition Tricks . . . . .	391
22.3 Convex Optimization Examples . . . . .	392
22.3.1 Unconstrained Optimization: Linear Regression . . . . .	392
22.4 Machine Learning - SVM . . . . .	394
22.4.0.1 Feasible separation . . . . .	395
22.4.0.2 Distance between hyperplanes . . . . .	395
22.4.0.3 SVM . . . . .	397
22.4.0.4 Approximate SVM . . . . .	398
22.4.1 SVM with non-linear separators . . . . .	398
22.4.2 Support Vector Machines . . . . .	400
22.5 Classification . . . . .	401
22.5.1 Machine Learning . . . . .	401
22.5.2 Neural Networks . . . . .	401
22.6 Box Volume Optimization in Scipy.Minimize . . . . .	401
22.7 Modeling . . . . .	401

22.7.1 Minimum distance to circles . . . . .	402
22.8 Machine Learning . . . . .	405
22.9 Machine Learning - Supervised Learning - Regression . . . . .	405
22.10 Machine learning - Supervised Learning - Classification . . . . .	406
22.10.1 Python SGD implementation and video . . . . .	406
<b>23 NLP Algorithms</b>	<b>407</b>
23.1 Algorithms Introduction . . . . .	407
23.2 1-Dimensional Algorithms . . . . .	407
23.2.1 Golden Search Method - Derivative Free Algorithm . . . . .	408
23.2.1.1 Example: . . . . .	409
23.2.2 Bisection Method - 1st Order Method (using Derivative) . . . . .	409
23.2.2.1 Minimization Interpretation . . . . .	409
23.2.2.2 Root finding Interpretation . . . . .	409
23.2.3 Gradient Descent - 1st Order Method (using Derivative) . . . . .	410
23.2.4 Newton's Method - 2nd Order Method (using Derivative and Hessian) . . . . .	410
23.3 Multi-Variate Unconstrained Optimizaiton . . . . .	411
23.3.1 Descent Methods - Unconstrained Optimization - Gradient, Newton . . . . .	411
23.3.1.1 Choice of $\alpha_t$ . . . . .	411
23.3.1.2 Choice of $d_t$ using $\nabla f(x)$ . . . . .	411
23.3.2 Stochastic Gradient Descent - The mother of all algorithms. . . . .	412
23.3.3 Neural Networks . . . . .	413
23.3.4 Choice of $\Delta_k$ using the hessian $\nabla^2 f(x)$ . . . . .	413
23.4 Constrained Convex Nonlinear Programming . . . . .	414
23.4.1 Barrier Method . . . . .	414
<b>24 Computational Issues with NLP</b>	<b>417</b>
24.1 Irrational Solutions . . . . .	417
24.2 Discrete Solutions . . . . .	417
24.3 Convex NLP Harder than LP . . . . .	417
24.4 NLP is harder than IP . . . . .	418
24.5 Karush-Huhn-Tucker (KKT) Conditions . . . . .	418
24.6 Gradient Free Algorithms . . . . .	421
24.6.1 Needler-Mead . . . . .	421
<b>25 Material to add...</b>	<b>423</b>
25.0.1 Bisection Method and Newton's Method . . . . .	423
25.1 Gradient Descent . . . . .	423
<b>26 Fairness in Algorithms</b>	<b>425</b>
<b>VI Appendix - Linear Algebra Background</b>	<b>427</b>
<b>A Linear Transformations</b>	<b>429</b>

<b>B Linear Systems</b>	<b>443</b>
<b>C Systems of Equations</b>	<b>459</b>
C.1 Systems of Equations, Geometry . . . . .	459
C.2 Systems Of Equations, Algebraic Procedures . . . . .	463
C.2.1 Elementary Operations . . . . .	464
C.2.2 Gaussian Elimination . . . . .	468
C.2.3 Uniqueness of the Reduced Row-Echelon Form . . . . .	480
C.2.4 Rank and Homogeneous Systems . . . . .	482
<b>D Matrices</b>	<b>487</b>
D.1 Matrix Arithmetic . . . . .	487
D.1.1 Addition of Matrices . . . . .	489
D.1.2 Scalar Multiplication of Matrices . . . . .	491
D.1.3 Multiplication of Matrices . . . . .	492
D.1.4 The $i j^{th}$ Entry of a Product . . . . .	499
D.1.5 Properties of Matrix Multiplication . . . . .	501
D.1.6 The Transpose . . . . .	502
D.1.7 The Identity and Inverses . . . . .	504
D.1.8 Finding the Inverse of a Matrix . . . . .	506
<b>E Determinants</b>	<b>513</b>
E.1 Basic Techniques and Properties . . . . .	513
E.1.1 Cofactors and $2 \times 2$ Determinants . . . . .	513
E.1.2 The Determinant of a Triangular Matrix . . . . .	518
E.2 Applications of the Determinant . . . . .	520
E.2.1 Cramer's Rule . . . . .	520
<b>F <math>\mathbb{R}^n</math></b>	<b>523</b>
F.1 Vectors in $\mathbb{R}^n$ . . . . .	523
F.2 Algebra in $\mathbb{R}^n$ . . . . .	526
F.2.1 Addition of Vectors in $\mathbb{R}^n$ . . . . .	526
F.2.2 Scalar Multiplication of Vectors in $\mathbb{R}^n$ . . . . .	528
F.3 Geometric Meaning of Vector Addition . . . . .	529
F.4 Length of a Vector . . . . .	532
F.5 Geometric Meaning of Scalar Multiplication . . . . .	536
F.6 The Dot Product . . . . .	538
F.6.1 The Dot Product . . . . .	538
F.6.2 The Geometric Significance of the Dot Product . . . . .	541
<b>G Spectral Theory</b>	<b>545</b>
G.1 Eigenvalues and Eigenvectors of a Matrix . . . . .	545
G.1.1 Definition of Eigenvectors and Eigenvalues . . . . .	545
G.1.2 Finding Eigenvectors and Eigenvalues . . . . .	548
G.1.3 Eigenvalues and Eigenvectors for Special Types of Matrices . . . . .	555

G.2 Positive Semi-Definite Matrices . . . . .	555
G.2.1 Definitions . . . . .	556
G.2.2 Eigenvalues . . . . .	558
G.2.3 Quadratic forms . . . . .	558
G.2.4 Properties . . . . .	558
G.2.4.1 Inverse of positive definite matrix . . . . .	558
G.2.4.2 Scaling . . . . .	559
G.2.4.3 Addition . . . . .	559
G.2.4.4 Multiplication . . . . .	559
G.2.4.5 Cholesky decomposition . . . . .	559
G.2.4.6 Square root . . . . .	559
G.2.4.7 Submatrices . . . . .	560
G.2.5 Convexity . . . . .	560
G.2.5.1 Further properties . . . . .	560
G.2.5.2 Block matrices . . . . .	560
G.2.5.3 Local extrema . . . . .	561
G.2.5.4 Covariance . . . . .	561
G.2.6 External links . . . . .	561
<b>VII Other Appendices</b>	<b>563</b>
<b>A Some Prerequisite Topics</b>	<b>565</b>
A.1 Sets and Set Notation . . . . .	565
A.2 Well Ordering and Induction . . . . .	567
A.3 GECODE and MiniZinc . . . . .	570
A.4 Optaplanner . . . . .	571
A.5 Python Modeling/Optimization . . . . .	571
A.5.1 SCIP . . . . .	571
A.5.2 Pyomo . . . . .	571
A.5.3 Python-MIP . . . . .	571
A.5.4 Local Solver . . . . .	571
A.5.5 GUROBI . . . . .	571
A.5.6 CPLEX . . . . .	572
A.5.7 Scipy . . . . .	572
A.6 Julia . . . . .	572
A.6.1 JuMP . . . . .	572
A.7 LINDO/LINGO . . . . .	572
A.8 Foundations of Machine Learning . . . . .	572
A.9 Convex Optimization . . . . .	572
<b>B Dynamic Programming</b>	<b>573</b>
B.1 Contributors . . . . .	582
B.1.1 Graph Theory . . . . .	3



# 1. Resources and Notation

---

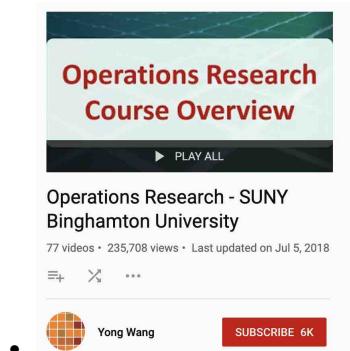
Here are a list of resources that may be useful as alternative references or additional references.

## FREE NOTES AND TEXTBOOKS

- Linear Programming by K.J. Mtetwa, David
- A first course in optimization by Jon Lee
- Introduction to Optimization Notes by Komei Fukuda
- Convex Optimization by Boyd and Vandenberghe
- LP notes of Michel Goemans from MIT
- Understanding and Using Linear Programming - Matousek and Gärtner [Downloadable from Springer with University account]
- Operations Research Problems Statements and Solutions - Raúl Poler, Josefa Mula, Manuel Díaz-Madroñero [Downloadable from Springer with University account]

## NOTES, BOOKS, AND VIDEOS BY VARIOUS SOLVER GROUPS

- AIMMS Optimization Modeling
- Optimization Modeling with LINGO by Linus Schrage
- The AMPL Book
- Microsoft Excel 2019 Data Analysis and Business Modeling, Sixth Edition, by Wayne Winston - Available to read for free as an e-book through Virginia Tech library at O'Reilly.com.
- Lesson files for the Winston Book
- Video instructions for solver and an example workbook



## 2 ■ Resources and Notation

### **GUROBI LINKS**

- Go to <https://github.com/Gurobi> and download the example files.
- Essential ingredients
- Gurobi Linear Programming tutorial
- Gurobi tutorial MILP
- GUROBI - Python 1 - Modeling with GUROBI in Python
- GUROBI - Python II: Advanced Algebraic Modeling with Python and Gurobi
- GUROBI - Python III: Optimization and Heuristics
- Webinar Materials
- GUROBI Tutorials

### **HOW TO PROVE THINGS**

- Hammack - Book of Proof

### **STATISTICS**

- Open Stax - Introductory Statistics

### **LINEAR ALGEBRA**

- Beezer - A first course in linear algebra
- Selinger - Linear Algebra
- Cherney, Denton, Thomas, Waldron - Linear Algebra

### **REAL ANALYSIS**

- Mathematical Analysis I by Elias Zakon

## DISCRETE MATHEMATICS, GRAPHS, ALGORITHMS, AND COMBINATORICS

- Levin - Discrete Mathematics - An Open Introduction, 3rd edition
- Github - Discrete Mathematics: an Open Introduction CC BY SA
- Keller, Trotter - Applied Combinatorics (CC-BY-SA 4.0)
- Keller - Github - Applied Combinatorics

## PROGRAMMING WITH PYTHON

- A Byte of Python
- Github - Byte of Python (CC-BY-SA)

Also, go to <https://github.com/open-optimization/open-optimization-or-examples> to look at more examples.

# Notation

---

- $\mathbf{1}$  - a vector of all ones (the size of the vector depends on context)
- $\forall$  - for all
- $\exists$  - there exists
- $\in$  - in
- $\therefore$  - therefore
- $\Rightarrow$  - implies
- s.t. - such that (or sometimes "subject to".... from context?)
- $\{0,1\}$  - the set of numbers 0 and 1
- $\mathbb{Z}$  - the set of integers (e.g.  $1, 2, 3, -1, -2, -3, \dots$ )
- $\mathbb{Q}$  - the set of rational numbers (numbers that can be written as  $p/q$  for  $p, q \in \mathbb{Z}$  (e.g.  $1, 1/6, 27/2$ )
- $\mathbb{R}$  - the set of all real numbers (e.g.  $1, 1.5, \pi, e, -11/5$ )
- $\setminus$  - setminus, (e.g.  $\{0, 1, 2, 3\} \setminus \{0, 3\} = \{1, 2\}$ )
- $\cup$  - union (e.g.  $\{1, 2\} \cup \{3, 5\} = \{1, 2, 3, 5\}$ )
- $\cap$  - intersection (e.g.  $\{1, 2, 3, 4\} \cap \{3, 4, 5, 6\} = \{3, 4\}$ )
- $\{0,1\}^4$  - the set of 4 dimensional vectors taking values 0 or 1, (e.g.  $[0,0,1,0]$  or  $[1,1,1,1]$ )
- $\mathbb{Z}^4$  - the set of 4 dimensional vectors taking integer values (e.g.,  $[1, -5, 17, 3]$  or  $[6, 2, -3, -11]$ )
- $\mathbb{Q}^4$  - the set of 4 dimensional vectors taking rational values (e.g.  $[1.5, 3.4, -2.4, 2]$ )
- $\mathbb{R}^4$  - the set of 4 dimensional vectors taking real values (e.g.  $[3, \pi, -e, \sqrt{2}]$ )
- $\sum_{i=1}^4 i = 1 + 2 + 3 + 4$
- $\sum_{i=1}^4 i^2 = 1^2 + 2^2 + 3^2 + 4^2$
- $\sum_{i=1}^4 x_i = x_1 + x_2 + x_3 + x_4$
- $\square$  - this is a typical Q.E.D. symbol that you put at the end of a proof meaning "I proved it."

## 4 ■ Resources and Notation

- For  $x, y \in \mathbb{R}^3$ , the following are equivalent (note, in other contexts, these notations can mean different things)
  - $x^\top y$  *matrix multiplication*
  - $x \cdot y$  *dot product*
  - $\langle x, y \rangle$  *inner product*

and evaluate to  $\sum_{i=1}^3 x_i y_i = x_1 y_1 + x_2 y_2 + x_3 y_3$ .

A sample sentence:

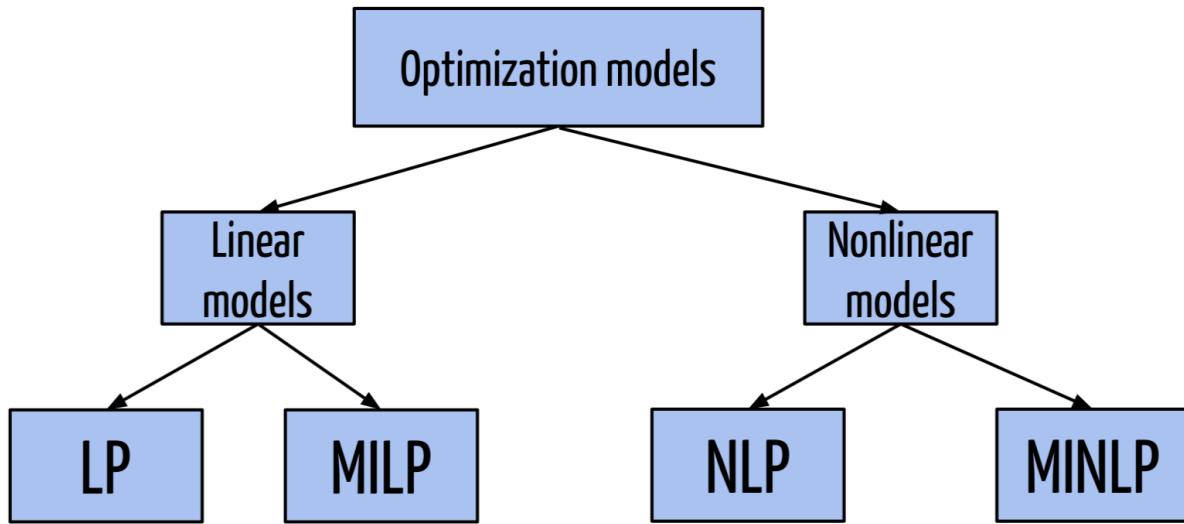
$$\forall x \in \mathbb{Q}^n \exists y \in \mathbb{Z}^n \setminus \{0\}^n s.t. x^\top y \in \{0, 1\}$$

"For all non-zero rational vectors  $x$  in  $n$ -dimensions, there exists a non-zero  $n$ -dimensional integer vector  $y$  such that the dot product of  $x$  with  $y$  evaluates to either 0 or 1."

## 2. Mathematical Programming

---

We will state main general problem classes to be associated with in these notes. These are Linear Programming (LP), Mixed-Integer Linear Programming (MILP), Non-Linear Programming (NLP), and Mixed-Integer Non-Linear Programming (MINLP).



© problem-class-diagram<sup>1</sup>  
**Figure 2.1: problem-class-diagram**

Along with each problem class, we will associate a complexity class for the general version of the problem. See chapter 17 for a discussion of complexity classes. Although we will often state that input data for a problem comes from  $\mathbb{R}$ , when we discuss complexity of such a problem, we actually mean that the data is rational, i.e., from  $\mathbb{Q}$ , and is given in binary encoding.

### 2.1 Linear Programming (LP)

---

Some linear programming background, theory, and examples will be provided in ??.

**Linear Programming (LP):**

*Polynomial time (P)*

<sup>1</sup>problem-class-diagram, from problem-class-diagram. problem-class-diagram, problem-class-diagram.

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{2.1}$$

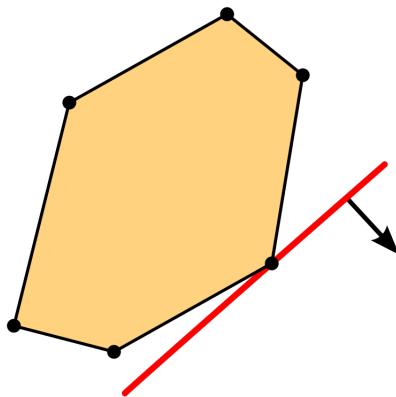
Linear programming can come in several forms, whether we are maximizing or minimizing, or if the constraints are  $\leq$ ,  $=$  or  $\geq$ . One form commonly used is *Standard Form* given as

### Linear Programming (LP) Standard Form:

#### *Polynomial time (P)*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *linear programming* problem in *standard form* is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.2}$$



© wiki/File/linear-programming.png<sup>2</sup>

**Figure 2.2: Linear programming constraints and objective.**

Figure 2.2

### Exercise 2.1:

Start with a problem in form given as (2.1) and convert it to standard form (2.2) by adding at most  $m$  many new variables and by enlarging the constraint matrix  $A$  by at most  $m$  new columns.

<sup>2</sup>wiki/File/linear-programming.png, from wiki/File/linear-programming.png. wiki/File/linear-programming.png, wiki/File/linear-programming.png.

## 2.2 Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming will be the focus of Sections 16, 19, 20, and ???. Recall that the notation  $\mathbb{Z}$  means the set of integers and the set  $\mathbb{R}$  means the set of real numbers. The first problem of interest here is a *binary integer program* (BIP) where all  $n$  variables are binary (either 0 or 1).

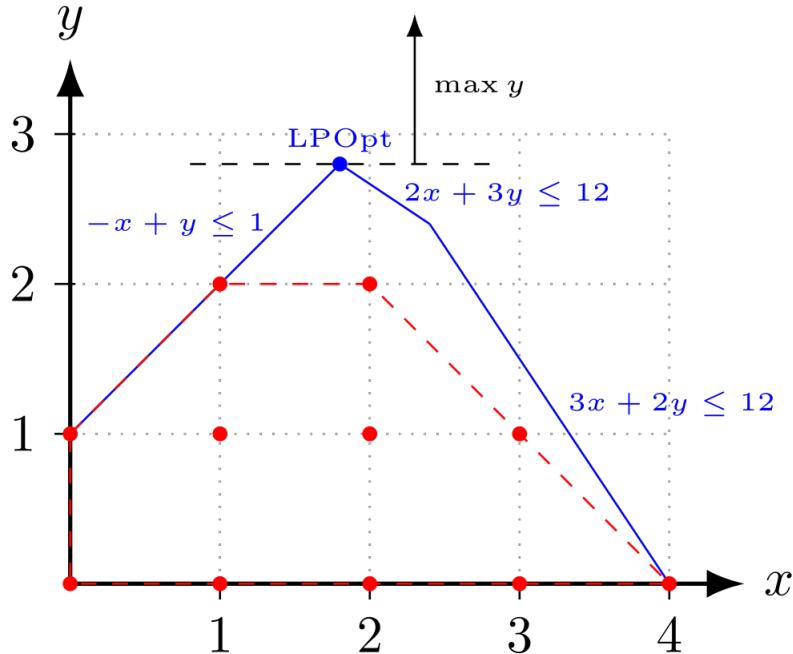
### Binary Integer programming (BIP):

#### *NP-Complete*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned} \tag{2.1}$$

A slightly more general class is the class of *Integer Linear Programs* (ILP). Often this is referred to as *Integer Program* (IP), although this term could leave open the possibility of non-linear parts.



© wiki/File/integer-programming.png<sup>3</sup>

Figure 2.3: Comparing the LP relaxation to the IP solutions.

Figure 2.3

<sup>3</sup>wiki/File/integer-programming.png, from wiki/File/integer-programming.png. wiki/File/integer-programming.png, wiki/File/integer-programming.png.

**Integer Linear Programming (ILP):***NP-Complete*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *integer linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned} \tag{2.2}$$

An even more general class is *Mixed-Integer Linear Programming (MILP)*. This is where we have  $n$  integer variables  $x_1, \dots, x_n \in \mathbb{Z}$  and  $d$  continuous variables  $x_{n+1}, \dots, x_{n+d} \in \mathbb{R}$ . Succinctly, we can write this as  $x \in \mathbb{Z}^n \times \mathbb{R}^d$ , where  $\times$  stands for the *cross-product* between two spaces.

Below, the matrix  $A$  now has  $n + d$  columns, that is,  $A \in \mathbb{R}^{m \times n+d}$ . Also note that we have not explicitly enforced non-negativity on the variables. If there are non-negativity restrictions, this can be assumed to be a part of the inequality description  $Ax \leq b$ .

**Mixed-Integer Linear Programming (MILP):***NP-Complete*

Given a matrix  $A \in \mathbb{R}^{m \times (n+d)}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^{n+d}$ , the *mixed-integer linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \times \mathbb{R}^d \end{aligned} \tag{2.3}$$

## 2.3 Non-Linear Programming (NLP)

---

**NLP:***NP-Hard*

Given a function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and other functions  $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *nonlinear programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{2.1}$$

Nonlinear programming can be separated into convex programming and non-convex programming. These two are very different beasts and it is important to distinguish between the two.

### 2.3.1. Convex Programming

---

Here the functions are all **convex**!

#### Convex Programming:

*Polynomial time (P)* (typically)

Given a convex function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{2.2}$$

#### Example 2.2

Convex programming is a generalization of linear programming. This can be seen by letting  $f(x) = c^\top x$  and  $f_i(x) = A_i x - b_i$ .

### 2.3.2. Non-Convex Non-linear Programming

---

When the function  $f$  or functions  $f_i$  are non-convex, this becomes a non-convex nonlinear programming problem. There are a few complexity issues with this.

**IP AS NLP** As seen above, quadratic constraints can be used to create a feasible region with discrete solutions. For example

$$x(1-x) = 0$$

has exactly two solutions:  $x = 0, x = 1$ . Thus, quadratic constraints can be used to model binary constraints.

#### Binary Integer programming (BIP) as a NLP:

*NP-Hard*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \\ & x_i(1 - x_i) = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \tag{2.3}$$

## 2.4 Mixed-Integer Non-Linear Programming (MINLP)

---

### 2.4.1. Convex Mixed-Integer Non-Linear Programming

---

### 2.4.2. Non-Convex Mixed-Integer Non-Linear Programming

---

# **Part I**

# **Linear Programming**



# 3. Linear Programming

---

## Outcomes

### A Generic Linear Program (LP)

#### Decision Variables:

$x_i$  : continuous variables ( $x_i \in \mathbb{R}$ , i.e., a real number),  $\forall i = 1, \dots, 3$ .

#### Parameters (known input parameters):

$c_i$  : cost coefficients  $\forall i = 1, \dots, n$

$a_{ij}$  : constraint coefficients  $\forall i = 1, \dots, n, j = 1, \dots, m$

$b_j$  : right hand side coefficient for constraint  $j$ ,  $j = 1, \dots, m$

The problem we will consider is

$$\begin{aligned} \max \quad & z = c_1x_1 + \dots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + \dots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + \dots + a_{mn}x_n \leq b_m \end{aligned} \tag{3.1}$$

For example, in 3 variables and 4 constraints this could look like the following. The following example considers other types of constraints, i.e.,  $\geq$  and  $=$ . We will show how all these forms can be converted later.

#### Decision Variables:

$x_i$  : continuous variables ( $x_i \in \mathbb{R}$ , i.e., a real number),  $\forall i = 1, \dots, 3$ .

#### Parameters (known input parameters):

$c_i$  : cost coefficients  $\forall i = 1, \dots, 3$

$a_{ij}$  : constraint coefficients  $\forall i = 1, \dots, 3, j = 1, \dots, 4$

$b_j$  : right hand side coefficient for constraint  $j$ ,  $j = 1, \dots, 4$

$$\text{Min } z = c_1x_1 + c_2x_2 + c_3x_3 \tag{3.2}$$

$$\text{s.t. } a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \geq b_1 \tag{3.3}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \leq b_2 \tag{3.4}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \tag{3.5}$$

$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 \geq b_4 \tag{3.6}$$

$$x_1 \geq 0, x_2 \leq 0, x_3 \text{ urs.} \tag{3.7}$$

**Definition 3.1: Linear Function**

A function  $z : \mathbb{R}^n \rightarrow \mathbb{R}$  is linear if there are constants  $c_1, \dots, c_n \in \mathbb{R}$  so that:

$$z(x_1, \dots, x_n) = c_1x_1 + \dots + c_nx_n \quad (3.8)$$

**Lemma 3.2: Linear Function**

If  $z : \mathbb{R}^n \rightarrow \mathbb{R}$  is linear then for all  $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$  and for all scalar constants  $\alpha \in \mathbb{R}$  we have:

$$z(\mathbf{x}_1 + \mathbf{x}_2) = z(\mathbf{x}_1) + z(\mathbf{x}_2) \quad (3.9)$$

$$z(\alpha\mathbf{x}_1) = \alpha z(\mathbf{x}_1) \quad (3.10)$$

**Exercise 3.3**

Prove Lemma 3.

For the time being, we will eschew the general form and focus exclusively on linear programming problems with two variables. Using this limited case, we will develop a graphical method for identifying optimal solutions, which we will generalize later to problems with arbitrary numbers of variables.

**Example 3.4: Toy Maker**

Consider the problem of a toy company that produces toy planes and toy boats. The toy company can sell its planes for \$10 and its boats for \$8 dollars. It costs \$3 in raw materials to make a plane and \$2 in raw materials to make a boat. A plane requires 3 hours to make and 1 hour to finish while a boat requires 1 hour to make and 2 hours to finish. The toy company knows it will not sell anymore than 35 planes per week. Further, given the number of workers, the company cannot spend anymore than 160 hours per week finishing toys and 120 hours per week making toys. The company wishes to maximize the profit it makes by choosing how much of each toy to produce.

We can represent the profit maximization problem of the company as a linear programming problem. Let  $x_1$  be the number of planes the company will produce and let  $x_2$  be the number of boats the company will produce. The profit for each plane is  $\$10 - \$3 = \$7$  per plane and the profit for each boat is  $\$8 - \$2 = \$6$  per boat. Thus the total profit the company will make is:

$$z(x_1, x_2) = 7x_1 + 6x_2 \quad (3.11)$$

The company can spend no more than 120 hours per week making toys and since a plane takes 3 hours to make and a boat takes 1 hour to make we have:

$$3x_1 + x_2 \leq 120 \quad (3.12)$$

Likewise, the company can spend no more than 160 hours per week finishing toys and since it takes

1 hour to finish a plane and 2 hour to finish a boat we have:

$$x_1 + 2x_2 \leq 160 \quad (3.13)$$

Finally, we know that  $x_1 \leq 35$ , since the company will make no more than 35 planes per week. Thus the complete linear programming problem is given as:

$$\left\{ \begin{array}{l} \max z(x_1, x_2) = 7x_1 + 6x_2 \\ \text{s.t. } 3x_1 + x_2 \leq 120 \\ x_1 + 2x_2 \leq 160 \\ x_1 \leq 35 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{array} \right. \quad (3.14)$$

### Exercise 3.5: Chemical Manufacturing

A chemical manufacturer produces three chemicals: A, B and C. These chemical are produced by two processes: 1 and 2. Running process 1 for 1 hour costs \$4 and yields 3 units of chemical A, 1 unit of chemical B and 1 unit of chemical C. Running process 2 for 1 hour costs \$1 and produces 1 units of chemical A, and 1 unit of chemical B (but none of Chemical C). To meet customer demand, at least 10 units of chemical A, 5 units of chemical B and 3 units of chemical C must be produced daily. Assume that the chemical manufacturer wants to minimize the cost of production. Develop a linear programming problem describing the constraints and objectives of the chemical manufacturer. [Hint: Let  $x_1$  be the amount of time Process 1 is executed and let  $x_2$  be amount of time Process 2 is executed. Use the coefficients above to express the cost of running Process 1 for  $x_1$  time and Process 2 for  $x_2$  time. Do the same to compute the amount of chemicals A, B, and C that are produced.]

## 3.1 Modeling Assumptions in Linear Programming

### Outcomes

1. Address crucial assumptions when choosing to model a problem with linear programming.

Inspecting Example 3 (or the more general Problem 3.1) we can see there are several assumptions that must be satisfied when using a linear programming model. We enumerate these below:

**Proportionality Assumption** A problem can be phrased as a linear program only if the contribution to the objective function *and* the left-hand-side of each constraint by each decision variable  $(x_1, \dots, x_n)$  is proportional to the value of the decision variable.

**Additivity Assumption** A problem can be phrased as a linear programming problem only if the contribution to the objective function *and* the left-hand-side of each constraint by any decision variable  $x_i$  ( $i = 1, \dots, n$ ) is completely independent of any other decision variable  $x_j$  ( $j \neq i$ ) and additive.

**Divisibility Assumption** A problem can be phrased as a linear programming problem only if the quantities represented by each decision variable are infinitely divisible (i.e., fractional answers make sense).

**Certainty Assumption** A problem can be phrased as a linear programming problem only if the coefficients in the objective function and constraints are known with certainty.

The first two assumptions simply assert (in English) that both the objective function and functions on the left-hand-side of the (in)equalities in the constraints are linear functions of the variables  $x_1, \dots, x_n$ .

The third assumption asserts that a valid optimal answer could contain fractional values for decision variables. It's important to understand how this assumption comes into play—even in the toy making example. Many quantities can be divided into non-integer values (ounces, pounds etc.) but many other quantities cannot be divided. For instance, can we really expect that it's reasonable to make  $\frac{1}{2}$  a plane in the toy making example? When values must be constrained to true integer values, the linear programming problem is called an *integer programming problem*. These problems are outside the scope of this course, but there is a vast literature dealing with them [PS98, WN99]. For many problems, particularly when the values of the decision variables may become large, a fractional optimal answer could be obtained and then rounded to the nearest integer to obtain a reasonable answer. For example, if our toy problem were re-written so that the optimal answer was to make 1045.3 planes, then we could round down to 1045.

The final assumption asserts that the coefficients (e.g., profit per plane or boat) is known with absolute certainty. In traditional linear programming, there is no lack of knowledge about the make up of the objective function, the coefficients in the left-hand-side of the constraints or the bounds on the right-hand-sides of the constraints. There is a literature on *stochastic programming* [KW94, BN02] that relaxes some of these assumptions, but this too is outside the scope of the course.

### Exercise 3.6

In a short sentence or two, discuss whether the problem given in Example 3 meets all of the assumptions of a scenario that can be modeled by a linear programming problem. Do the same for Exercise 3. [Hint: Can you make  $\frac{2}{3}$  of a toy? Can you run a process for  $\frac{1}{3}$  of an hour?]

### Exercise 3.7: Stochastic Objective

Suppose the costs are not known with certainty but instead a probability distribution for each value of  $c_i$  ( $i = 1, \dots, n$ ) is known. Suggest a way of constructing a linear program from the probability distributions.

[Hint: Suppose I tell you that I'll give you a uniformly random amount of money between \$1 and \$2. How much money do you expect to receive? Use the same reasoning to answer the question.]

## 3.2 Examples

---

### Outcomes

- A. Learn how to format a linear optimization problem.
- B. Identify and understand common classes of linear optimization problems.

We will begin with a few examples, and then discuss specific problem types that occur often.

**Example 3.8: Production Problem**

You have 21 units of transparent aluminum alloy (TAA), LazWeld1, a joining robot leased for 23 hours, and CrumCut1, a cutting robot leased for 17 hours of aluminum cutting. You also have production code for a bookcase, desk, and cabinet, along with commitments to buy any of these you can produce for \$18, \$16, and \$10 apiece, respectively. A bookcase requires 2 units of TAA, 3 hours of joining, and 1 hour of cutting, a desk requires 2 units of TAA, 2 hours of joining, and 2 hour of cutting, and a cabinet requires 1 unit of TAA, 2 hours of joining, and 1 hour of cutting. Formulate an LP to maximize your revenue given your current resources.

**Solution.****Sets:**

- The types of objects = { bookcase, desk, cabinet}.

**Parameters:**

- Purchase cost of each object
- Units of TAA needed for each object
- Hours of joining needed for each object
- Hours of cutting needed for each object
- Hours of TAA, Joining, and Cutting available on robots

**Decision variables:**

$x_i$  : number of units of product  $i$  to produce,  
for all  $i$  =bookcase, desk, cabinet.

**Objective and Constraints:**

$$\begin{aligned}
 \max z &= 18x_1 + 16x_2 + 10x_3 && \text{(profit)} \\
 2x_1 + 2x_2 + 1x_3 &\leq 21 && \text{(TAA)} \\
 3x_1 + 2x_2 + 2x_3 &\leq 23 && \text{(LazWeld1)} \\
 1x_1 + 2x_2 + 1x_3 &\leq 17 && \text{(CrumCut1)} \\
 x_1, x_2, x_3 &\geq 0.
 \end{aligned}$$

**Example 3.9: The Diet Problem**

In the future (as envisioned in a bad 70's science fiction film) all food is in tablet form, and there are four types, green, blue, yellow, and red. A balanced, futuristic diet requires, at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D. Formulate an LP that ensures a balanced diet at the minimum possible cost.

Tablet	Iron	B	C	D	Cost (\$)
green (1)	6	6	7	4	1.25
blue (2)	4	5	4	9	1.05
yellow (3)	5	2	5	6	0.85
red (4)	3	6	3	2	0.65

**Solution.** Now we formulate the problem: Sets:

- Set of tablets  $\{1, 2, 3, 4\}$

Parameters:

- Iron in each tablet
- Vitamin B in each tablet
- Vitamin C in each tablet
- Vitamin D in each tablet
- Cost of each tablet

Decision variables:

$x_i$  : number of tablet of type  $i$  to include in the diet,  $\forall i \in \{1, 2, 3, 4\}$ .

Objective and Constraints:

$$\begin{aligned}
 \text{Min } z &= 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4 \\
 \text{s.t. } 6x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 20 \\
 6x_1 + 5x_2 + 2x_3 + 6x_4 &\geq 25 \\
 7x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 30 \\
 4x_1 + 9x_2 + 6x_3 + 2x_4 &\geq 15 \\
 x_1, x_2, x_3, x_4 &\geq 0.
 \end{aligned}$$



### Example 3.10: The Next Diet Problem

Progress is important, and our last problem had too many tablets, so we are going to produce a single, purple, 10 gram tablet for our futuristic diet requires, which are at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D, and 2000 calories. The tablet is made from blending 4 nutritious chemicals; the following table shows the units of our nutrients per, and cost of, grams of each chemical. Formulate an LP that ensures a balanced diet at the minimum possible cost.

**Solution.** Sets:

- Set of chemicals  $\{1, 2, 3, 4\}$

Tablet	Iron	B	C	D	Calories	Cost (\$)
Chem 1	6	6	7	4	1000	1.25
Chem 2	4	5	4	9	250	1.05
Chem 3	5	2	5	6	850	0.85
Chem 4	3	6	3	2	750	0.65

Parameters:

- Iron in each chemical
- Vitamin B in each chemical
- Vitamin C in each chemical
- Vitamin D in each chemical
- Cost of each chemical

Decision variables:

$x_i$  : grams of chemical  $i$  to include in the purple tablet,  $\forall i = 1, 2, 3, 4$ .

Objective and Constraints:

$$\begin{aligned}
 \text{Min } z &= 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4 \\
 \text{s.t. } 6x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 20 \\
 6x_1 + 5x_2 + 2x_3 + 6x_4 &\geq 25 \\
 7x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 30 \\
 4x_1 + 9x_2 + 6x_3 + 2x_4 &\geq 15 \\
 1000x_1 + 250x_2 + 850x_3 + 750x_4 &\geq 2000 \\
 x_1 + x_2 + x_3 + x_4 &= 10 \\
 x_1, x_2, x_3, x_4 &\geq 0.
 \end{aligned}$$



**Example 3.11: Work Scheduling Problem**

You are the manager of LP Burger. The following table shows the minimum number of employees required to staff the restaurant on each day of the week. Each employee must work for five consecutive days. Formulate an LP to find the minimum number of employees required to staff the restaurant.

Day of Week	Workers Required
1 = Monday	6
2 = Tuesday	4
3 = Wednesday	5
4 = Thursday	4
5 = Friday	3
6 = Saturday	7
7 = Sunday	7

**Solution.** Decision variables:

Decision variables:

$x_i$  : the number of workers that start 5 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$

$$\begin{aligned}
 \text{Min } z &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\
 \text{s.t. } x_1 + x_4 + x_5 + x_6 + x_7 &\geq 6 \\
 x_2 + x_5 + x_6 + x_7 + x_1 &\geq 4 \\
 x_3 + x_6 + x_7 + x_1 + x_2 &\geq 5 \\
 x_4 + x_7 + x_1 + x_2 + x_3 &\geq 4 \\
 x_5 + x_1 + x_2 + x_3 + x_4 &\geq 3 \\
 x_6 + x_2 + x_3 + x_4 + x_5 &\geq 7 \\
 x_7 + x_3 + x_4 + x_5 + x_6 &\geq 7 \\
 x_1, x_2, x_3, x_4, x_5, x_6, x_7 &\geq 0.
 \end{aligned}$$

The solution is as follows:

LP Solution	IP Solution
$z_{LP} = 7.333$	$z_I = 8.0$
$x_1 = 0$	$x_1 = 0$
$x_2 = 0.333$	$x_2 = 0$
$x_3 = 1$	$x_3 = 0$
$x_4 = 2.333$	$x_4 = 3$
$x_5 = 0$	$x_5 = 0$
$x_6 = 3.333$	$x_6 = 4$
$x_7 = 0.333$	$x_7 = 1$



**Example 3.12: LP Burger - extended**

*LP Burger has changed its policy, and allows, at most, two part time workers, who work for two consecutive days in a week. Formulate this problem.*

**Solution.** Decision variables:

$x_i$  : the number of workers that start 5 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$

$y_i$  : the number of workers that start 2 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$ .

$$\begin{aligned} \text{Min } z &= 5(x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7) \\ &\quad + 2(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7) \\ \text{s.t. } x_1 + x_4 + x_5 + x_6 + x_7 + y_1 + y_7 &\geq 6 \\ x_2 + x_5 + x_6 + x_7 + x_1 + y_2 + y_1 &\geq 4 \\ x_3 + x_6 + x_7 + x_1 + x_2 + y_3 + y_2 &\geq 5 \\ x_4 + x_7 + x_1 + x_2 + x_3 + y_4 + y_3 &\geq 4 \\ x_5 + x_1 + x_2 + x_3 + x_4 + y_5 + y_4 &\geq 3 \\ x_6 + x_2 + x_3 + x_4 + x_5 + y_6 + y_5 &\geq 7 \\ x_7 + x_3 + x_4 + x_5 + x_6 + y_7 + y_6 &\geq 7 \\ y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 &\leq 2 \\ x_i &\geq 0, y_i \geq 0, \forall i = 1, \dots, 7. \end{aligned}$$

**3.2.1. Knapsack Problem****3.2.2. Work Scheduling****3.2.3. Assignment Problem**

Consider the assignment of  $n$  teams to  $n$  projects, where each team ranks the projects, where their favorite project is given a rank of  $n$ , their next favorite  $n - 1$ , and their least favorite project is given a rank of 1. The assignment problem is formulated as follows (we denote ranks using the  $R$ -parameter):

Variables:

$x_{ij}$  : 1 if project  $i$  assigned to team  $j$ , else 0.

$$\begin{aligned} \text{Max } z &= \sum_{i=1}^n \sum_{j=1}^n R_{ij} x_{ij} \\ \text{s.t. } &\sum_{i=1}^n x_{ij} = 1, \quad \forall j = 1, \dots, n \\ &\sum_{j=1}^n x_{ij} = 1, \quad \forall i = 1, \dots, n \\ &x_{ij} \geq 0, \quad \forall i = 1, \dots, n, j = 1, \dots, n. \end{aligned}$$

The assignment problem has an integrality property, such that if we remove the binary restriction on the  $x$  variables (now just non-negative, i.e.,  $x_{ij} \geq 0$ ) then we still get binary assignments, despite the fact that it is now an LP. This property is very interesting and useful. Of course, the objective function might not quite what we want, we might be interested ensuring that the team with the worst assignment is as good as possible (a fairness criteria). One way of doing this is to modify the assignment problem using a max-min objective:

### Max-min Assignment-like Formulation

$$\begin{aligned} \text{Max } z &= \sum_{i=1}^n \sum_{j=1}^n R_{ij} x_{ij} \\ \text{s.t. } &\sum_{i=1}^n x_{ij} = 1, \quad \forall j = 1, \dots, n \\ &\sum_{j=1}^n x_{ij} = 1, \quad \forall i = 1, \dots, n \\ &x_{ij} \geq 0, \quad \forall i = 1, \dots, n, j = 1, \dots, n \\ &z \leq \sum_{i=1}^n R_{ij} x_{ij}, \quad \forall j = 1, \dots, n. \end{aligned}$$

Does this formulation have the integrality property (it is not an assignment problem)? Consider a very simple example where two teams are to be assigned to two projects and the teams give the projects the following rankings: Both teams prefer Project 2. For both problems, if we remove the binary restriction on

	Project 1	Project 2
Team 1	2	1
Team 2	2	1

the  $x$ -variable, they can take values between (and including) zero and one. For the assignment problem the optimal solution will have  $z = 3$ , and fractional  $x$ -values will not improve  $z$ . For the max-min assignment problem this is not the case, the optimal solution will have  $z = 1.5$ , which occurs when each team is assigned half of each project (i.e., for Team 1 we have  $x_{11} = 0.5$  and  $x_{21} = 0.5$ ).

### 3.2.4. Multi period Models

---

Fill in this subsection

#### 3.2.4.1. Production Planning

---

#### 3.2.4.2. Crop Planning

---

### 3.2.5. Mixing Problems

---

### 3.2.6. Financial Planning

---

Fill in this subsection

### 3.2.7. Network Flow

---

#### Resources

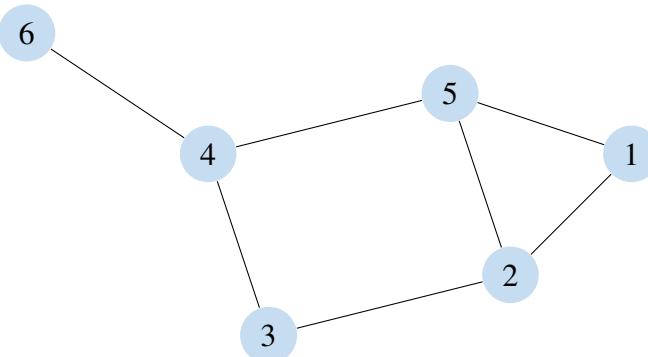
- MIT - CC BY NC SA 4.0 license
- Slides for Algorithms book by Kleinberg-Tardos

To begin a discussion on Network flow, we first need to discuss graphs.

#### 3.2.7.1. Graphs

---

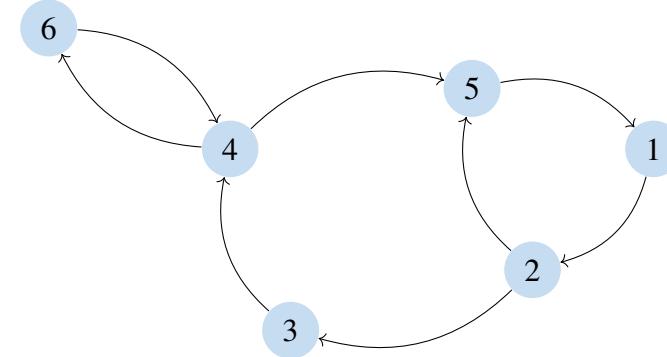
A graph  $G = (V, E)$  is defined by a set of vertices  $V$  and a set of edges  $E$  that contains pairs of vertices. For example, the following graph  $G$  can be described by the vertex set  $V = \{1, 2, 3, 4, 5, 6\}$  and the edge set  $E = \{(4, 6), (4, 5), (5, 1), (1, 2), (2, 5), (2, 3), (3, 4)\}$ .



In an undirected graph, we do not distinguish the direction of the edge. That is, for two vertices  $i, j \in V$ , we can equivalently write  $(i, j)$  or  $(j, i)$  to represent the edge.

Alternatively, we will want to consider directed graphs. We denote these as  $G = (V, \mathcal{A})$  where  $\mathcal{A}$  is a set of arcs where an arc is a directed edge.

For example, the following directed graph  $G$  can be described by the vertex set  $V = \{1, 2, 3, 4, 5, 6\}$  and the edge set  $\mathcal{A} = \{(4, 6), (6, 4), (4, 5), (5, 1), (1, 2), (2, 5), (2, 3), (3, 4)\}$ .



**SETS** A finite network  $G$  is described by a finite set of vertices  $V$  and a finite set  $\mathcal{A}$  of arcs. Each arc  $(i, j)$  has two key attributes, namely its tail  $j \in V$  and its head  $i \in V$ .

We think of a (single) commodity as being allowed to "flow" along each arc, from its tail to its head.

**VARIABLES** Indeed, we have "flow" variables

$$x_{ij} := \text{amount of flow on arc}(i, j) \text{ from vertex } i \text{ to vertex } j,$$

for all  $(i, j) \in \mathcal{A}$ .

### 3.2.7.2. Maximum Flow Problem

---

$$\max \sum_{(s,i) \in \mathcal{A}} x_{si} \quad \text{max total flow from source} \quad (3.1)$$

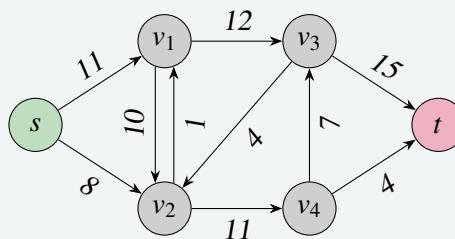
$$s.t. \quad \sum_{i:(i,v) \in \mathcal{A}} x_{iv} - \sum_{j:(v,j) \in \mathcal{A}} x_{vj} = 0 \quad v \in V \setminus \{s, t\} \quad (3.2)$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in \mathcal{A} \quad (3.3)$$

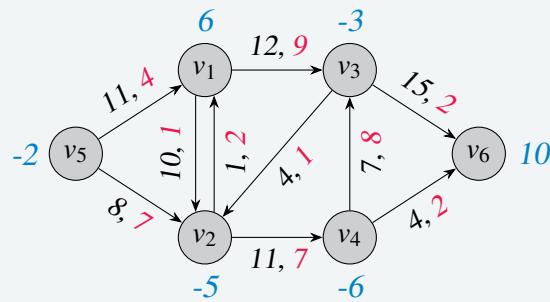
## SHORTEST PATH PROBLEM

$$\begin{aligned}
 & \text{minimize} && \sum_{u \rightarrow v} \ell_{u \rightarrow v} \cdot x_{u \rightarrow v} \\
 & \text{subject to} && \sum_u x_{u \rightarrow s} - \sum_w x_{s \rightarrow w} = 1 \\
 & && \sum_u x_{u \rightarrow t} - \sum_w x_{t \rightarrow w} = -1 \\
 & && \sum_u x_{u \rightarrow v} - \sum_w x_{v \rightarrow w} = 0 \quad \text{for every vertex } v \neq s, t \\
 & && x_{u \rightarrow v} \geq 0 \quad \text{for every edge } u \rightarrow v
 \end{aligned}$$

## Example 3.13: Max flow example



## Example 3.14: Min Cost Network Flow



## 3.2.7.3. Minimum Cost Network Flow

**PARAMETERS** We assume that flow on arc  $(i, j)$  should be non-negative and should not exceed

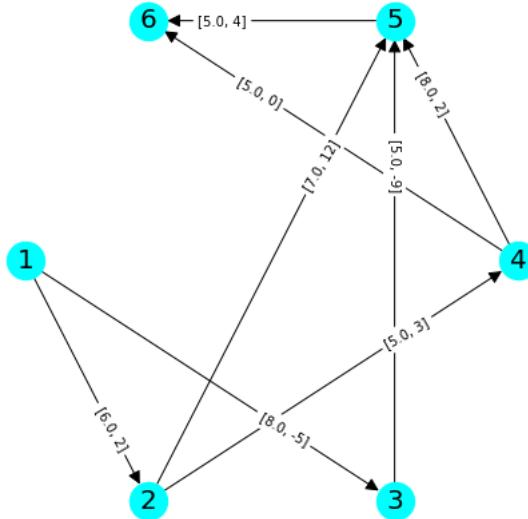
$$u_{ij} := \text{the flow upper bound on arc } (i, j),$$

for  $(i, j) \in \mathcal{A}$ . Associated with each arc  $(i, j)$  is a cost

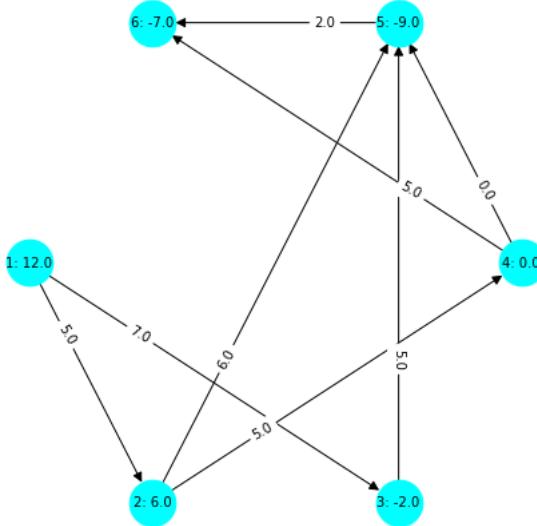
$$c_{ij} := \text{cost per-unit-flow on arc } (i, j),$$

<sup>1</sup>network-flow, from **network-flow**. **network-flow**, **network-flow**.

<sup>2</sup>network-flow-solution, from **network-flow-solution**. **network-flow-solution**, **network-flow-solution**.



© network-flow<sup>1</sup>  
**Figure 3.1: network-flow**



© network-flow-solution<sup>2</sup>  
**Figure 3.2: network-flow-solution**

for  $(i, j) \in \mathcal{A}$ . The (total) cost of the flow  $x$  is defined to be

$$\sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}.$$

We assume that we have further data for the nodes. Namely,

$$b_v := \text{the net supply at node } v,$$

for  $v \in V$ .

A flow is conservative if the net flow out of node  $v$ , minus the net flow into node  $v$ , is equal to the net supply at node  $v$ , for all nodes  $v \in V$ .

The (single-commodity min-cost) network-flow problem is to find a minimumcost conservative flow that is non-negative and respects the flow upper bounds on the arcs.

**OBJECTIVE AND CONSTRAINTS** We can formulate this as follows:

$$\begin{aligned} \min \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} & \quad \text{minimize cost} \\ \sum_{(i,v) \in \mathcal{A}} x_{iv} - \sum_{(v,i) \in \mathcal{A}} x_{vi} = b_v, & \quad \text{for all } v \in V, \quad \text{flow conservation} \\ 0 \leq x_{ij} \leq u_{ij}, & \quad \text{for all } (i,j) \in \mathcal{A}. \end{aligned}$$

### Theorem 3.15: Integrality of Network Flow

If the capacities and demands are all integer values, then there always exists an optimal solution to the LP that has integer values.

## 3.2.8. Multi-Commodity Network Flow

In the same vein as the Network Flow Problem

$$\begin{aligned} \min \sum_{k=1}^K \sum_{e \in \mathcal{A}} c_e^k x_e^k \\ \sum_{e \in \mathcal{A} : t(e)=v} x_e^k - \sum_{e \in \mathcal{A} : h(e)=v} x_e^k = b_v^k, \quad \text{for } v \in \mathcal{N}, k = 1, 2, \dots, K; \\ \sum_{k=0}^K x_e^k \leq u_e, \quad \text{for } e \in \mathcal{A}; \\ x_e^k \geq 0, \quad \text{for } e \in \mathcal{A}, k = 1, 2, \dots, K \end{aligned}$$

Notes:

K=1 is ordinary single-commodity network flow. Integer solutions for free when node-supplies and arc capacities are integer. K=2 example below with integer data gives a fractional basic optimum. This example doesn't have any feasible integer flow at all.

### Remark 3.16

Unfortunately, the same integrality theorem does not hold in the multi-commodity network flow problem. Nonetheless, if the quantities in each flow are very large, then the LP solution will likely be very close to an integer valued solution.

## 3.3 Modeling Tricks

---

### 3.3.1. Maximizing a minimum

---

When the constraints could be general, we will write  $x \in X$  to define general constraints. For instance, we could have  $X = \{x \in \mathbb{R}^n : Ax \leq b\}$  or  $X = \{x \in \mathbb{R}^n : Ax \leq b, x \in \mathbb{Z}^n\}$  or many other possibilities.

Consider the problem

$$\begin{aligned} & \max \quad \min\{x_1, \dots, x_n\} \\ \text{such that } & x \in X \end{aligned}$$

Having the minimum on the inside is inconvenient. To remove this, we just define a new variable  $y$  and enforce that  $y \leq x_i$  and then we maximize  $y$ . Since we are maximizing  $y$ , it will take the value of the smallest  $x_i$ . Thus, we can recast the problem as

$$\begin{aligned} & \max \quad y \\ \text{such that } & y \leq x_i \text{ for } i = 1, \dots, n \\ & x \in X \end{aligned}$$

#### Example 3.17: Minimizing an Absolute Value

Note that

$$|t| = \max(t, -t),$$

Thus, if we need to minimize  $|t|$  we can instead write

$$\min z \tag{3.1}$$

$$s.t. \tag{3.2}$$

$$t \leq z - t \leq z \tag{3.3}$$

## 3.4 Other examples

---

Food manfacturing - GUROBI

Optimization Methods in Finance - Corneujoles, Tütüncü

# 4. Graphically Solving Linear Programs

---

## Outcomes

- A. Learn how to plot the feasible region and the objective function.
- B. Identify and compute extreme points of the feasible region.
- C. Find the optimal solution(s) to a linear program graphically.
- D. Classify the type of result of the problem as infeasible, unbounded, unique optimal solution, or infinitely many optimal solutions.

Linear Programs (LP's) with two variables can be solved graphically by plotting the feasible region along with the level curves of the objective function.<sup>1</sup> We will show that we can find a point in the feasible region that maximizes the objective function using the level curves of the objective function.

We will begin with an easy example that is bounded and investigate the structure of the feasible region. We will then explore other examples.

## 4.1 Nonempty and Bounded Problem

---

Consider the problem

$$\begin{aligned} \max \quad & 2X + 5Y \\ \text{s.t.} \quad & X + 2Y \leq 16 \\ & 5X + 3Y \leq 45 \\ & X, Y \geq 0 \end{aligned}$$

We want to start by plotting the *feasible region*, that is, the set points  $(X, Y)$  that satisfy all the constraints.

We can plot this by first plotting the four lines

- $X + 2Y = 16$
- $5X + 3Y = 45$
- $X = 0$
- $Y = 0$

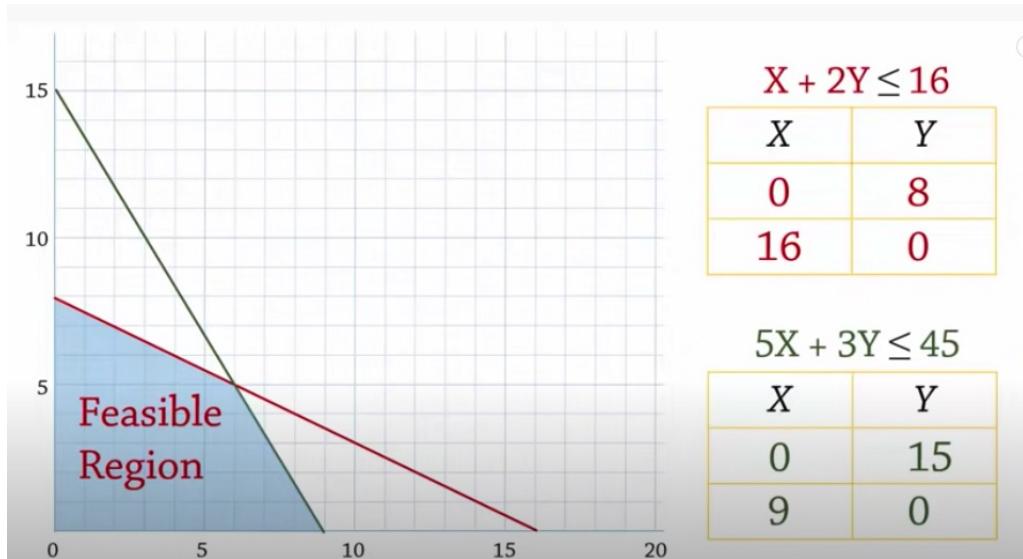
and then shading in the side of the space cut out by the corresponding inequality.

---

<sup>1</sup>Special thanks to Joshua Emmanuel and Christopher Griffin for sharing their content to help put this section together. Proper citations and references are forthcoming.



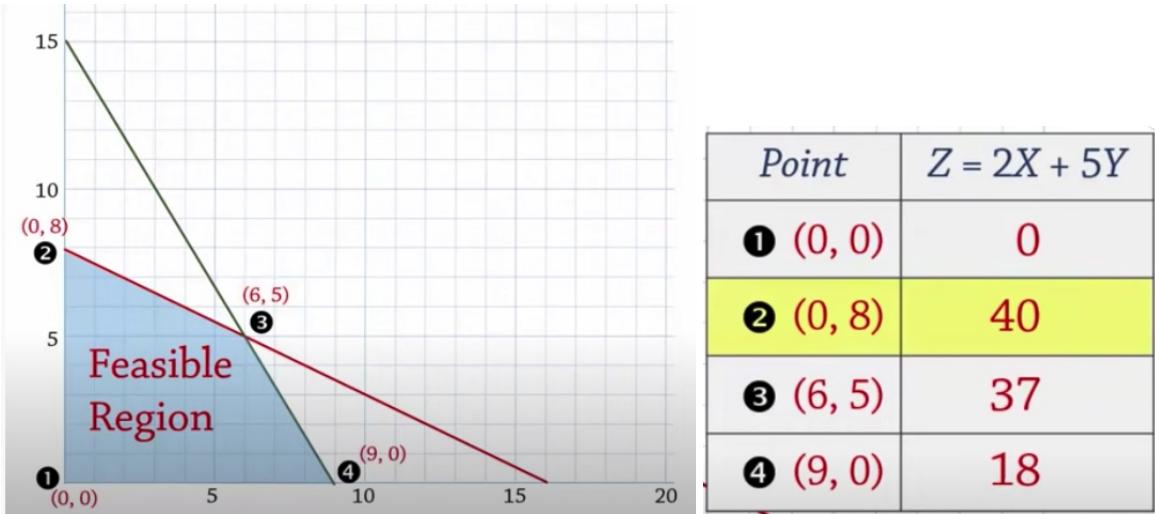
The resulting feasible region can then be shaded in as the region that satisfies all the inequalities.



Notice that the feasible region is nonempty (it has points that satisfy all the inequalities) and also that it is bounded (the feasible points don't continue infinitely in any direction).

We want to identify the *extreme points* (i.e., the corners) of the feasible region. Understanding these points will be critical to understanding the optimal solutions of the model. Notice that all extreme points can be computed by finding the intersection of 2 of the lines. But! Not all intersections of any two lines are feasible.

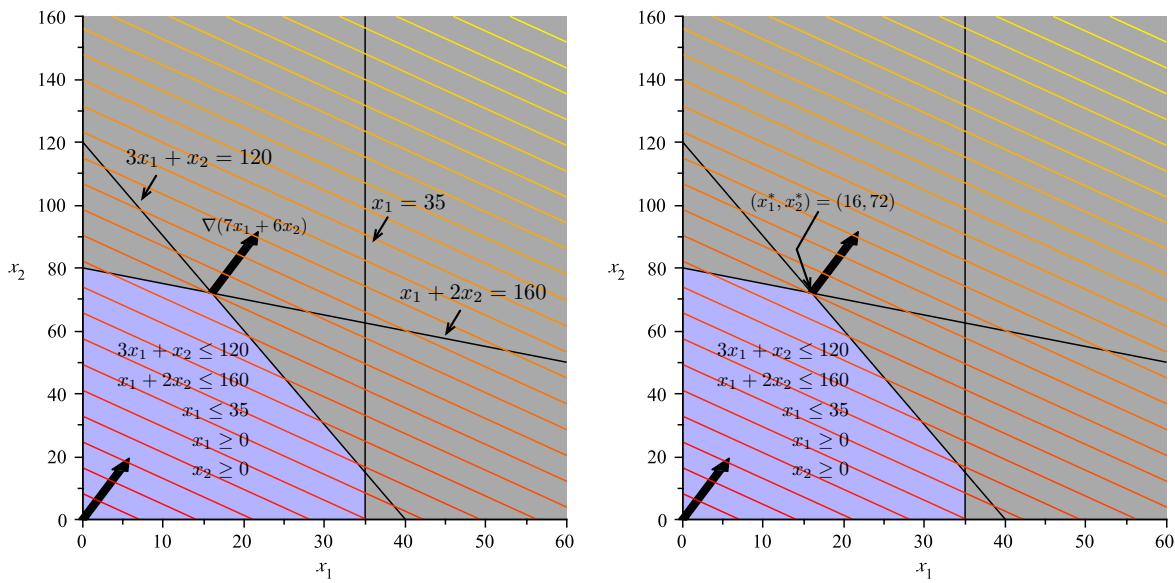
We will later use the terminology *basic feasible solution* for an extreme point of the feasible region, and *basic solution* as a point that is the intersection of 2 lines, but is actually infeasible (does not satisfy all the constraints).



### Theorem 4.1: Optimal Extreme Point

If the feasible region is nonempty and bounded, then there exists an optimal solution at an extreme point of the feasible region.

We will explore why this theorem is true, and also what happens when the feasible region does not satisfy the assumptions of either nonempty or bounded. We illustrate the idea first using the problem from Example 3.



**Figure 4.1: Feasible Region and Level Curves of the Objective Function:** The shaded region in the plot is the feasible region and represents the intersection of the five inequalities constraining the values of  $x_1$  and  $x_2$ . On the right, we see the optimal solution is the “last” point in the feasible region that intersects a level set as we move in the direction of increasing profit.

**Example 4.2: Continuation of Example 3**

*Let's continue the example of the Toy Maker begin in Example 3. Solve this problem graphically.*

**Solution.** To solve the linear programming problem graphically, begin by drawing the feasible region. This is shown in the blue shaded region of Figure 4.1.

After plotting the feasible region, the next step is to plot the level curves of the objective function. In our problem, the level sets will have the form:

$$7x_1 + 6x_2 = c \implies x_2 = \frac{-7}{6}x_1 + \frac{c}{6}$$

This is a set of parallel lines with slope  $-7/6$  and intercept  $c/6$  where  $c$  can be varied as needed. The level curves for various values of  $c$  are parallel lines. In Figure 4.1 they are shown in colors ranging from red to yellow depending upon the value of  $c$ . Larger values of  $c$  are more yellow.

To solve the linear programming problem, follow the level sets along the gradient (shown as the black arrow) until the last level set (line) intersects the feasible region. If you are doing this by hand, you can draw a single line of the form  $7x_1 + 6x_2 = c$  and then simply draw parallel lines in the direction of the gradient  $(7, 6)$ . At some point, these lines will fail to intersect the feasible region. The last line to intersect the feasible region will do so at a point that maximizes the profit. In this case, the point that maximizes  $z(x_1, x_2) = 7x_1 + 6x_2$ , subject to the constraints given, is  $(x_1^*, x_2^*) = (16, 72)$ .

Note the point of optimality  $(x_1^*, x_2^*) = (16, 72)$  is at a corner of the feasible region. This corner is formed by the intersection of the two lines:  $3x_1 + x_2 = 120$  and  $x_1 + 2x_2 = 160$ . In this case, the constraints

$$\begin{aligned}3x_1 + x_2 &\leq 120 \\x_1 + 2x_2 &\leq 160\end{aligned}$$

are both *binding*, while the other constraints are non-binding. In general, we will see that when an optimal solution to a linear programming problem exists, it will always be at the intersection of several binding constraints; that is, it will occur at a corner of a higher-dimensional polyhedron. ♠

We can now define an algorithm for identifying the solution to a linear programming problem in two variables with a *bounded* feasible region (see Algorithm 1):

---

**Algorithm 1** Algorithm for Solving a Two Variable Linear Programming Problem Graphically–Bounded Feasible Region, Unique Solution Case

---

#### **Algorithm for Solving a Linear Programming Problem Graphically**

##### *Bounded Feasible Region, Unique Solution*

1. Plot the feasible region defined by the constraints.
  2. Plot the level sets of the objective function.
  3. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
  4. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.
- 

The example linear programming problem presented in the previous section has a single optimal solution. In general, the following outcomes can occur in solving a linear programming problem:

1. The linear programming problem has a unique solution. (We've already seen this.)
2. There are infinitely many alternative optimal solutions.
3. There is no solution and the problem's objective function can grow to positive infinity for maximization problems (or negative infinity for minimization problems).
4. There is no solution to the problem at all.

Case 3 above can only occur when the feasible region is unbounded; that is, it cannot be surrounded by a ball with finite radius. We will illustrate each of these possible outcomes in the next four sections. We will prove that this is true in a later chapter.

## 4.2 Infinitely Many Optimal Solutions

---

It can happen that there is more than one solution. In fact, in this case, there are infinitely many optimal solutions. We'll study a specific linear programming problem with an infinite number of solutions by modifying the objective function in Example 3.

**Example 4.3: Toy Maker Alternative Solutions**

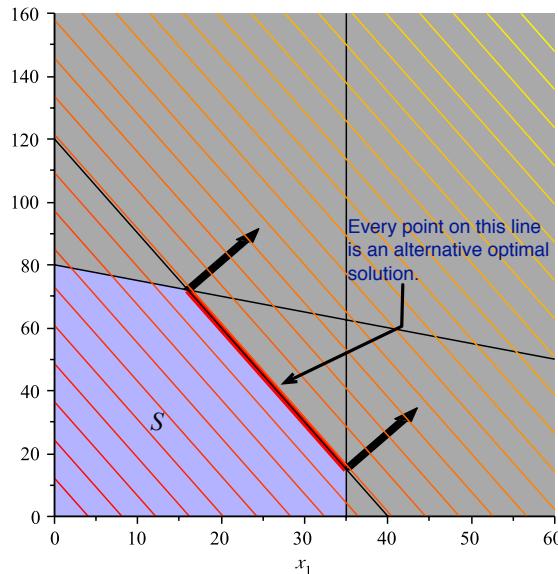
Suppose the toy maker in Example 3 finds that it can sell planes for a profit of \$18 each instead of \$7 each. The new linear programming problem becomes:

$$\left\{ \begin{array}{l} \max z(x_1, x_2) = 18x_1 + 6x_2 \\ \text{s.t. } 3x_1 + x_2 \leq 120 \\ \quad x_1 + 2x_2 \leq 160 \\ \quad x_1 \leq 35 \\ \quad x_1 \geq 0 \\ \quad x_2 \geq 0 \end{array} \right. \quad (4.1)$$

**Solution.** Applying our graphical method for finding optimal solutions to linear programming problems yields the plot shown in Figure 4.2. The level curves for the function  $z(x_1, x_2) = 18x_1 + 6x_2$  are *parallel* to one face of the polygon boundary of the feasible region. Hence, as we move further up and to the right in the direction of the gradient (corresponding to larger and larger values of  $z(x_1, x_2)$ ) we see that there is not *one* point on the boundary of the feasible region that intersects that level set with greatest value, but instead a side of the polygon boundary described by the line  $3x_1 + x_2 = 120$  where  $x_1 \in [16, 35]$ . Let:

$$S = \{(x_1, x_2) | 3x_1 + x_2 \leq 120, x_1 + 2x_2 \leq 160, x_1 \leq 35, x_1, x_2 \geq 0\}$$

that is,  $S$  is the feasible region of the problem. Then for any value of  $x_1^* \in [16, 35]$  and any value  $x_2^*$  so that  $3x_1^* + x_2^* = 120$ , we will have  $z(x_1^*, x_2^*) \geq z(x_1, x_2)$  for all  $(x_1, x_2) \in S$ . Since there are infinitely many values that  $x_1$  and  $x_2$  may take on, we see this problem has an infinite number of alternative optimal solutions.



**Figure 4.2:** An example of infinitely many alternative optimal solutions in a linear programming problem. The level curves for  $z(x_1, x_2) = 18x_1 + 6x_2$  are parallel to one face of the polygon boundary of the feasible region. Moreover, this side contains the points of greatest value for  $z(x_1, x_2)$  inside the feasible region. Any combination of  $(x_1, x_2)$  on the line  $3x_1 + x_2 = 120$  for  $x_1 \in [16, 35]$  will provide the largest possible value  $z(x_1, x_2)$  can take in the feasible region  $S$ .



#### Exercise 4.4

Use the graphical method for solving linear programming problems to solve the linear programming problem you defined in Exercise 3.

Based on the example in this section, we can modify our algorithm for finding the solution to a linear programming problem graphically to deal with situations with an infinite set of alternative optimal solutions (see Algorithm 2):

**Algorithm 2** Algorithm for Solving a Two Variable Linear Programming Problem Graphically–Bounded Feasible Region Case

---

### **Algorithm for Solving a Linear Programming Problem Graphically**

#### *Bounded Feasible Region*

1. Plot the feasible region defined by the constraints.
  2. Plot the level sets of the objective function.
  3. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
  4. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.
  5. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**
- 

#### **Exercise 4.5**

*Modify the linear programming problem from Exercise 3 to obtain a linear programming problem with an infinite number of alternative optimal solutions. Solve the new problem and obtain a description for the set of alternative optimal solutions. [Hint: Just as in the example,  $x_1$  will be bound between two values corresponding to a side of the polygon. Find those values and the constraint that is binding. This will provide you with a description of the form for any  $x_1^* \in [a, b]$  and  $x_2^*$  is chosen so that  $cx_1^* + dx_2^* = v$ , the point  $(x_1^*, x_2^*)$  is an alternative optimal solution to the problem. Now you fill in values for  $a, b, c, d$  and  $v$ .]*

---

## 4.3 Problems with No Solution

---

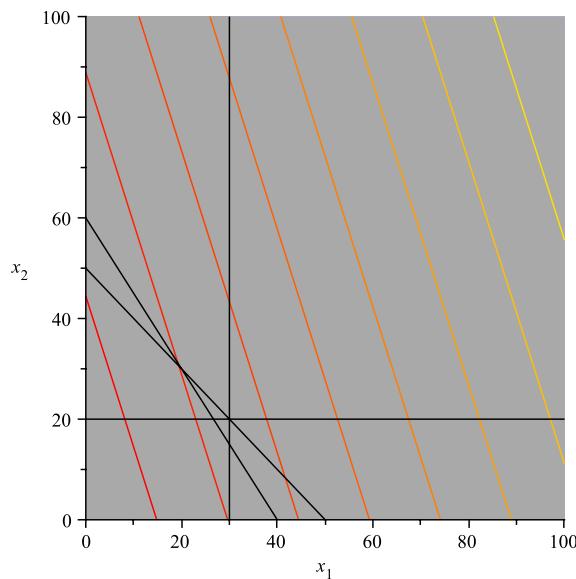
Recall for *any* mathematical programming problem, the feasible set or region is simply a subset of  $\mathbb{R}^n$ . If this region is empty, then there is no solution to the mathematical programming problem and the problem is said to be *over constrained*. In this case, we say that the problem is *infeasible*. We illustrate this case for linear programming problems with the following example.

**Example 4.6: Infeasible Problem**

Consider the following linear programming problem:

$$\left\{ \begin{array}{l} \max z(x_1, x_2) = 3x_1 + 2x_2 \\ \text{s.t. } \frac{1}{40}x_1 + \frac{1}{60}x_2 \leq 1 \\ \quad \frac{1}{50}x_1 + \frac{1}{50}x_2 \leq 1 \\ \quad x_1 \geq 30 \\ \quad x_2 \geq 20 \end{array} \right. \quad (4.1)$$

**Solution.** The level sets of the objective and the constraints are shown in Figure 4.3.



**Figure 4.3: A Linear Programming Problem with no solution.** The feasible region of the linear programming problem is empty; that is, there are no values for  $x_1$  and  $x_2$  that can simultaneously satisfy all the constraints. Thus, no solution exists.

The fact that the feasible region is empty is shown by the fact that in Figure 4.3 there is no blue region—i.e., all the regions are gray indicating that the constraints are not satisfiable. ♠

Based on this example, we can modify our previous algorithm for finding the solution to linear programming problems graphically (see Algorithm 3):

**Algorithm 3** Algorithm for Solving a Two Variable Linear Programming Problem Graphically–Bounded Feasible Region Case

#### Algorithm for Solving a Linear Programming Problem Graphically

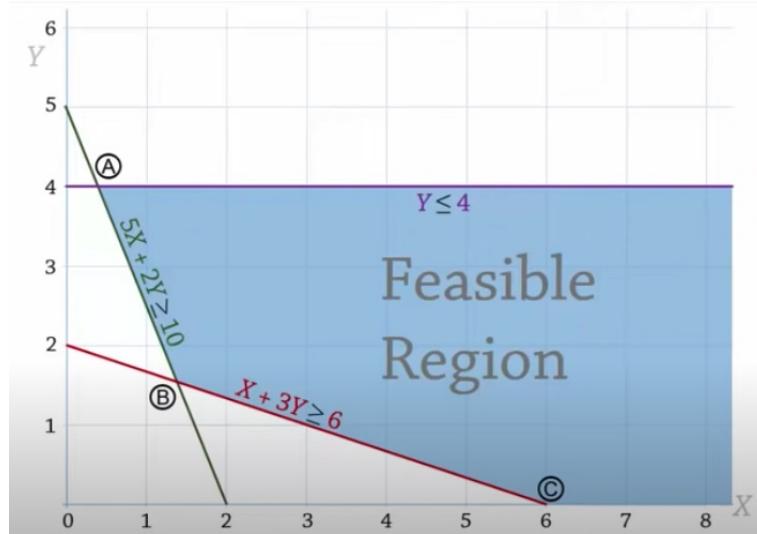
##### *Bounded Feasible Region*

1. Plot the feasible region defined by the constraints.
2. **If the feasible region is empty, then no solution exists.**
3. Plot the level sets of the objective function.
4. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
5. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.
6. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**

## 4.4 Problems with Unbounded Feasible Regions

Consider the problem

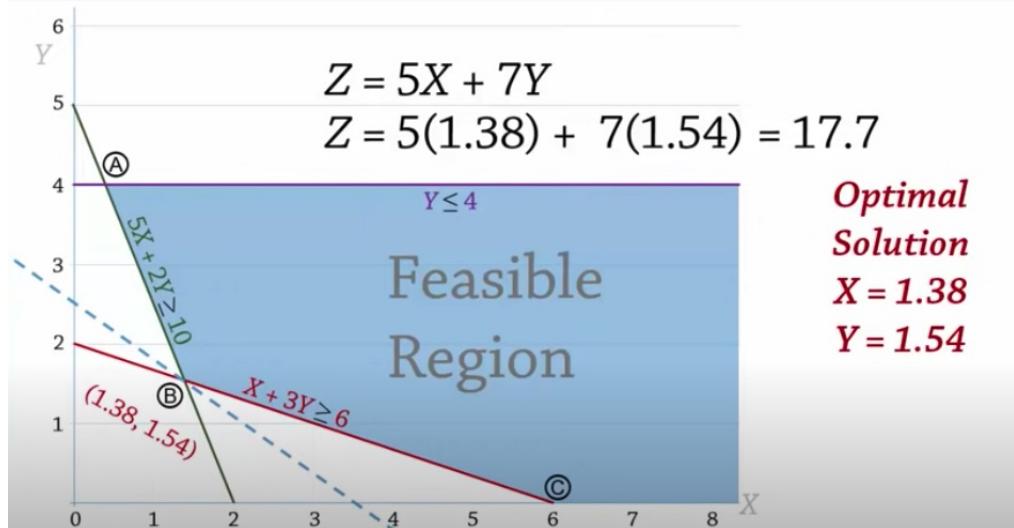
$$\begin{aligned} \min \quad & Z = 5X + 7Y \\ \text{s.t.} \quad & X + 3Y \geq 6 \\ & 5X + 2Y \geq 10 \\ & Y \leq 4 \\ & X, Y \geq 0 \end{aligned}$$



As you can see, the feasible region is *unbounded*. In particular, from any point in the feasible region, one can always find another feasible point by increasing the  $X$  coordinate (i.e., move to the right in the picture). However, this does not necessarily mean that the optimization problem is unbounded.

Indeed, the optimal solution is at the B, the extreme point in the lower left hand corner.

To do: add contours to plot to show extreme point is the optimal solution.



Consider however, if we consider a different problem where we try to maximize the objective

$$\begin{aligned} \max \quad & Z = 5X + 7Y \\ \text{s.t.} \quad & X + 3Y \geq 6 \\ & 5X + 2Y \geq 10 \\ & Y \leq 4 \\ & X, Y \geq 0 \end{aligned}$$

**Solution.** This optimization problem is unbounded! For example, notice that the point  $(X, Y) = (n, 0)$  is feasible for all  $n = 1, 2, 3, \dots$ . Then the objective function  $Z = 5n + 0$  follows the sequence 5, 10, 15, ..., which diverges to infinity. ♠

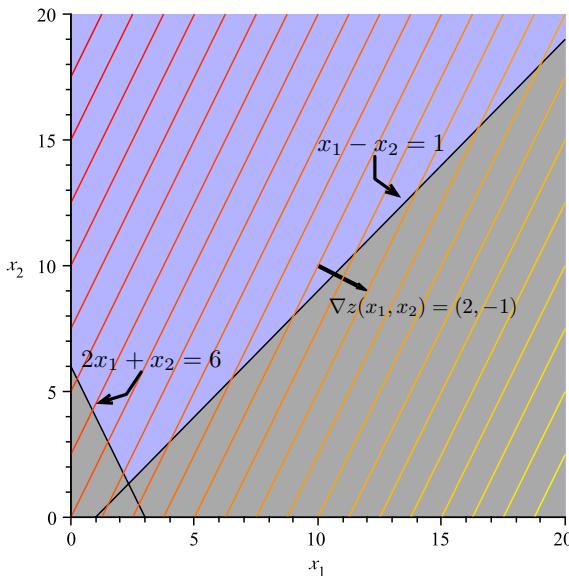
Again, we'll tackle the issue of linear programming problems with unbounded feasible regions by illustrating the possible outcomes using examples.

### Example 4.7

Consider the linear programming problem below:

$$\left\{ \begin{array}{l} \max z(x_1, x_2) = 2x_1 - x_2 \\ \text{s.t. } x_1 - x_2 \leq 1 \\ \quad 2x_1 + x_2 \geq 6 \\ \quad x_1, x_2 \geq 0 \end{array} \right. \quad (4.1)$$

**Solution.** The feasible region and level curves of the objective function are shown in Figure 4.4.



**Figure 4.4: A Linear Programming Problem with Unbounded Feasible Region:** Note that we can continue to make level curves of  $z(x_1, x_2)$  corresponding to larger and larger values as we move down and to the right. These curves will continue to intersect the feasible region for any value of  $v = z(x_1, x_2)$  we choose. Thus, we can make  $z(x_1, x_2)$  as large as we want and still find a point in the feasible region that will provide this value. Hence, the optimal value of  $z(x_1, x_2)$  subject to the constraints  $+\infty$ . That is, the problem is unbounded.

The feasible region in Figure 4.4 is clearly unbounded since it stretches upward along the  $x_2$  axis infinitely far and also stretches rightward along the  $x_1$  axis infinitely far, bounded below by the line  $x_1 - x_2 = 1$ . There is no way to enclose this region by a disk of finite radius, hence the feasible region is not bounded.

We can draw more level curves of  $z(x_1, x_2)$  in the direction of increase (down and to the right) as long as we wish. There will always be an intersection point with the feasible region because it is infinite. That is, these curves will continue to intersect the feasible region for any value of  $v = z(x_1, x_2)$  we choose. Thus, we can make  $z(x_1, x_2)$  as large as we want and still find a point in the feasible region that will provide this value. Hence, the largest value  $z(x_1, x_2)$  can take when  $(x_1, x_2)$  are in the feasible region is  $+\infty$ . That is, the problem is unbounded. ♠

Just because a linear programming problem has an unbounded feasible region does not imply that there is not a finite solution. We illustrate this case by modifying example 4.4.

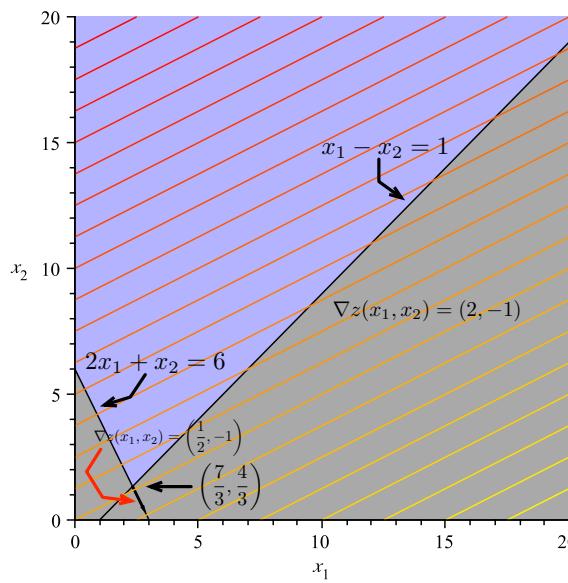
#### Example 4.8: Continuation of Example 4.4

Consider the linear programming problem from Example 4.4 with the new objective function:

$z(x_1, x_2) = (1/2)x_1 - x_2$ . Then we have the new problem:

$$\begin{cases} \max z(x_1, x_2) = \frac{1}{2}x_1 - x_2 \\ \text{s.t. } x_1 - x_2 \leq 1 \\ \quad 2x_1 + x_2 \geq 6 \\ \quad x_1, x_2 \geq 0 \end{cases} \quad (4.2)$$

**Solution.** The feasible region, level sets of  $z(x_1, x_2)$  and gradients are shown in Figure 4.5. In this case note, that the direction of increase of the objective function is *away* from the direction in which the feasible region is unbounded (i.e., downward). As a result, the point in the feasible region with the largest  $z(x_1, x_2)$  value is  $(7/3, 4/3)$ . Again this is a vertex: the binding constraints are  $x_1 - x_2 = 1$  and  $2x_1 + x_2 = 6$  and the solution occurs at the point these two lines intersect.



**Figure 4.5: A Linear Programming Problem with Unbounded Feasible Region and Finite Solution:**  
In this problem, the level curves of  $z(x_1, x_2)$  increase in a more “southerly” direction than in Example 4.4—that is, *away from the direction in which the feasible region increases without bound*. The point in the feasible region with largest  $z(x_1, x_2)$  value is  $(7/3, 4/3)$ . Note again, this is a vertex.



Based on these two examples, we can modify our algorithm for graphically solving a two variable linear programming problems to deal with the case when the feasible region is unbounded.

---

**Algorithm 4** Algorithm for Solving a Linear Programming Problem Graphically–Bounded and Unbounded Case

---

**Algorithm for Solving a Two Variable Linear Programming Problem Graphically**

1. Plot the feasible region defined by the constraints.
  2. If the feasible region is empty, then no solution exists.
  3. If the feasible region is unbounded goto Line 8. Otherwise, Goto Line 4.
  4. Plot the level sets of the objective function.
  5. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
  6. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.
  7. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**
  8. (The feasible region is unbounded): Plot the level sets of the objective function.
  9. If the level sets intersect the feasible region at larger and larger (smaller and smaller for a minimization problem), then the problem is unbounded and the solution is  $+\infty$  ( $-\infty$  for minimization problems).
  10. Otherwise, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
  11. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**
- 

**Exercise 4.9**

Does the following problem have a bounded solution? Why?

$$\left\{ \begin{array}{l} \min z(x_1, x_2) = 2x_1 - x_2 \\ \text{s.t. } x_1 - x_2 \leq 1 \\ \quad 2x_1 + x_2 \geq 6 \\ \quad x_1, x_2 \geq 0 \end{array} \right. \quad (4.3)$$

[Hint: Use Figure 4.5 and Algorithm 4.]

**Exercise 4.10**

Modify the objective function in Example 4.4 or Example 4.4 to produce a problem with an infinite number of solutions.

**Exercise 4.11**

Modify the objective function in Exercise 4.4 to produce a **minimization** problem that has a finite solution. Draw the feasible region and level curves of the objective to “prove” your example works. [Hint: Think about what direction of increase is required for the level sets of  $z(x_1, x_2)$  (or find a trick using Exercise ??).]

## 4.5 Formal Mathematical Statements

---

### Vectors and Linear and Convex Combinations

**Vectors:** Vector  $\mathbf{n}$  has  $n$ -elements and represents a point (or an arrow from the origin to the point, denoting a direction) in  $\mathcal{R}^n$  space (Euclidean or real space). Vectors can be expressed as either row or column vectors.

**Vector Addition:** Two vectors of the same size can be added, componentwise, e.g., for vectors  $\mathbf{a} = (2, 3)$  and  $\mathbf{b} = (3, 2)$ ,  $\mathbf{a} + \mathbf{b} = (2 + 3, 3 + 2) = (5, 5)$ .

**Scalar Multiplication:** A vector can be multiplied by a scalar  $k$  (constant) component-wise. If  $k > 0$  then this does not change the direction represented by the vector, it just scales the vector.

**Inner or Dot Product:** Two vectors of the same size can be multiplied to produce a real number. For example,  $\mathbf{ab} = 2 * 3 + 3 * 2 = 10$ .

**Linear Combination:** The vector  $\mathbf{b}$  is a **linear combination** of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  if  $\mathbf{b} = \sum_{i=1}^k \lambda_i \mathbf{a}_i$  for  $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathcal{R}$ . If  $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathcal{R}_{\geq 0}$  then  $\mathbf{b}$  is a *non-negative linear combination* of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$ .

**Convex Combination:** The vector  $\mathbf{b}$  is a **convex combination** of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  if  $\mathbf{b} = \sum_{i=1}^k \lambda_i \mathbf{a}_i$ , for  $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathcal{R}_{\geq 0}$  and  $\sum_{i=1}^k \lambda_i = 1$ . For example, any convex combination of two points will lie on the line segment between the points.

**Linear Independence:** Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  are *linearly independent* if the following linear combination  $\sum_{i=1}^k \lambda_i \mathbf{a}_i = 0$  implies that  $\lambda_i = 0$ ,  $i = 1, 2, \dots, k$ . In  $\mathcal{R}^2$  two vectors are only linearly dependent if they lie on the same line. Can you have three linearly independent vectors in  $\mathcal{R}^2$ ?

**Spanning Set:** Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  span  $\mathcal{R}^m$  if any vector in  $\mathcal{R}^m$  can be represented as a linear combination of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$ , i.e.,  $\sum_{i=1}^m \lambda_i \mathbf{a}_i$  can represent any vector in  $\mathcal{R}^m$ .

**Basis:** Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  form a basis of  $\mathcal{R}^m$  if they span  $\mathcal{R}^m$  and any smaller subset of these vectors does not span  $\mathcal{R}^m$ . Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  can only form a basis of  $\mathcal{R}^m$  if  $k = m$  and they are linearly independent.

## Convex and Polyhedral Sets

**Convex Set:** Set  $\mathcal{S}$  in  $\mathbb{R}^n$  is a *convex set* if a line segment joining any pair of points  $\mathbf{a}_1$  and  $\mathbf{a}_2$  in  $\mathcal{S}$  is completely contained in  $\mathcal{S}$ , that is,  $\lambda\mathbf{a}_1 + (1 - \lambda)\mathbf{a}_2 \in \mathcal{S}, \forall \lambda \in [0, 1]$ .

**Hyperplanes and Half-Spaces:** A hyperplane in  $\mathbb{R}^n$  divides  $\mathbb{R}^n$  into 2 half-spaces (like a line does in  $\mathbb{R}^2$ ). A hyperplane is the set  $\{\mathbf{x} : \mathbf{p}\mathbf{x} = k\}$ , where  $\mathbf{p}$  is the gradient to the hyperplane (i.e., the coefficients of our linear expression). The corresponding half-spaces is the set of points  $\{\mathbf{x} : \mathbf{p}\mathbf{x} \geq k\}$  and  $\{\mathbf{x} : \mathbf{p}\mathbf{x} \leq k\}$ .

**Polyhedral Set:** A *polyhedral set* (or polyhedron) is the set of points in the intersection of a finite set of half-spaces. Set  $\mathcal{S} = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ , where  $\mathbf{A}$  is an  $m \times n$  matrix,  $\mathbf{x}$  is an  $n$ -vector, and  $\mathbf{b}$  is an  $m$ -vector, is a *polyhedral set* defined by  $m+n$  hyperplanes (i.e., the intersection of  $m+n$  half-spaces).

- Polyhedral sets are convex.
- A polytope is a bounded polyhedral set.
- A polyhedral cone is a polyhedral set where the hyperplanes (that define the half-spaces) pass through the origin, thus  $\mathcal{C} = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq 0\}$  is a polyhedral cone.

**Edges and Faces:** An *edge* of a polyhedral set  $\mathcal{S}$  is defined by  $n-1$  hyperplanes, and a *face* of  $\mathcal{S}$  by one of more defining hyperplanes of  $\mathcal{S}$ , thus an extreme point and an edge are faces (an extreme point is a zero-dimensional face and an edge a one-dimensional face). In  $\mathbb{R}^2$  faces are only edges and extreme points, but in  $\mathbb{R}^3$  there is a third type of face, and so on...

**Extreme Points:**  $\mathbf{x} \in \mathcal{S}$  is an extreme point of  $\mathcal{S}$  if:

**Definition 1:**  $\mathbf{x}$  is not a convex combination of two other points in  $\mathcal{S}$ , that is, all line segments that are completely in  $\mathcal{S}$  that contain  $\mathbf{x}$  must have  $\mathbf{x}$  as an endpoint.

**Definition 2:**  $\mathbf{x}$  lies on  $n$  linearly independent defining hyperplanes of  $\mathcal{S}$ .

If more than  $n$  hyperplanes pass through an extreme points then it is a degenerate extreme point, and the polyhedral set is considered degenerate. This just adds a bit of complexity to the algorithms we will study, but it is quite common.

## Unbounded Sets:

**Rays:** A ray in  $\mathbb{R}^n$  is the set of points  $\{\mathbf{x} : \mathbf{x}_0 + \lambda\mathbf{d}, \lambda \geq 0\}$ , where  $\mathbf{x}_0$  is the vertex and  $\mathbf{d}$  is the direction of the ray.

**Convex Cone:** A *Convex Cone* is a convex set that consists of rays emanating from the origin. A convex cone is completely specified by its extreme directions. If  $\mathcal{C}$  is convex cone, then for any  $\mathbf{x} \in \mathcal{C}$  we have  $\lambda\mathbf{x} \in \mathcal{C}, \lambda \geq 0$ .

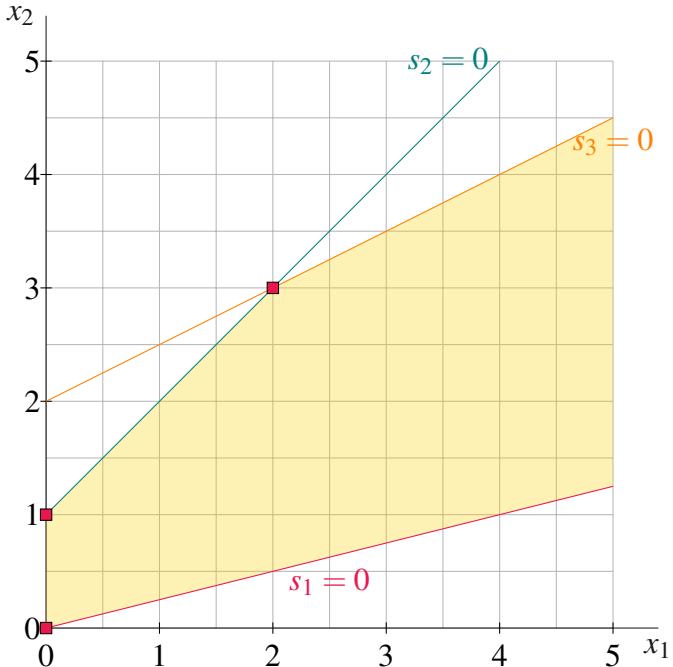
**Unbounded Polyhedral Sets:** If  $\mathcal{S}$  is unbounded, it will have *directions*.  $\mathbf{d}$  is a direction of  $\mathcal{S}$  only if  $\mathbf{Ax} + \lambda\mathbf{d} \leq \mathbf{b}, \mathbf{x} + \lambda\mathbf{d} \geq 0$  for all  $\lambda \geq 0$  and all  $\mathbf{x} \in \mathcal{S}$ . In other words, consider the ray  $\{\mathbf{x} : \mathbf{x}_0 + \lambda\mathbf{d}, \lambda \geq 0\}$

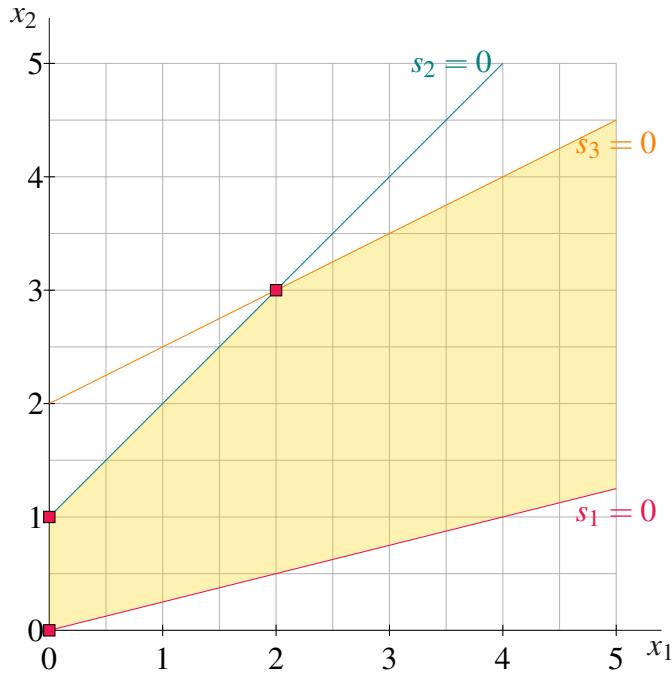
in  $\mathbb{R}^n$ , where  $\mathbf{x}_0$  is the vertex and  $\mathbf{d}$  is the direction of the ray.  $\mathbf{d} \neq 0$  is a **direction** of set  $\mathcal{S}$  if for each  $\mathbf{x}_0$  in  $\mathcal{S}$  the ray  $\{\mathbf{x}_0 + \lambda \mathbf{d}, \lambda \geq 0\}$  also belongs to  $\mathcal{S}$ .

**Extreme Directions:** An *extreme direction* of  $\mathcal{S}$  is a direction that *cannot* be represented as positive linear combination of other directions of  $\mathcal{S}$ . A non-negative linear combination of extreme directions can be used to represent all other directions of  $\mathcal{S}$ . A polyhedral cone is completely specified by its extreme directions.

Let's define a procedure for finding the extreme directions, using the following LP's feasible region. Graphically, we can see that the extreme directions should follow the the  $s_1 = 0$  (red) line and the  $s_3 = 0$  (orange) line.

$$\begin{aligned} \max \quad & z = -5x_1 - x_2 \\ \text{s.t.} \quad & x_1 - 4x_2 + s_1 = 0 \\ & -x_1 + x_2 + s_2 = 1 \\ & -x_1 + 2x_2 + s_3 = 4 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0. \end{aligned}$$



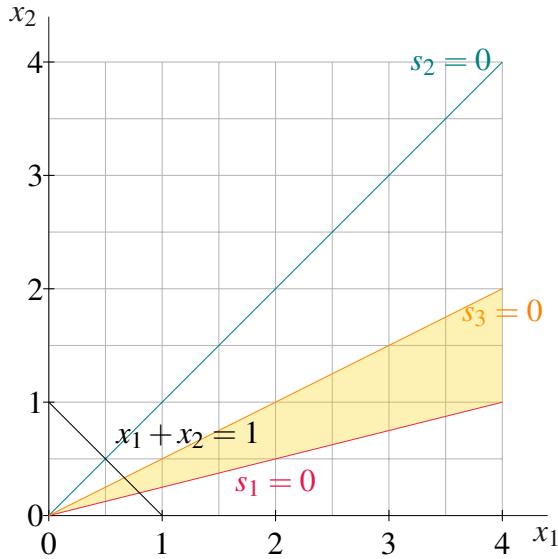


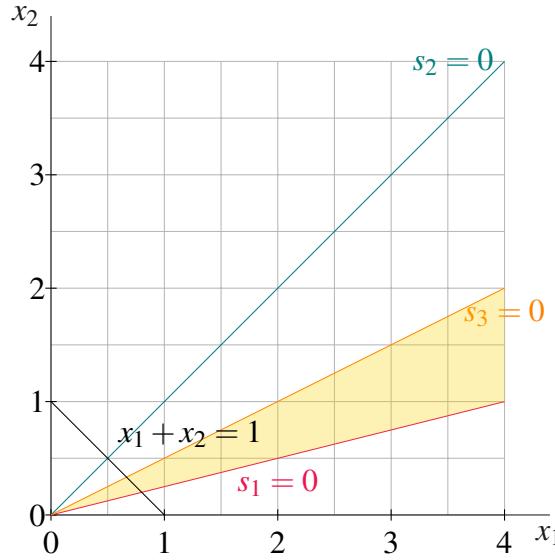
E.g., consider the  $s_3 = 0$  (orange) line, to find the extreme direction start at extreme point (2,3) and find another feasible point on the orange line, say (4,4) and subtract (2,3) from (4,4), which yields (2,1).

This is related to the slope in two-dimensions, as discussed in class, the rise is 1 and the run is 2. So this direction has a slope of 1/2, but this does not carry over easily to higher dimensions where directions cannot be defined by a single number.

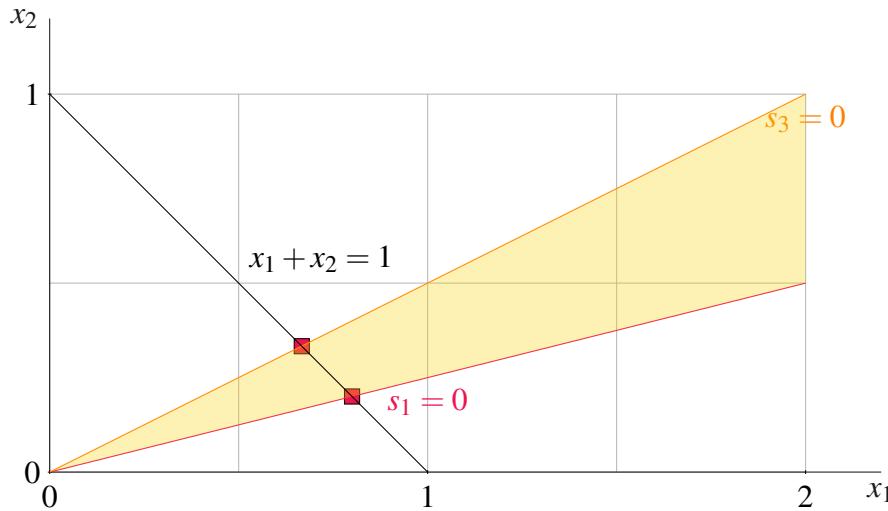
To find the extreme directions we can change the right-hand-side to  $\mathbf{b} = 0$ , which forms a polyhedral cone (in yellow), and then add the constraint  $x_1 + x_2 = 1$ . The intersection of the cone and  $x_1 + x_2 = 1$  form a line segment.

$$\begin{aligned} \max \quad & z = -5x_1 - x_2 \\ \text{s.t.} \quad & x_1 - 4x_2 + s_1 = 0 \\ & -x_1 + x_2 + s_2 = 0 \\ & -x_1 + 2x_2 + s_3 = 0 \\ & x_1 + x_2 = 1 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0. \end{aligned}$$





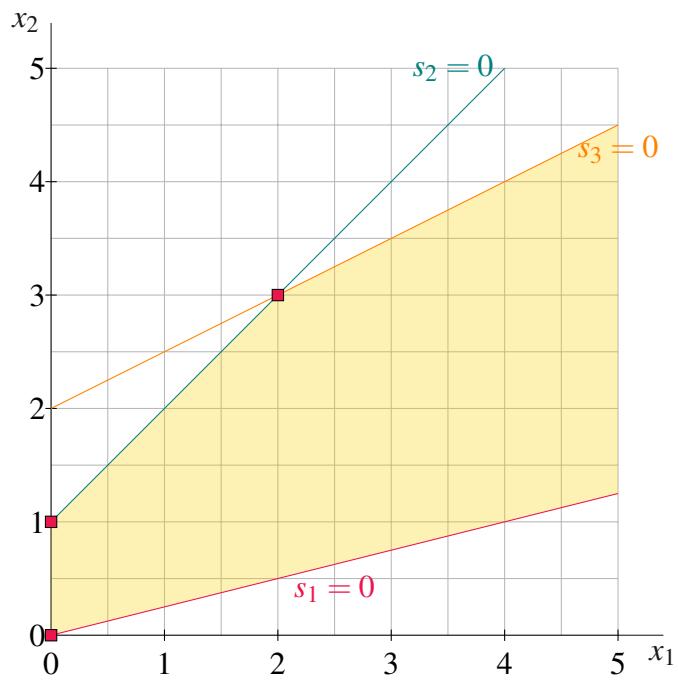
Magnifying for clarity, and removing the  $s_2 = 0$  (teal) line, as it is redundant, and marking the extreme points of the new feasible region,  $(4/5, 1/5)$  and  $(2/3, 1/3)$ , with red boxes, we have:



The extreme directions are thus  $(4/5, 1/5)$  and  $(2/3, 1/3)$ .

**Representation Theorem:** Let  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$  be the set of extreme points of  $\mathcal{S}$ , and if  $\mathcal{S}$  is unbounded,  $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_l$  be the set of extreme directions. Then any  $\mathbf{x} \in \mathcal{S}$  is equal to a convex combination of the extreme points and a non-negative linear combination of the extreme directions:  $\mathbf{x} = \sum_{j=1}^k \lambda_j \mathbf{x}_j + \sum_{j=1}^l \mu_j \mathbf{d}_j$ , where  $\sum_{j=1}^k \lambda_j = 1$ ,  $\lambda_j \geq 0$ ,  $\forall j = 1, 2, \dots, k$ , and  $\mu_j \geq 0$ ,  $\forall j = 1, 2, \dots, l$ .

$$\begin{aligned}
 \max \quad & z = -5x_1 - x_2 \\
 \text{s.t.} \quad & x_1 - 4x_2 + s_1 = 0 \\
 & -x_1 + x_2 + s_2 = 1 \\
 & -x_1 + 2x_2 + s_3 = 4 \\
 & x_1, x_2, s_1, s_2, s_3 \geq 0.
 \end{aligned}$$



Represent point  $(1/2, 1)$  as a convex combination of the extreme points of the above LP. Find  $\lambda$ s to solve the following system of equations:

$$\lambda_1 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \lambda_3 \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$$

# 5. Software - Excel

---

## Resources

- *Excel Solver - Introduction on Youtube*
- *Some notes from MIT*

### 5.0.1. Excel Solver

---

### 5.0.2. Videos

---

Solving a linear program Optimal product mix Set Cover

Introduction to Designing Optimization Models Using Excel Solver

Traveling Salesman Problem

Also Travelin Salesman Problem

Multiple Traveling Salesman Problem

Shortest Path

### 5.0.3. Links

---

Loan Example

Several Examples including TSP



# 6. Software - Python

---

## Outcomes

- *Install and get python up and running in some form*
- *Introduce basic python skills that will be helpful*

## Resources

- *A Byte of Python*
- *Github - Byte of Python (CC-BY-SA)*

## 6.1 Installing and Managing Python

---

Installing and Managing Python

**Lab Objective:** *One of the great advantages of Python is its lack of overhead: it is relatively easy to download, install, start up, and execute. This appendix introduces tools for installing and updating specific packages and gives an overview of possible environments for working efficiently in Python.*

## Installing Python via Anaconda

---

A *Python distribution* is a single download containing everything needed to install and run Python, together with some common packages. For this curriculum, we **strongly** recommend using the *Anaconda* distribution to install Python. Anaconda includes IPython, a few other tools for developing in Python, and a large selection of packages that are common in applied mathematics, numerical computing, and data science. Anaconda is free and available for Windows, Mac, and Linux.

Follow these steps to install Anaconda.

1. Go to <https://www.anaconda.com/download/>.
2. Download the **Python 3.6** graphical installer specific to your machine.
3. Open the downloaded file and proceed with the default configurations.

For help with installation, see <https://docs.anaconda.com/anaconda/install/>. This page contains links to detailed step-by-step installation instructions for each operating system, as well as information for updating and uninstalling Anaconda.

**ACHTUNG!**

This curriculum uses Python 3.6, **not** Python 2.7. With the wrong version of Python, some example code within the labs may not execute as intended or result in an error.

## Managing Packages

---

A *Python package manager* is a tool for installing or updating Python packages, which involves downloading the right source code files, placing those files in the correct location on the machine, and linking the files to the Python interpreter. **Never** try to install a Python package without using a package manager (see <https://xkcd.com/349/>).

### Conda

---

Many packages are not included in the default Anaconda download but can be installed via Anaconda's package manager, conda. See <https://docs.anaconda.com/anaconda/packages/pkg-docs> for the complete list of available packages. When you need to update or install a package, **always** try using conda first.

Command	Description
conda install <package-name>	Install the specified package.
conda update <package-name>	Update the specified package.
conda update conda	Update conda itself.
conda update anaconda	Update <b>all</b> packages included in Anaconda.
conda --help	Display the documentation for conda.

For example, the following terminal commands attempt to install and update matplotlib.

```
$ conda update conda          # Make sure that conda is up to date.
$ conda install matplotlib    # Attempt to install matplotlib.
$ conda update matplotlib     # Attempt to update matplotlib.
```

See <https://conda.io/docs/user-guide/tasks/manage-pkgs.html> for more examples.

**NOTE**

The best way to ensure a package has been installed correctly is to try importing it in IPython.

```
# Start IPython from the command line.
```

```
$ ipython
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

# Try to import matplotlib.
In [1]: from matplotlib import pyplot as plt      # Success!
```

## ACHTUNG!

Be careful not to attempt to update a Python package while it is in use. It is safest to update packages while the Python interpreter is not running.

## Pip

---

The most generic Python package manager is called pip. While it has a larger package list, conda is the cleaner and safer option. Only use pip to manage packages that are not available through conda.

Command	Description
pip install package-name	Install the specified package.
pip install --upgrade package-name	Update the specified package.
pip freeze	Display the version number on all installed packages.
pip --help	Display the documentation for pip.

See [https://pip.pypa.io/en/stable/user\\_guide/](https://pip.pypa.io/en/stable/user_guide/) for more complete documentation.

## Workflows

---

There are several different ways to write and execute programs in Python. Try a variety of workflows to find what works best for you.

## Text Editor + Terminal

---

The most basic way of developing in Python is to write code in a text editor, then run it using either the Python or IPython interpreter in the terminal.

There are many different text editors available for code development. Many text editors are designed specifically for computer programming which contain features such as syntax highlighting and error detection, and are highly customizable. Try installing and using some of the popular text editors listed below.

- Atom: <https://atom.io/>
- Sublime Text: <https://www.sublimetext.com/>
- Notepad++ (Windows): <https://notepad-plus-plus.org/>
- Geany: <https://www.geany.org/>
- Vim: <https://www.vim.org/>
- Emacs: <https://www.gnu.org/software/emacs/>

Once Python code has been written in a text editor and saved to a file, that file can be executed in the terminal or command line.

```
$ ls                               # List the files in the current directory.
hello_world.py
$ cat hello_world.py               # Print the contents of the file to the terminal.
print("hello, world!")
$ python hello_world.py           # Execute the file.
hello, world!

# Alternatively, start IPython and run the file.
$ ipython
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %run hello_world.py
hello, world!
```

IPython is an enhanced version of Python that is more user-friendly and interactive. It has many features that cater to productivity such as tab completion and object introspection.

### NOTE

While Mac and Linux computers come with a built-in bash terminal, Windows computers do not. Windows does come with *Powershell*, a terminal-like application, but some commands in Powershell are different than their bash analogs, and some bash commands are missing from Powershell altogether. There are two good alternatives to the bash terminal for Windows:

- Windows subsystem for linux: [docs.microsoft.com/en-us/windows/wsl/](https://docs.microsoft.com/en-us/windows/wsl/).

- Git bash: <https://gitforwindows.org/>.

## Jupyter Notebook

---

The Jupyter Notebook (previously known as IPython Notebook) is a browser-based interface for Python that comes included as part of the Anaconda Python Distribution. It has an interface similar to the IPython interpreter, except that input is stored in cells and can be modified and re-evaluated as desired. See <https://github.com/jupyter/jupyter/wiki/> for some examples.

To begin using Jupyter Notebook, run the command `jupyter notebook` in the terminal. This will open your file system in a web browser in the Jupyter framework. To create a Jupyter Notebook, click the **New** drop down menu and choose **Python 3** under the **Notebooks** heading. A new tab will open with a new Jupyter Notebook.

Jupyter Notebooks differ from other forms of Python development in that notebook files contain not only the raw Python code, but also formatting information. As such, Jupyter Notebook files cannot be run in any other development environment. They also have the file extension `.ipynb` rather than the standard Python extension `.py`.

Jupyter Notebooks also support Markdown—a simple text formatting language—and  $\text{\LaTeX}$ , and can embedded images, sound clips, videos, and more. This makes Jupyter Notebook the ideal platform for presenting code.

## Integrated Development Environments

---

An *integrated development environment* (IDEs) is a program that provides a comprehensive environment with the tools necessary for development, all combined into a single application. Most IDEs have many tightly integrated tools that are easily accessible, but come with more overhead than a plain text editor. Consider trying out each of the following IDEs.

- JupyterLab: <http://jupyterlab.readthedocs.io/en/stable/>
- PyCharm: <https://www.jetbrains.com/pycharm/>
- Spyder: <http://code.google.com/p/spyderlib/>
- Eclipse with PyDev: <http://www.eclipse.org/>, <https://www.pydev.org/>

See <https://realpython.com/python-ides-code-editors-guide/> for a good overview of these (and other) workflow tools.

## 6.2 NumPy Visual Guide

---

**NumPy Visual Guide Lab Objective:** *NumPy operations can be difficult to visualize, but the concepts are straightforward. This appendix provides visual demonstrations of how NumPy arrays are used with slicing syntax, stacking, broadcasting, and axis-specific operations. Though these visualizations are for 1- or 2-dimensional arrays, the concepts can be extended to n-dimensional arrays.*

## Data Access

---

The entries of a 2-D array are the rows of the matrix (as 1-D arrays). To access a single entry, enter the row index, a comma, and the column index. Remember that indexing begins with 0.

$$A[0] = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \quad A[2,1] = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$

## Slicing

---

A lone colon extracts an entire row or column from a 2-D array. The syntax  $[a:b]$  can be read as “the  $a$ th entry up to (but not including) the  $b$ th entry.” Similarly,  $[a:]$  means “the  $a$ th entry to the end” and  $[:b]$  means “everything up to (but not including) the  $b$ th entry.”

$$A[1] = A[1,:] = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \quad A[:,2] = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$
  

$$A[1:,:2] = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \quad A[1:-1,1:-1] = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$

## Stacking

---

`np.hstack()` stacks sequence of arrays horizontally and `np.vstack()` stacks a sequence of arrays vertically.

$$A = \begin{bmatrix} \textcolor{blue}{\times} & \textcolor{blue}{\times} & \textcolor{blue}{\times} \\ \textcolor{blue}{\times} & \textcolor{blue}{\times} & \textcolor{blue}{\times} \\ \textcolor{blue}{\times} & \textcolor{blue}{\times} & \textcolor{blue}{\times} \end{bmatrix} \quad B = \begin{bmatrix} \textcolor{red}{*} & \textcolor{red}{*} & \textcolor{red}{*} \\ \textcolor{red}{*} & \textcolor{red}{*} & \textcolor{red}{*} \\ \textcolor{red}{*} & \textcolor{red}{*} & \textcolor{red}{*} \end{bmatrix}$$

$$\text{np.hstack}((A, B, A)) = \begin{bmatrix} \times & \times & \times & * & * & * & \times & \times & \times \\ \times & \times & \times & * & * & * & \times & \times & \times \\ \times & \times & \times & * & * & * & \times & \times & \times \end{bmatrix}$$

$$\text{np.vstack}((A, B, A)) = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ * & * & * \\ * & * & * \\ * & * & * \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}$$

Because 1-D arrays are flat, `np.hstack()` concatenates 1-D arrays and `np.vstack()` stacks them vertically. To make several 1-D arrays into the columns of a 2-D array, use `np.column_stack()`.

$$x = [ \times \times \times \times ]$$

$$y = [ * * * * ]$$

$$\text{np.hstack}((x, y, x)) = [ \times \times \times \times * * * * \times \times \times \times ]$$

$$\text{np.vstack}((x, y, x)) = \begin{bmatrix} \times & \times & \times & \times \\ * & * & * & * \\ \times & \times & \times & \times \end{bmatrix}$$

$$\text{np.column_stack}((x, y, x)) = \begin{bmatrix} \times & * & \times \\ \times & * & \times \\ \times & * & \times \\ \times & * & \times \end{bmatrix}$$

The functions `np.concatenate()` and `np.stack()` are more general versions of `np.hstack()` and `np.vstack()`, and `np.row_stack()` is an alias for `np.vstack()`.

## Broadcasting

---

NumPy automatically aligns arrays for component-wise operations whenever possible. See <http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html> for more in-depth examples and broadcasting rules.

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

$$x = [ 10 \ 20 \ 30 ]$$

$$A + x = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \\ + \\ [10 & 20 & 30] \end{bmatrix} = \begin{bmatrix} 11 & 22 & 33 \\ 11 & 22 & 33 \\ 11 & 22 & 33 \end{bmatrix}$$

$$A + x.reshape((1, -1)) = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} + \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$$

## Operations along an Axis

---

Most array methods have an `axis` argument that allows an operation to be done along a given axis. To compute the sum of each column, use `axis=0`; to compute the sum of each row, use `axis=1`.

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

$$A.sum(axis=0) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix} = [ 4 \ 8 \ 12 \ 16 ]$$

$$A.sum(axis=1) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix} = [ 10 \ 10 \ 10 \ 10 ]$$

## 6.3 Plot Customization and Matplotlib Syntax Guide

---

### Matplotlib Customization

**Lab Objective:** *The documentation for Matplotlib can be a little difficult to maneuver and basic information is sometimes difficult to find. This appendix condenses and demonstrates some of the more applicable and useful information on plot customizations. For an introduction to Matplotlib, see lab ??.*

## Colors

---

By default, every plot is assigned a different color specified by the “color cycle”. It can be overwritten by specifying what color is desired in a few different ways.

- Matplotlib recognizes some basic built-in colors.

Code	Color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

The following displays how these colors can be implemented. The result is displayed in Figure 6.1.

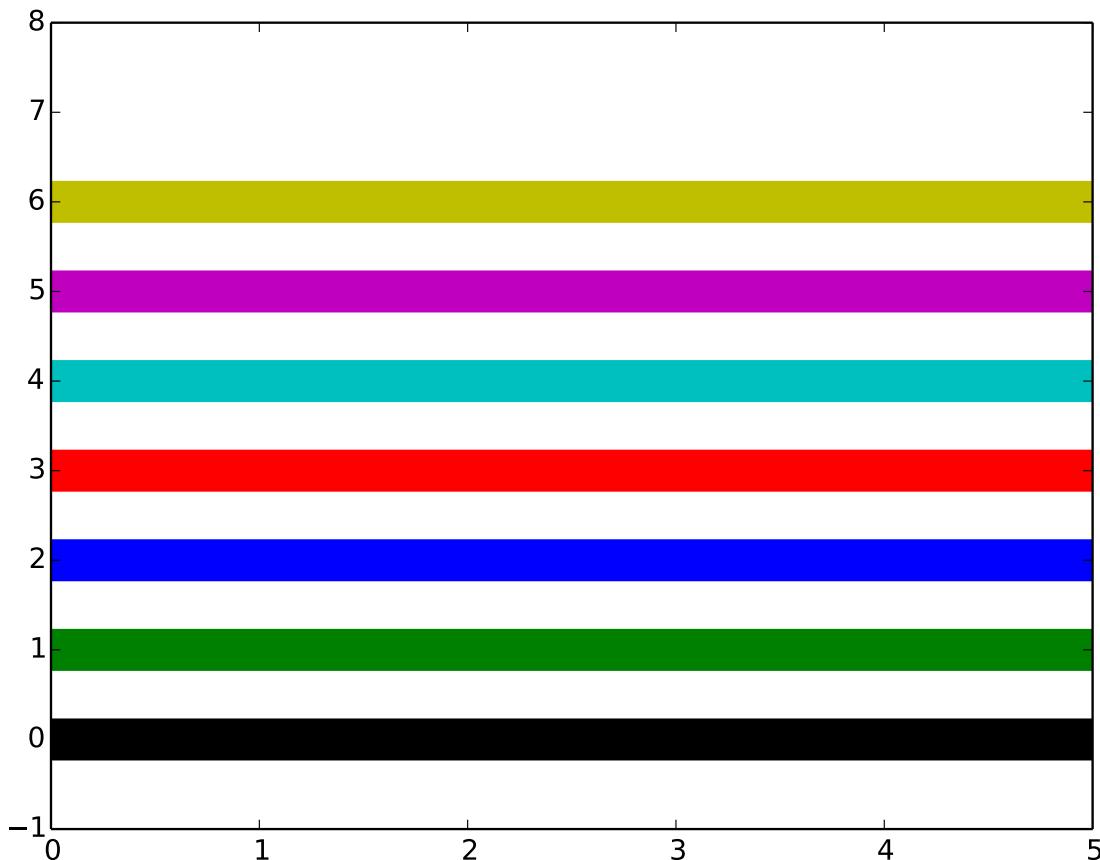
```

1 import numpy as np
2 from matplotlib import pyplot as plt

4 colors = np.array(["k", "g", "b", "r", "c", "m", "y", "w"])
x = np.linspace(0, 5, 1000)
6 y = np.ones(1000)

8 for i in xrange(8):
    plt.plot(x, i*y, colors[i], linewidth=18)
10
11 plt.ylim([-1, 8])
12 plt.savefig("colors.pdf", format='pdf')
plt.clf()

```

**colors.py**

**Figure 6.1: A display of all the built-in colors.**

There are many other ways to specific colors. A popular method to access colors that are not built-in is to use a RGB tuple. Colors can also be specified using an html hex string or its associated html color name like "DarkOliveGreen", "FireBrick", "LemonChiffon", "MidnightBlue", "PapayaWhip", or "SeaGreen".

## Window Limits

You may have noticed the use of `plt.ylim([ymin, ymax])` in the previous code. This explicitly sets the boundary of the y-axis. Similarly, `plt.xlim([xmin, xmax])` can be used to set the boundary of the x-axis. Doing both commands simultaneously is possible with the `plt.axis([xmin, xmax, ymin, ymax])`. Remember that these commands must be executed after the plot.

# Lines

---

## Thickness

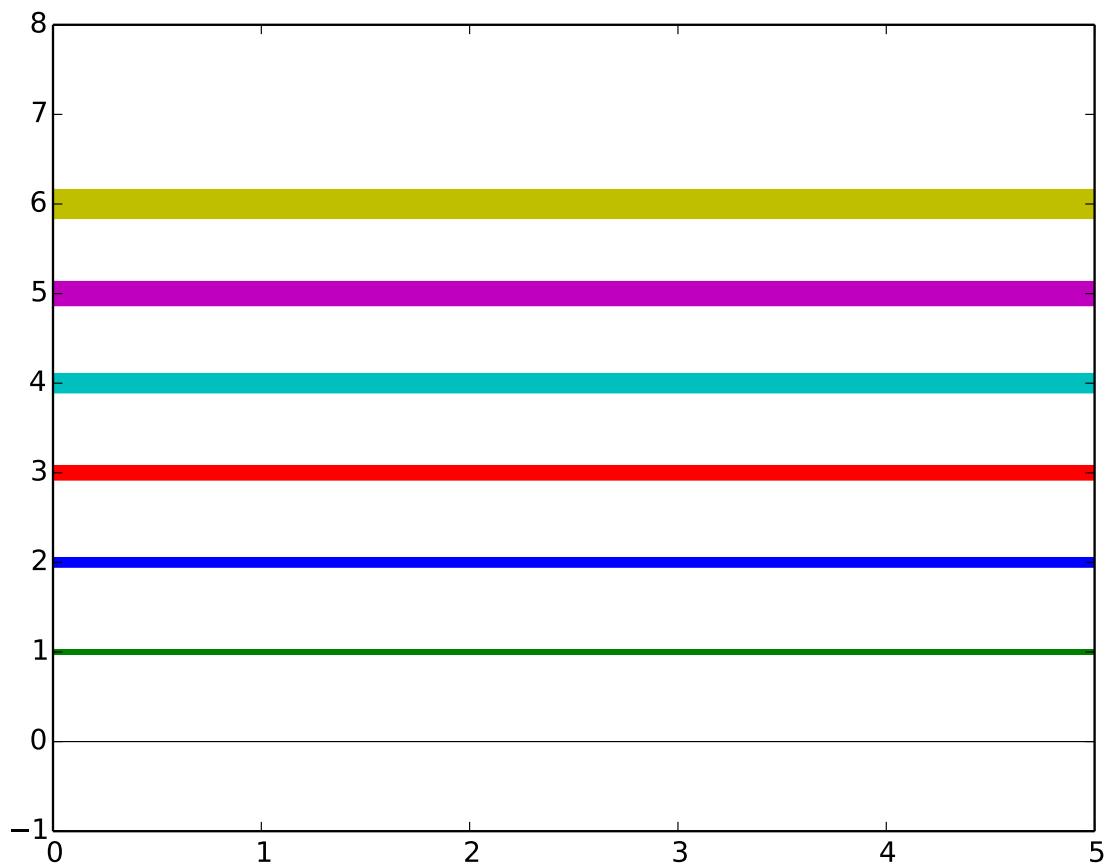
---

You may have noticed that the width of the lines above seemed thin considering we wanted to inspect the line color. `linewidth` is a keyword argument that is defaulted to be `None` but can be given any real number to adjust the line width.

The following displays how `linewidth` is implemented. It is displayed in Figure 6.2.

```
1 lw = np.linspace(.5, 15, 8)
2
3 for i in xrange(8):
4     plt.plot(x, i*y, colors[i], linewidth=lw[i])
5
6 plt.ylim([-1, 8])
7 plt.show()
```

**linewidth.py**



**Figure 6.2:** plot of varying linewidths.

## Style

---

By default, plots are drawn with a solid line. The following are accepted format string characters to indicate line style.

character	description
-	solid line style
--	dashed line style
-.	dash-dot line style
:	dotted line style
.	point marker
,	pixel marker
o	circle marker
v	triangle_down marker
^	triangle_up marker
<	triangle_left marker
>	triangle_right marker
1	tri_down marker
2	tri_up marker
3	tri_left marker
4	tri_right marker
s	square marker
p	pentagon marker
*	star marker
h	hexagon1 marker
H	hexagon2 marker
+	plus marker
x	x marker
D	diamond marker
d	thin_diamond marker
	vline marker
-	hline marker

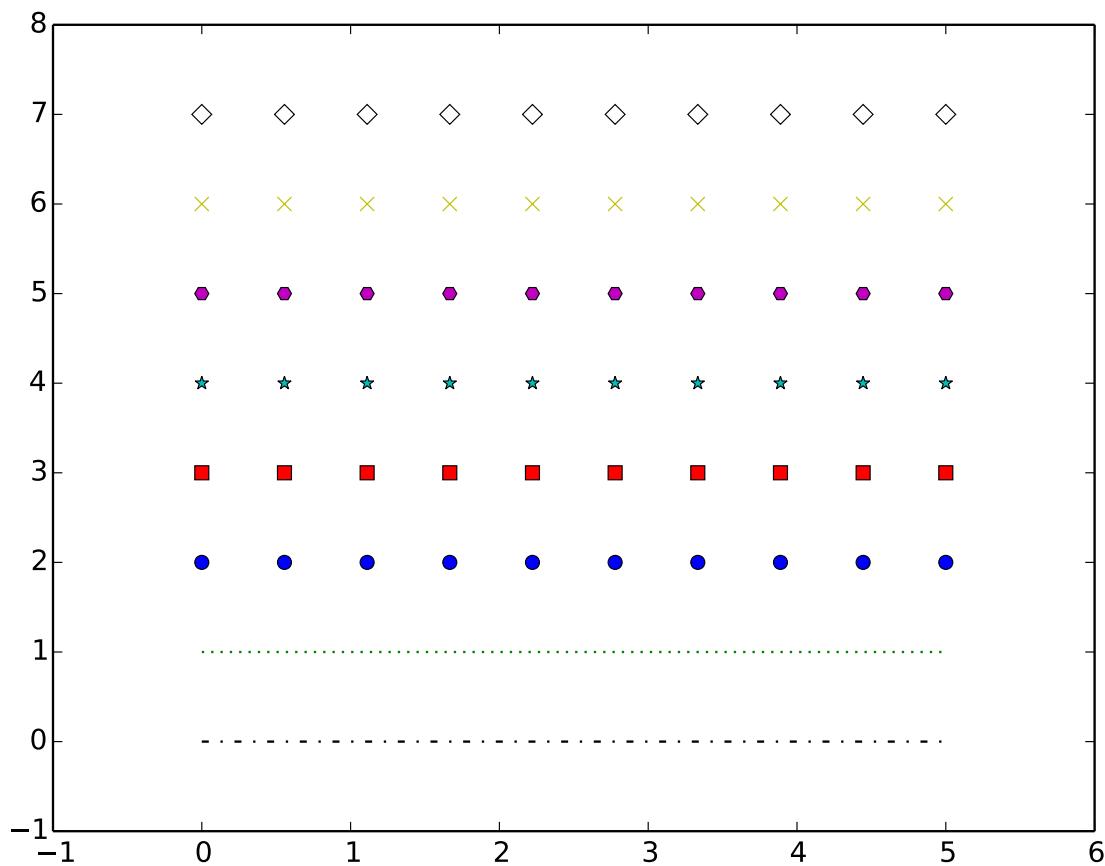
The following displays how `linestyle` can be implemented. It is displayed in Figure 6.3.

```

1 x = np.linspace(0, 5, 10)
2 y = np.ones(10)
ls = np.array(['-.', ':', 'o', 's', '*', 'H', 'x', 'D'])
4
5 for i in xrange(8):
6     plt.plot(x, i*y, colors[i]+ls[i])
8
9 plt.axis([-1, 6, -1, 8])
plt.show()

```

**linestyle.py**



**Figure 6.3:** plot of varying linestyles.

## Text

---

It is also possible to add text to your plots. To label your axes, the `plt.xlabel()` and the `plt.ylabel()` can both be used. The function `plt.title()` will add a title to a plot. If you are working with subplots, this command will add a title to the subplot you are currently modifying. To add a title above the entire figure, use `plt.suptitle()`.

All of the `text()` commands can be customized with `fontsize` and `color` keyword arguments.

We can add these elements to our previous example. It is displayed in Figure 6.4.

```

1 for i in xrange(8):
2     plt.plot(x, i*y, colors[i]+ls[i])
4 plt.title("My Plot of Varying Linestyles", fontsize = 20, color = "gold")
5 plt.xlabel("x-axis", fontsize = 10, color = "darkcyan")
```

```
6 plt.ylabel("y-axis", fontsize = 10, color = "darkcyan")  
8 plt.axis([-1, 6, -1, 8])  
plt.show()
```

text.py

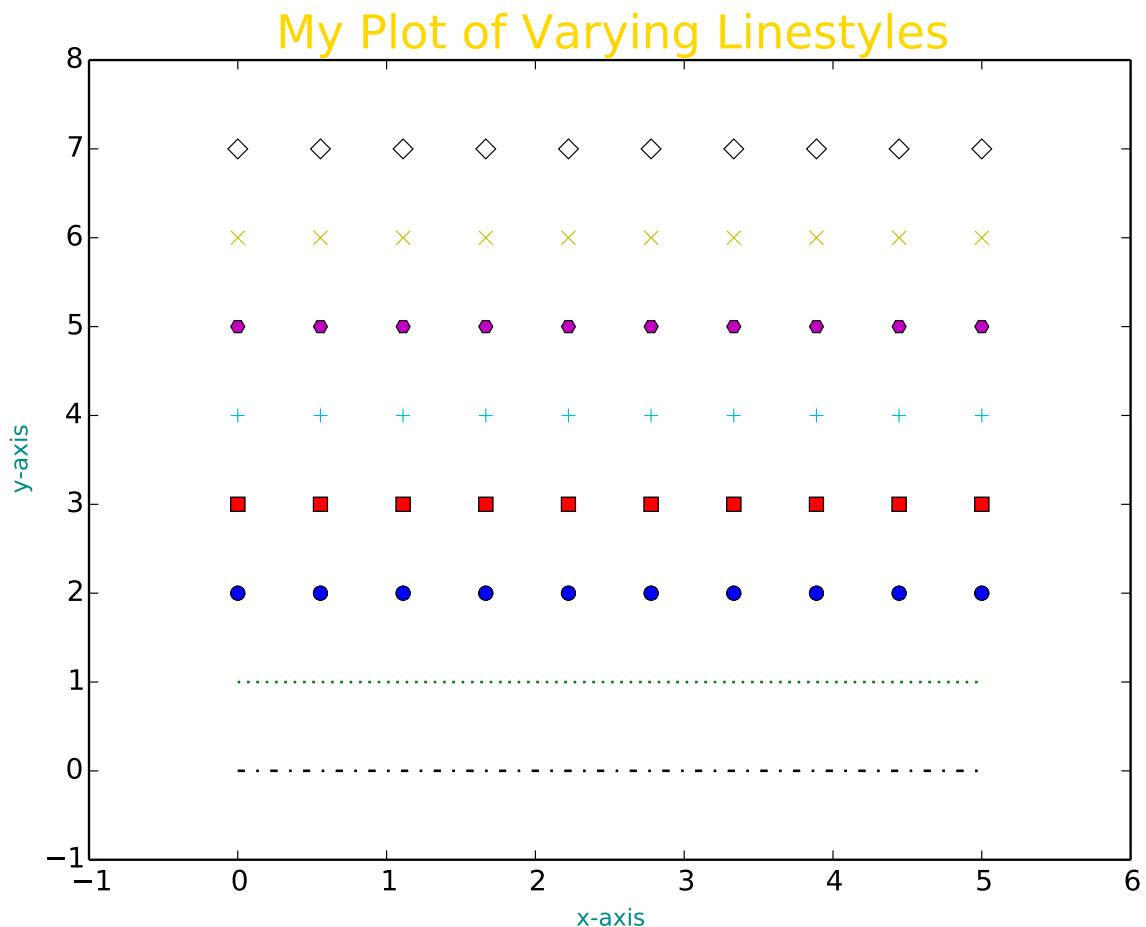


Figure 6.4: plot of varying linestyles using text labels.

See <http://matplotlib.org> for Matplotlib documentation.

## 6.4 Networkx - A Python Graph Algorithms Package

## 6.5 PuLP - An Optimization Modeling Tool for Python

---

### Outcomes

- *Install and import PuLP*
- *Run basic first PuLP model*
- *Run "advanced" PuLP model using the algebraic modeling approach and importing data.*
- *Explore PuLP objects and possibilities*
- *Solve a Multi-Objective problem*

### Resources

- *Documentation*
- *PyPi installation*
- *Examples*
- *Blog with tutorial*

PuLP is an optimization modeling language that is written for Python. It is free and open source. Yay! See Section ?? for a discussion of other options for implementing your optimization problem. PuLP is convenient for its simple syntax and easy installation.

Key benefits of using an algebraic modeling language like PuLP over Excel

- Easily readable models
- Precompute parameters within Python
- Reuse of common optimization models without recreating the equations

We will follow the introduction to pulp Jupyter Notebook Tutorial and the following application with a cleaner implementation.

### 6.5.1. Installation

---

Open a Jupyter notebook. In one of the cells, run the following command, based on which system you are running. It will take a minute to load and download the package.

```
[ ]: ## Install pulp (on windows)
!pip install pulp
```

```
[ ]: # on a mac
pip install pulp
```

```
[ ]: # on the VT ARC servers
import sys
!{sys.executable} -m pip install pulp
```

### Installation (Continued) Now restart the kernel of your notebook (find the tab labeled Kernel in your Jupyter notebook, and in the drop down, select restart).

## 6.5.2. Example Problem

---

### 6.5.2.1. Product Mix Problem

---

$$\text{maximize} \quad Z = 3X_1 + 2X_2 \quad (\text{Objective function}) \quad (1.1)$$

$$\text{subject to} \quad 10X_1 + 5X_2 \leq 300 \quad (\text{Constraint 1}) \quad (1.2)$$

$$4X_1 + 4X_2 \leq 160 \quad (\text{Constraint 2}) \quad (1.3)$$

$$2X_1 + 6X_2 \leq 180 \quad (\text{Constraint 3}) \quad (1.4)$$

$$\text{and} \quad X_1, X_2 \geq 0 \quad (\text{Non-negative}) \quad (1.5)$$

#### OPTIMIZATION WITH PULP

```
[1]: from pulp import *

# Define problem
prob = LpProblem(name='Product_Mix_Problem', sense=LpMaximize)

# Create decision variables and non-negative constraint
x1 = LpVariable(name='X1', lowBound=0, upBound=None, cat='Continuous')
x2 = LpVariable(name='X2', lowBound=0, upBound=None, cat='Continuous')

# Set objective function
prob += 3*x1 + 2*x2

# Set constraints
prob += 10*x1 + 5*x2 <= 300
prob += 4*x1 + 4*x2 <= 160
prob += 2*x1 + 6*x2 <= 180

# Solving problem
prob.solve()
print('Status', LpStatus[prob.status])
```

Status Optimal

```
[2]: print("Status:", LpStatus[prob.status])
print("Objective value: ", prob.objective.value())
```

```
for v in prob.variables():
    print(v.name, ': ', v.value())
```

Status: Optimal  
 Objective value: 100.0  
 X1 : 20.0  
 X2 : 20.0

### 6.5.3. Things we can do

---

[3]: # print the problem  
 prob

[3]: Product\_Mix\_Problem:  
 MAXIMIZE  
 $3*X1 + 2*X2 + 0$   
 SUBJECT TO  
 $_C1: 10 X1 + 5 X2 \leq 300$

$_C2: 4 X1 + 4 X2 \leq 160$

$_C3: 2 X1 + 6 X2 \leq 180$

VARIABLES

X1 Continuous  
 X2 Continuous

[4]: # get the objective function  
 prob.objective.value()

[4]: 100.0

[5]: # get list of the variables  
 prob.variables()

[5]: [X1, X2]

[6]: for v in prob.variables():
 print(f'{v}: {v.varValue}')

X1: 20.0  
 X2: 20.0

### 6.5.3.1. Exploring the variables

---

```
[7]: v = prob.variables()[0]
```

```
[9]: v.name
```

```
[9]: 'X1'
```

```
[10]: v.value()
```

```
[10]: 20.0
```

```
[11]: v.varValue
```

```
[11]: 20.0
```

### 6.5.3.2. Other things you can do

---

```
[12]: # get list of the constraints
prob.constraints
```

```
[12]: OrderedDict([('C1', 10*X1 + 5*X2 + -300 <= 0),
                 ('C2', 4*X1 + 4*X2 + -160 <= 0),
                 ('C3', 2*X1 + 6*X2 + -180 <= 0)])
```

```
[13]: prob.to_dict()
```

```
[13]: {'objective': {'name': 'OBJ',
                   'coefficients': [{'name': 'X1', 'value': 3}, {'name': 'X2', 'value': 2}],
                   'constraints': [{"sense": -1,
                                   'pi': 0.2,
                                   'constant': -300,
                                   'name': None,
                                   'coefficients': [{'name': 'X1', 'value': 10}, {'name': 'X2', 'value': 5}],
                                   'sense': -1,
                                   'pi': 0.25,
                                   'constant': -160,
                                   'name': None,
                                   'coefficients': [{'name': 'X1', 'value': 4}, {'name': 'X2', 'value': 4}],
                                   'sense': -1,
                                   'pi': -0.0,
                                   'constant': -180,
                                   'name': None,
                                   'coefficients': [{'name': 'X1', 'value': 2}, {'name': 'X2', 'value': 6}]},
                   'variables': [{'lowBound': 0,
```

```
'upBound': None,
'cat': 'Continuous',
'varValue': 20.0,
'dj': -0.0,
'name': 'X1'},
{'lowBound': 0,
'upBound': None,
'cat': 'Continuous',
'varValue': 20.0,
'dj': -0.0,
'name': 'X2'}],
'parameters': {'name': 'Product_Mix_Problem',
'sense': -1,
'status': 1,
'sol_status': 1},
'sos1': [],
'sos2': []}
```

```
[15]: # Store problem information in a json
prob.to_json('Product_Mix_Problem.json')
```

## 6.5.4. Common issue

---

If you forget the  $\leq$ ,  $=$ , or  $\geq$  when writing a constraint, you will silently overwrite the objective function instead of adding a constraint!

### 6.5.4.1. Transportation Problem

---

Transport programming is a special form of linear programming, and in general, the objective function is cost minimization. The formula form and applicable variables of the Transport Planning Act are as follows. When supply and demand match, the constraint becomes an equation, but when supply and demand do not match, the constraint becomes an inequality.

Sets: - J = set of demand nodes - I = set of supply nodes

Parameters:

- $D_j$ : Demand at node  $j$
- $S_i$ : Supply from node  $i$
- $c_{ij}$ : cost per unit to send supply  $i$  to demand  $j$

Variables:

- $X_{ij}$ : Transport volume from supply  $i$  to demand  $j$  (units)

- Objective function:

$$\min \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij}$$

- Constraints:

$$\sum_{i=1}^n x_{ij} = S_i$$

$$\sum_{i=1}^m x_{ij} = D_j$$

$$x_{ij} \geq 0 \text{ for } i \in I, j \in J$$

#### 6.5.4.2. Optimization with PuLP

---

Here we do a very basic implementation of the problem

```
[1]: from pulp import *

prob = LpProblem('Transportation_Problem', LpMinimize)

x11 = LpVariable('X11', lowBound=0)
x12 = LpVariable('X12', lowBound=0)
x13 = LpVariable('X13', lowBound=0)
x14 = LpVariable('X14', lowBound=0)
x21 = LpVariable('X21', lowBound=0)
x22 = LpVariable('X22', lowBound=0)
x23 = LpVariable('X23', lowBound=0)
x24 = LpVariable('X24', lowBound=0)
x31 = LpVariable('X31', lowBound=0)
x32 = LpVariable('X32', lowBound=0)
x33 = LpVariable('X33', lowBound=0)
x34 = LpVariable('X34', lowBound=0)

prob += 4*x11 + 5*x12 + 6*x13 + 8*x14 + 4*x21 + 7*x22 + 9*x23 + 2*x24 + 5*x31 + 
       8*x32 + 7*x33 + 6*x34

prob += x11 + x12 + x13 + x14 == 120
prob += x21 + x22 + x23 + x24 == 150
prob += x31 + x32 + x33 + x34 == 200

prob += x11 + x21 + x31 == 100
prob += x12 + x22 + x32 == 60
prob += x13 + x23 + x33 == 130
prob += x14 + x24 + x34 == 180
```

```
# Solving problem
prob.solve();

[2]: print("Status:", LpStatus[prob.status])
print("Objective value: ", prob.objective.value())

for v in prob.variables():
    print(v.name, ': ', v.value())
```

```
Status: Optimal
Objective value:  2130.0
X11 :  60.0
X12 :  60.0
X13 :  0.0
X14 :  0.0
X21 :  0.0
X22 :  0.0
X23 :  0.0
X24 :  150.0
X31 :  40.0
X32 :  0.0
X33 :  130.0
X34 :  30.0
```

### 6.5.4.3. Optimization with PuLP: Round 2!

---

We now use set notation for this implementation

```
[3]: from pulp import *

prob = LpProblem('Transportation_Problem', LpMinimize)

# Sets
n_suppliers = 3
n_buyers = 4

I = range(n_suppliers)
J = range(n_buyers)

routes = [(i, j) for i in I for j in J]

# Parameters
```

```

costs = [
    [4, 5, 6, 8],
    [4, 7, 9, 2],
    [5, 8, 7, 6]
]

supply = [120, 150, 200]
demand = [100, 60, 130, 180]

# Variables
x = LpVariable.dicts('X', routes, lowBound=0)

# Objective
prob += lpSum([x[i, j] * costs[i][j] for i in I for j in J])

# Constraints
## Supply Constraints
for i in range(n_suppliers):
    prob += lpSum([x[i, j] for j in J]) == supply[i], f"Supply{i}"

## Demand Constraints
for j in range(n_buyers):
    prob += lpSum([x[i, j] for i in I]) == demand[j], f"Demand{j}"

# Solving problem
prob.solve();

```

```
[4]: print("Status:", LpStatus[prob.status])
print("Objective value: ", prob.objective.value())

for v in prob.variables():
    print(v.name, ': ', v.value())
```

Status: Optimal  
 Objective value: 2130.0  
 X\_(0,\_0) : 60.0  
 X\_(0,\_1) : 60.0  
 X\_(0,\_2) : 0.0  
 X\_(0,\_3) : 0.0  
 X\_(1,\_0) : 0.0

```
X_(1,_1) : 0.0
X_(1,_2) : 0.0
X_(1,_3) : 150.0
X_(2,_0) : 40.0
X_(2,_1) : 0.0
X_(2,_2) : 130.0
X_(2,_3) : 30.0
```

### 6.5.5. Changing details of the problem

---

```
[5]: original_obj = prob.objective
val = prob.objective.value()
r = 1.2

[6]: prob += original_obj <= r*val, "Objective bound"

[7]: prob
```

**[7]:** Transportation\_Problem:

MINIMIZE

$$4*X_{(0,0)} + 5*X_{(0,1)} + 6*X_{(0,2)} + 8*X_{(0,3)} + 4*X_{(1,0)} + 7*X_{(1,1)} + 9*X_{(1,2)} + 2*X_{(1,3)} + 5*X_{(2,0)} + 8*X_{(2,1)} + 7*X_{(2,2)} + 6*X_{(2,3)} + 0$$

SUBJECT TO

Supply0:  $X_{(0,0)} + X_{(0,1)} + X_{(0,2)} + X_{(0,3)} = 120$

Supply1:  $X_{(1,0)} + X_{(1,1)} + X_{(1,2)} + X_{(1,3)} = 150$

Supply2:  $X_{(2,0)} + X_{(2,1)} + X_{(2,2)} + X_{(2,3)} = 200$

Demand0:  $X_{(0,0)} + X_{(1,0)} + X_{(2,0)} = 100$

Demand1:  $X_{(0,1)} + X_{(1,1)} + X_{(2,1)} = 60$

Demand2:  $X_{(0,2)} + X_{(1,2)} + X_{(2,2)} = 130$

Demand3:  $X_{(0,3)} + X_{(1,3)} + X_{(2,3)} = 180$

Objective\_bound:  $4 X_{(0,0)} + 5 X_{(0,1)} + 6 X_{(0,2)} + 8 X_{(0,3)} + 4 X_{(1,0)} + 7 X_{(1,1)} + 9 X_{(1,2)} + 2 X_{(1,3)} + 5 X_{(2,0)} + 8 X_{(2,1)} + 7 X_{(2,2)} + 6 X_{(2,3)} \leq 2556$

VARIABLES

$X_{(0,0)}$  Continuous

```
X_(0,_1) Continuous
X_(0,_2) Continuous
X_(0,_3) Continuous
X_(1,_0) Continuous
X_(1,_1) Continuous
X_(1,_2) Continuous
X_(1,_3) Continuous
X_(2,_0) Continuous
X_(2,_1) Continuous
X_(2,_2) Continuous
X_(2,_3) Continuous
```

[8]: # Change the objective  
`prob += x[0,0] # minimize x[0,0]`

```
/opt/anaconda3/envs/python377/lib/python3.7/site-packages/pulp/pulp.py:1544:
UserWarning: Overwriting previously set objective.
    warnings.warn("Overwriting previously set objective.")
```

[9]: `prob.solve()`

[9]: 1

[10]: `LpStatus[prob.status]`

[10]: 'Optimal'

```
[11]: print("Status:", LpStatus[prob.status])
print("Objective value: ", prob.objective.value())

for v in prob.variables():
    print(v.name, ': ', v.value())
```

```
Status: Optimal
Objective value:  0.0
X_(0,_0) :  0.0
X_(0,_1) :  60.0
X_(0,_2) :  60.0
X_(0,_3) :  0.0
X_(1,_0) :  100.0
X_(1,_1) :  0.0
X_(1,_2) :  0.0
X_(1,_3) :  50.0
X_(2,_0) :  0.0
X_(2,_1) :  0.0
X_(2,_2) :  70.0
```

```
X_(2,_3) : 130.0
```

```
[12]: original_obj
```

```
[12]: 4*X_(0,_0) + 5*X_(0,_1) + 6*X_(0,_2) + 8*X_(0,_3) + 4*X_(1,_0) + 7*X_(1,_1) +
9*X_(1,_2) + 2*X_(1,_3) + 5*X_(2,_0) + 8*X_(2,_1) + 7*X_(2,_2) + 6*X_(2,_3) + 0
```

```
[13]: original_obj.value()
```

```
[13]: 2430.0
```

## 6.5.6. Changing Constraint Coefficients

---

```
[14]: a = prob.constraints['Supply0']
```

```
[15]: a.changeRHS(500)
```

```
[16]: a
```

```
[16]: 1*X_(0,_0) + 1*X_(0,_1) + 1*X_(0,_2) + 1*X_(0,_3) + -500 = 0
```

```
[17]: prob.constraints['Supply0'].keys()
```

```
[17]: odict_keys([X_(0,_0), X_(0,_1), X_(0,_2), X_(0,_3)])
```

# 6.6 Multi Objective Optimization with PuLP

---

We consider two objectives and compute the pareto efficient frontier. We will implement the  $\varepsilon$ -constraint method. That is, we will add bounds based on an objective function and the optimize the alternate objective function.

## 6.6.0.1. Transportation Problem

---

Sets: -  $J$  = set of demand nodes -  $I$  = set of supply nodes

Parameters: -  $D_j$ : Demand at node  $j$  -  $S_i$ : Supply from node  $i$  -  $c_{ij}$ : cost per unit to send supply  $i$  to demand  $j$

Variables: -  $x_{ij}$ : Transport volume from supply  $i$  to demand  $j$  (units)

- Objective function:

$$\min \left( obj1 = \sum_{i=1}^n \sum_{j=1}^m c_{ij}x_{ij}, \quad obj2 = x_{00} + x_{13} + x_{22} - x_{21} - x_{03} \right)$$

- Constraints:

$$\sum_{i=1}^n x_{ij} = S_i$$

$$\sum_{i=1}^m x_{ij} = D_j$$

- Decision variables:

$$x_{ij} \geq 0 \quad i \in I, j \in J$$

### 6.6.0.2. Initial Optimization with PuLP

---

```
[1]: from pulp import *

prob = LpProblem('Transportation_Problem', LpMinimize)

# Sets
n_suppliers = 3
n_buyers = 4

I = range(n_suppliers)
J = range(n_buyers)

routes = [(i, j) for i in I for j in J]

# Parameters
costs = [
    [4, 5, 6, 8],
    [4, 7, 9, 2],
    [5, 8, 7, 6]
]

supply = [120, 150, 200]
demand = [100, 60, 130, 180]

# Variables
x = LpVariable.dicts('X', routes, lowBound=0)

# Objectives
obj1 = lpSum([x[i, j] * costs[i][j] for i in I for j in J])
```

```

obj2 = x[0,0] + x[1,3] + x[2,2] - x[2,1] - x[0,3]

## start with first objective
prob += obj1

# Constraints

## Supply Constraints
for i in range(n_suppliers):
    prob += lpSum([x[i, j] for j in J]) == supply[i], f"Supply{i}"

## Demand Constraints
for j in range(n_buyers):
    prob += lpSum([x[i, j] for i in I]) == demand[j], f"Demand{j}"

# Solving problem
prob.solve();

```

```

[2]: print("Status:", LpStatus[prob.status])
print("Objective value: ", prob.objective.value())

for v in prob.variables():
    print(v.name, ': ', v.value())

```

Status: Optimal  
 Objective value: 2130.0  
 X\_(0,\_0) : 60.0  
 X\_(0,\_1) : 60.0  
 X\_(0,\_2) : 0.0  
 X\_(0,\_3) : 0.0  
 X\_(1,\_0) : 0.0  
 X\_(1,\_1) : 0.0  
 X\_(1,\_2) : 0.0  
 X\_(1,\_3) : 150.0  
 X\_(2,\_0) : 40.0  
 X\_(2,\_1) : 0.0  
 X\_(2,\_2) : 130.0  
 X\_(2,\_3) : 30.0

```

[3]: # Record objective value
obj1_opt = obj1.value()
obj1_opt

```

[3]: 2130.0

```
[4]: # Add both objective values to a list and also the solution
obj1_vals = [obj1.value()]
obj2_vals = [obj2.value()]
feasible_points = [prob.variables()]

[5]: # Change objective functions and compute optimal objective value for obj2
prob += obj2
prob.solve()

obj2_opt = obj2.value()
obj2_opt
```

/opt/anaconda3/envs/python377/lib/python3.7/site-packages/pulp/pulp.py:1537:  
UserWarning: Overwriting previously set objective.  
  warnings.warn("Overwriting previously set objective.")

[5]: -180.0

```
[6]: # Append these values to the lists
obj1_vals.append(obj1.value())
obj2_vals.append(obj2.value())
feasible_points.append(prob.variables())
```

## 6.6.1. Creating the Pareto Efficient Frontier

---

```
[7]: import numpy as np

# Create an inequality for objective 1
prob += obj1 <= obj1_opt, "Objective_bound1"
obj_constraint = prob.constraints["Objective_bound1"]
```

```
[8]: # Set to optimize objective 2
prob += obj2
```

/opt/anaconda3/envs/python377/lib/python3.7/site-packages/pulp/pulp.py:1537:  
UserWarning: Overwriting previously set objective.  
  warnings.warn("Overwriting previously set objective.")

```
[9]: # Adjusting objective bound of objective 1

r_values = np.arange(1,2000,10)
for r in r_values:
    obj_constraint.changeRHS(r + obj1_opt)
    if 1 == prob.solve():
        obj1_vals.append(obj1.value())
```

```

    obj2_vals.append(obj2.value())
    feasible_points.append(prob.variables())

# Remove objective 1 constraint
obj_constraint.changeRHS(0)
obj_constraint.clear()

```

[10]:

```

# Create constraint for objective 2
prob += obj2 <= obj2_opt, "Objective_bound2"
obj2_constraint = prob.constraints["Objective_bound2"]

# set objective to objective 1
prob += obj1

```

[11]:

```

# Adjusting objective bound of objective 2

r_values = np.arange(1,400,5) # may need to adjust this
for r in r_values:
    obj2_constraint.changeRHS(r*obj2_opt)
    if 1 == prob.solve():
        obj1_vals.append(obj1.value())
        obj2_vals.append(obj2.value())
        feasible_points.append(prob.variables())

# Remove objective 2 constraint
obj2_constraint.changeRHS(0)
obj2_constraint.clear()

```

[12]:

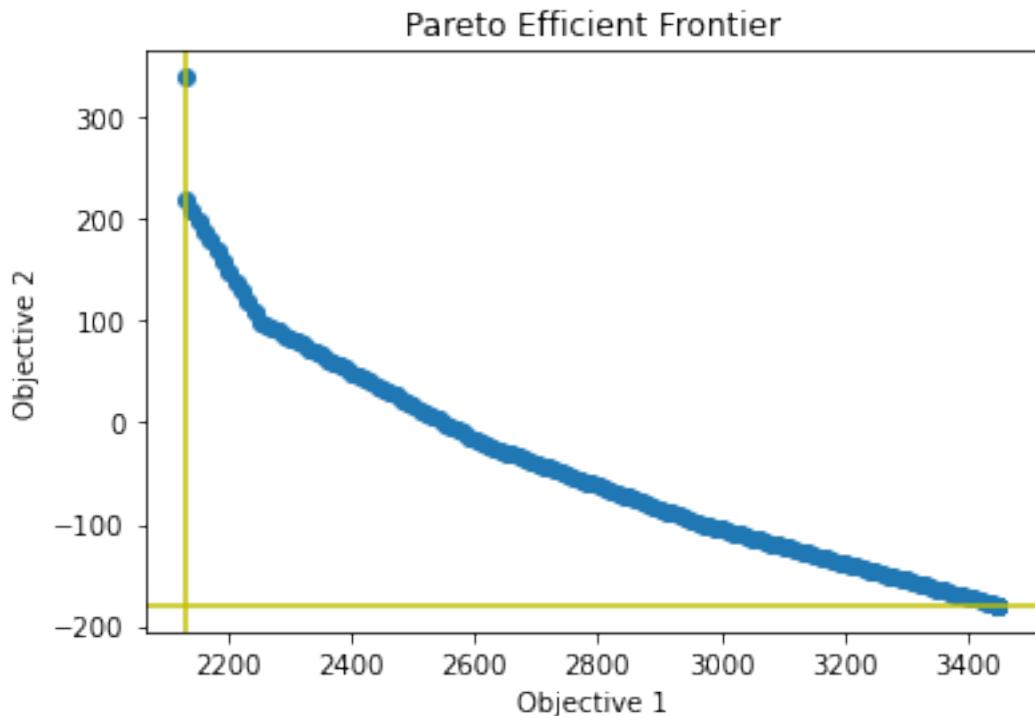
```

import matplotlib.pyplot as plt
plt.scatter(obj1_vals, obj2_vals)
plt.axvline(x=obj1_opt, color = 'y')
plt.axhline(y=obj2_opt, color = 'y')
plt.title("Pareto Efficient Frontier")
plt.xlabel("Objective 1")
plt.ylabel("Objective 2")

```

[12]:

```
Text(0, 0.5, 'Objective 2')
```



## 6.7 Comments

---

This code is a bit inefficient. It probably computes more pareto points than needed.

## 6.8 Jupyter Notebooks

---

### Resources

- [https://github.com/mathinmse/mathinmse.github.io/blob/master/  
Lecture-00B-Notebook-Basics.ipynb](https://github.com/mathinmse/mathinmse.github.io/blob/master/Lecture-00B-Notebook-Basics.ipynb)
- [https://github.com/mathinmse/mathinmse.github.io/blob/master/  
Lecture-00C-Writing-In-Jupyter.ipynb](https://github.com/mathinmse/mathinmse.github.io/blob/master/Lecture-00C-Writing-In-Jupyter.ipynb)

## 6.9 Reading and Writing

---

[https://github.com/mathinmse/mathinmse.github.io/blob/master/  
Lecture-10B-Reading-and-Writing-Data.ipynb](https://github.com/mathinmse/mathinmse.github.io/blob/master/Lecture-10B-Reading-and-Writing-Data.ipynb)

## 6.10 Python Crash Course

---

<https://github.com/rpmuller/PythonCrashCourse>

## 6.11 Gurobi

---

Gurobi Log Tools

## 6.12 Plots, Pandas, and Geopandas

---

### 6.12.1. Geopandas

---

<https://jcutrer.com/python/learn-geopandas-plotting-usmaps>

<https://github.com/joncutter/geopandas-tutorial>

## 6.13 Linear Optimization

---

In this section, we study on linear optimization problems, i.e., linear programs (LPs).

### 6.13.1. Problem Formulation

---

Remember, for a linear program (LP), we want to maximize or minimize a linear **objective function** of the continuous decision variables, while considering linear constraints on the values of the decision variables.

#### Definition 6.1: Linear Function

*function  $f(x_1, x_2, \dots, x_n)$  is linear if, and only if, we have  $f(x_1, x_2, \dots, x_n) = c_1x_1 + c_2x_2 + \dots + c_nx_n$ , where the  $c_1, c_2, \dots, c_n$  coefficients are constants.*

#### A Generic Linear Program (LP)

---

##### Decision Variables:

$x_i$  : continuous variables ( $x_i \in \mathcal{R}$ , i.e., a real number),  $\forall i = 1, \dots, 3$ .

##### Parameters (known input parameters):

$c_i$  : cost coefficients  $\forall i = 1, \dots, 3$

$a_{ij}$  : constraint coefficients  $\forall i = 1, \dots, 3, j = 1, \dots, 4$

$b_j$  : right hand side coefficient for constraint  $j$ ,  $j = 1, \dots, 4$

$$\text{Min } z = c_1x_1 + c_2x_2 + c_3x_3 \quad (6.1)$$

$$\text{s.t. } a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \geq b_1 \quad (6.2)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \leq b_2 \quad (6.3)$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \quad (6.4)$$

$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 \geq b_4 \quad (6.5)$$

$$x_1 \geq 0, x_2 \leq 0, x_3 \text{ urs.} \quad (6.6)$$

Eq. (10.1) is the objective function, (10.2)-(10.5) are the functional constraints, while (10.6) is the sign restrictions (*urs* signifies that the variable is unrestricted). If we were to add any one of these following constraints  $x_2 \in \{0, 1\}$  ( $x_2$  is binary-valued) or  $x_3 \in \mathcal{Z}$  ( $x_3$  is integer-valued) we would have an Integer Program. For the purposes of this class, an Integer Program (IP) is just an LP with added integer restrictions on (some) variables.

While, in general, solvers will take any form of the LP, there are some special forms we use in analysis:

**LP Standard Form:** The standard form has all constraints as equalities, and all variables as non-negative. The generic LP is not in standard form, but any LP can be converted to standard form.

Since  $x_2$  is non-positive and  $x_3$  unrestricted, perform the following substitutions  $x_2 = -\hat{x}_2$  and  $x_3 = x_3^+ - x_3^-$ , where  $\hat{x}_2, x_3^+, x_3^- \geq 0$ . Eqs. (10.2) and (10.5) are in the form left-hand side (LHS)  $\geq$  right-hand side (RHS), so to make an equality, subtract a non-negative slack variable from the LHS ( $s_1$  and  $s_4$ ). Eq. (10.3) is in the form LHS  $\leq$  RHS, so add a non-negative slack variable to the LHS.

$$\begin{aligned} \text{Min } z &= c_1x_1 - c_2\hat{x}_2 + c_3(x_3^+ - x_3^-) \\ \text{s.t. } a_{11}x_1 - a_{12}x_2 + a_{13}(x_3^+ - x_3^-) - s_1 &= b_1 \\ a_{21}x_1 - a_{22}\hat{x}_2 + a_{23}(x_3^+ - x_3^-) + s_2 &= b_2 \\ a_{31}x_1 - a_{32}\hat{x}_2 + a_{33}(x_3^+ - x_3^-) &= b_3 \\ a_{41}x_1 - a_{42}\hat{x}_2 + a_{43}x_3 - s_4 &= b_4 \\ x_1, \hat{x}_2, x_3^+, x_3^-, s_1, s_2, s_4 &\geq 0. \end{aligned}$$

**LP Canonical Form:** For a minimization problem the canonical form of the LP has the LHS of each constraint greater than or equal to the the RHS, and a maximization the LHS less than or equal to the RHS, and non-negative variables.

Next we consider some formulation examples:

**Production Problem:** You have 21 units of transparent aluminum alloy (TAA), LazWeld1, a joining robot leased for 23 hours, and CrumCut1, a cutting robot leased for 17 hours of aluminum cutting. You also have production code for a bookcase, desk, and cabinet, along with commitments to buy any of these you can produce for \$18, \$16, and \$10 apiece, respectively. A bookcase requires 2 units of TAA, 3 hours of joining, and 1 hour of cutting, a desk requires 2 units of TAA, 2 hours of joining, and 2 hour of cutting, and a cabinet requires 1 unit of TAA, 2 hours of joining, and 1 hour of cutting. Formulate an LP to maximize your revenue given your current resources.

Decision variables:

$x_i$  : number of units of product  $i$  to produce,

$\forall i = \{\text{bookcase, desk, cabinet}\}$ .

$$\begin{aligned} \max z &= 18x_1 + 16x_2 + 10x_3 : \\ 2x_1 + 2x_2 + 1x_3 &\leq 21 && (\text{TAA}) \\ 3x_1 + 2x_2 + 2x_3 &\leq 23 && (\text{LazWeld1}) \\ 1x_1 + 2x_2 + 1x_3 &\leq 17 && (\text{CrumCut1}) \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

**Work Scheduling Problem:** You are the manager of LP Burger. The following table shows the minimum number of employees required to staff the restaurant on each day of the week. Each employees must work for five consecutive days. Formulate an LP to find the minimum number of employees required to staff the restaurant.

Decision variables:

$x_i$  : the number of workers that start 5 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$

Day of Week	Workers Required
1 = Monday	6
2 = Tuesday	4
3 = Wednesday	5
4 = Thursday	4
5 = Friday	3
6 = Saturday	7
7 = Sunday	7

$$\begin{aligned}
 \text{Min } z &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\
 \text{s.t. } x_1 + x_4 + x_5 + x_6 + x_7 &\geq 6 \\
 x_2 + x_5 + x_6 + x_7 + x_1 &\geq 4 \\
 x_3 + x_6 + x_7 + x_1 + x_2 &\geq 5 \\
 x_4 + x_7 + x_1 + x_2 + x_3 &\geq 4 \\
 x_5 + x_1 + x_2 + x_3 + x_4 &\geq 3 \\
 x_6 + x_2 + x_3 + x_4 + x_5 &\geq 7 \\
 x_7 + x_3 + x_4 + x_5 + x_6 &\geq 7 \\
 x_1, x_2, x_3, x_4, x_5, x_6, x_7 &\geq 0.
 \end{aligned}$$

The solution is as follows:

LP Solution	IP Solution
$z_{LP} = 7.333$	$z_I = 8.0$
$x_1 = 0$	$x_1 = 0$
$x_2 = 0.333$	$x_2 = 0$
$x_3 = 1$	$x_3 = 0$
$x_4 = 2.333$	$x_4 = 3$
$x_5 = 0$	$x_5 = 0$
$x_6 = 3.333$	$x_6 = 4$
$x_7 = 0.333$	$x_7 = 1$

LP Burger has changed its policy, and allows, at most, two part time workers, who work for two consecutive days in a week. Formulate this problem.

Decision variables:

$x_i$  : the number of workers that start 5 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$

$y_i$  : the number of workers that start 2 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$ .

$$\begin{aligned}
\text{Min } z &= 5(x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7) \\
&\quad + 2(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7) \\
\text{s.t. } x_1 + x_4 + x_5 + x_6 + x_7 + y_1 + y_7 &\geq 6 \\
x_2 + x_5 + x_6 + x_7 + x_1 + y_2 + y_1 &\geq 4 \\
x_3 + x_6 + x_7 + x_1 + x_2 + y_3 + y_2 &\geq 5 \\
x_4 + x_7 + x_1 + x_2 + x_3 + y_4 + y_3 &\geq 4 \\
x_5 + x_1 + x_2 + x_3 + x_4 + y_5 + y_4 &\geq 3 \\
x_6 + x_2 + x_3 + x_4 + x_5 + y_6 + y_5 &\geq 7 \\
x_7 + x_3 + x_4 + x_5 + x_6 + y_7 + y_6 &\geq 7 \\
y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 &\leq 2 \\
x_i &\geq 0, y_i \geq 0, \forall i = 1, \dots, 7.
\end{aligned}$$

**The Diet Problem:** In the future (as envisioned in a bad 70's science fiction film) all food is in tablet form, and there are four types, green, blue, yellow, and red. A balanced, futuristic diet requires, at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D. Formulate an LP that ensures a balanced diet at the minimum possible cost.

Tablet	Iron	B	C	D	Cost (\$)
green (1)	6	6	7	4	1.25
blue (2)	4	5	4	9	1.05
yellow (3)	5	2	5	6	0.85
red (4)	3	6	3	2	0.65

Now we formulate the problem:

Decision variables:

$x_i$  : number of tablet of type  $i$  to include in the diet,  $\forall i \in \{1, 2, 3, 4\}$ .

$$\begin{aligned}
\text{Min } z &= 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4 \\
\text{s.t. } 6x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 20 \\
6x_1 + 5x_2 + 2x_3 + 6x_4 &\geq 25 \\
7x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 30 \\
4x_1 + 9x_2 + 6x_3 + 2x_4 &\geq 15 \\
x_1, x_2, x_3, x_4 &\geq 0.
\end{aligned}$$

**The Next Diet Problem:** Progress is important, and our last problem had too many tablets, so we are going to produce a single, purple, 10 gram tablet for our futuristic diet requires, which are at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D, and 2000 calories. The tablet is made from blending 4 nutritious chemicals; the following table shows the units of our nutrients

Tablet	Iron	B	C	D	Calories	Cost (\$)
Chem 1	6	6	7	4	1000	1.25
Chem 2	4	5	4	9	250	1.05
Chem 3	5	2	5	6	850	0.85
Chem 4	3	6	3	2	750	0.65

per, and cost of, grams of each chemical. Formulate an LP that ensures a balanced diet at the minimum possible cost.

#### Decision variables:

$x_i$  : grams of chemical  $i$  to include in the purple tablet,  $\forall i = 1, 2, 3, 4$ .

$$\begin{aligned}
 \text{Min } z &= 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4 \\
 \text{s.t. } 6x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 20 \\
 6x_1 + 5x_2 + 2x_3 + 6x_4 &\geq 25 \\
 7x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 30 \\
 4x_1 + 9x_2 + 6x_3 + 2x_4 &\geq 15 \\
 1000x_1 + 250x_2 + 850x_3 + 750x_4 &\geq 2000 \\
 x_1 + x_2 + x_3 + x_4 &= 10 \\
 x_1, x_2, x_3, x_4 &\geq 0.
 \end{aligned}$$

**The Assignment Problem:** Consider the assignment of  $n$  teams to  $n$  projects, where each team ranks the projects, where their favorite project is given a rank of  $n$ , their next favorite  $n - 1$ , and their least favorite project is given a rank of 1. The assignment problem is formulated as follows (we denote ranks using the  $R$ -parameter):

#### Variables:

$x_{ij}$  : 1 if project  $i$  assigned to team  $j$ , else 0.

$$\begin{aligned}
 \text{Max } z &= \sum_{i=1}^n \sum_{j=1}^n R_{ij}x_{ij} \\
 \text{s.t. } \sum_{i=1}^n x_{ij} &= 1, \quad \forall j = 1, \dots, n \\
 \sum_{j=1}^n x_{ij} &= 1, \quad \forall i = 1, \dots, n \\
 x_{ij} &\geq 0, \quad \forall i = 1, \dots, n, j = 1, \dots, n.
 \end{aligned}$$

The assignment problem has an integrality property, such that if we remove the binary restriction on the  $x$  variables (now just non-negative, i.e.,  $x_{ij} \geq 0$ ) then we still get binary assignments, despite the fact that it is now an LP. This property is very interesting and useful. Of course, the objective function might not quite what we want, we might be interested ensuring that the team with the worst assignment is as good as possible (a fairness criteria). One way of doing this is to modify the assignment problem using a max-min objective:

### Max-min Assignment-like Formulation

$$\begin{aligned}
 & \text{Max} \quad z \\
 \text{s.t.} \quad & \sum_{i=1}^n x_{ij} = 1, \quad \forall j = 1, \dots, n \\
 & \sum_{j=1}^n x_{ij} = 1, \quad \forall i = 1, \dots, n \\
 & x_{ij} \geq 0, \quad \forall i = 1, \dots, n, J = 1, \dots, n \\
 & z \leq \sum_{i=1}^n R_{ij} x_{ij}, \quad \forall j = 1, \dots, n.
 \end{aligned}$$

Does this formulation have the integrality property (it is not an assignment problem)? Consider a very simple example where two teams are to be assigned to two projects and the teams give the projects the following rankings: Both teams prefer Project 2. For both problems, if we remove the binary restriction on

	Project 1	Project 2
Team 1	2	1
Team 2	2	1

the  $x$ -variable, they can take values between (and including) zero and one. For the assignment problem the optimal solution will have  $z = 3$ , and fractional  $x$ -values will not improve  $z$ . For the max-min assignment problem this is not the case, the optimal solution will have  $z = 1.5$ , which occurs when each team is assigned half of each project (i.e., for Team 1 we have  $x_{11} = 0.5$  and  $x_{21} = 0.5$ ).

**Linear Data Models:** Consider a data set that consists of  $n$  data points  $(x_i, y_i)$ . We want to fit the best line to this data, such that given an  $x$ -value, we can predict the associated  $y$ -value. Thus, the form is  $y_i = \alpha x_i + \beta$  and we want to choose the  $\alpha$  and  $\beta$  values such that we minimize the error for our  $n$  data points.

**Variables:**

$e_i$  : error for data point  $i$ ,  $i = 1, \dots, n$ .

$\alpha$  : slope of fitted line.

$\beta$  : intercept of fitted line.

$$\begin{aligned} \text{Min } & \sum_{i=1}^n |e_i| \\ \text{s.t. } & \alpha x_i + \beta - y_i = e_i, \quad i = 1, \dots, n \\ & e_i, \alpha, \beta \text{ urs.} \end{aligned}$$

Of course, absolute values are not linear function, so we can linearize as follows:

**Decision variables:**

$e_i^+$  : positive error for data point  $i$ ,  $i = 1, \dots, n$ .

$e_i^-$  : negative error for data point  $i$ ,  $i = 1, \dots, n$ .

$\alpha$  : slope of fitted line.

$\beta$  : intercept of fitted line.

$$\begin{aligned} \text{Min } & \sum_{i=1}^n e_i^+ + e_i^- \\ \text{s.t. } & \alpha x_i + \beta - y_i = e_i^+ - e_i^-, \quad i = 1, \dots, n \\ & e_i^+, e_i^- \geq 0, \alpha, \beta \text{ urs.} \end{aligned}$$

**Two-Person Zero-Sum Games:** Consider a game with two players,  $\mathcal{A}$  and  $\mathcal{B}$ . In each round of the game,  $\mathcal{A}$  chooses one out of  $m$  possible actions, while  $\mathcal{B}$  chooses one out of  $n$  actions. If  $\mathcal{A}$  takes action  $j$  while  $\mathcal{B}$  takes action  $i$ , then  $c_{ij}$  is the payoff for  $\mathcal{A}$ , if  $c_{ij} > 0$ ,  $\mathcal{A}$  “wins”  $c_{ij}$  (and  $\mathcal{B}$  losses that amount), and if  $c_{ij} < 0$  if  $\mathcal{B}$  “wins”  $-c_{ij}$  (and  $\mathcal{A}$  losses that amount). This is a two-person zero-sum game.

Rock, Paper, Scissors is a two-person zero-sum game, with the following payoff matrix.

		$\mathcal{A}$		
		R	P	S
$\mathcal{B}$	R	0	1	-1
	P	-1	0	1
	S	1	-1	0

We can have a similar game, but with a different payoff matrix, as follows:

		$\mathcal{A}$		
		R	P	S
$\mathcal{B}$	R	4	-1	-1
	P	-2	4	-2
	S	-3	-3	4

What is the optimal strategy for  $\mathcal{A}$  (for either game)? We define  $x_j$  as the probability that  $\mathcal{A}$  takes action  $j$  (related to the columns). Then the payoff for  $\mathcal{A}$ , if  $\mathcal{B}$  takes action  $i$  is  $\sum_{j=1}^m c_{ij}x_j$ . Of course,  $\mathcal{A}$  does not know what action  $\mathcal{B}$  will take, so let's find a strategy that maximizes the minimum expected winnings of  $\mathcal{A}$  given any random strategy of  $\mathcal{B}$ , which we can formulate as follows:

$$\begin{aligned} \text{Max } & \left( \min_{i=1, \dots, n} \sum_{j=1}^m c_{ij}x_j \right) \\ \text{s.t. } & \sum_{j=1}^m x_j = 1 \\ & x_j \geq 0, \quad i = 1, \dots, m, \end{aligned}$$

which can be linearized as follows:

$$\begin{aligned} \text{Max } & z \\ \text{s.t. } & z \leq \sum_{j=1}^m c_{ij}x_j, \quad i = 1, \dots, n \\ & \sum_{j=1}^m x_j = 1 \\ & x_j \geq 0, \quad i = 1, \dots, m. \end{aligned}$$

The last two constraints ensure the that  $x_i$ -variables are valid probabilities. If you solved this LP for the first game (i.e., payoff matrix) you find the best strategy is  $x_1 = 1/3$ ,  $x_2 = 1/3$ , and  $x_3 = 1/3$  and there is no expected gain for player  $\mathcal{A}$ . For the second game, the best strategy is  $x_1 = 23/107$ ,  $x_2 = 37/107$ , and  $x_3 = 47/107$ , with  $\mathcal{A}$  gaining, on average,  $8/107$  per round.

## **Part II**

## **OLD LP Stuff**



# 7. LP Notes from Foundations of Applied Mathematics

---

## Linear Programs

---

A *linear program* is a linear constrained optimization problem. Such a problem can be stated in several different forms, one of which is

$$\begin{aligned} & \text{minimize} && \mathbf{c}^T \mathbf{x} \\ & \text{subject to} && G\mathbf{x} \leq \mathbf{h} \\ & && A\mathbf{x} = \mathbf{b}. \end{aligned}$$

The symbol  $\leq$  denotes that the components of  $G\mathbf{x}$  are less than the components of  $\mathbf{h}$ . In other words, if  $\mathbf{x} \leq \mathbf{y}$ , then  $x_i < y_i$  for all  $x_i \in \mathbf{x}$  and  $y_i \in \mathbf{y}$ .

Define vector  $\mathbf{s} \geq 0$  such that the constraint  $G\mathbf{x} + \mathbf{s} = \mathbf{h}$ . This vector is known as a *slack variable*. Since  $\mathbf{s} \geq 0$ , the constraint  $G\mathbf{x} + \mathbf{s} = \mathbf{h}$  is equivalent to  $G\mathbf{x} \leq \mathbf{h}$ .

With a slack variable, a new form of the linear program is found:

$$\begin{aligned} & \text{minimize} && \mathbf{c}^T \mathbf{x} \\ & \text{subject to} && G\mathbf{x} + \mathbf{s} = \mathbf{h} \\ & && A\mathbf{x} = \mathbf{b} \\ & && \mathbf{s} \geq 0. \end{aligned}$$

This is the formulation used by CVXOPT. It requires that the matrix  $A$  has full row rank, and that the block matrix  $[G \ A]^T$  has full column rank.

Consider the following example:

$$\begin{aligned} & \text{minimize} && -4x_1 - 5x_2 \\ & \text{subject to} && x_1 + 2x_2 \leq 3 \\ & && 2x_1 + x_2 = 3 \\ & && x_1, x_2 \geq 0 \end{aligned}$$

Recall that all inequalities must be less than or equal to, so that  $G\mathbf{x} \leq \mathbf{h}$ . Because the final two constraints are  $x_1, x_2 \geq 0$ , they need to be adjusted to be  $\leq$  constraints. This is easily done by multiplying by  $-1$ ,

resulting in the constraints  $-x_1, -x_2 \leq 0$ . If we define

$$G = \begin{bmatrix} 1 & 2 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \quad A = [2 \ 1], \quad \text{and} \quad \mathbf{b} = [3]$$

then we can express the constraints compactly as

$$\begin{aligned} G\mathbf{x} &\leq \mathbf{h}, \\ A\mathbf{x} &= \mathbf{b}, \end{aligned} \quad \text{where} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

By adding a slack variable  $\mathbf{s}$ , we can write our constraints as

$$G\mathbf{x} + \mathbf{s} = \mathbf{h},$$

which matches the form discussed above.

### Problem 7.1: Linear Optimization

Solve the following linear optimization problem:

$$\begin{aligned} \text{minimize} \quad & 2x_1 + x_2 + 3x_3 \\ \text{subject to} \quad & x_1 + 2x_2 \geq 3 \\ & 2x_1 + 10x_2 + 3x_3 \geq 10 \\ & x_i \geq 0 \text{ for } i = 1, 2, 3 \end{aligned}$$

Return the minimizer  $\mathbf{x}$  and the primal objective value.

(Hint: make the necessary adjustments so that all inequality constraints are  $\leq$  rather than  $\geq$ ).

## $l_1$ Norm

The  $l_1$  norm is defined

$$\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|.$$

A  $l_1$  minimization problem is minimizing a vector's  $l_1$  norm, while fitting certain constraints. It can be written in the following form:

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{x}\|_1 \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b}. \end{aligned}$$

This problem can be converted into a linear program by introducing an additional vector  $\mathbf{u}$  of length  $n$ . Define  $\mathbf{u}$  such that  $|x_i| \leq u_i$ . Thus,  $-u_i - x_i \leq 0$  and  $-u_i + x_i \leq 0$ . These two inequalities can be added to the linear system as constraints. Additionally, this means that  $\|\mathbf{x}\|_1 \leq \|\mathbf{u}\|_1$ . So minimizing  $\|\mathbf{u}\|_1$  subject

to the given constraints will in turn minimize  $\|\mathbf{x}\|_1$ . This can be written as follows:

$$\begin{aligned} \text{minimize} \quad & [1^T \quad 0^T] \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} -I & I \\ -I & -I \\ -I & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \\ & [0 \quad A] \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} = \mathbf{b}. \end{aligned}$$

Solving this gives values for the optimal  $\mathbf{u}$  and the optimal  $\mathbf{x}$ , but we only care about the optimal  $\mathbf{x}$ .

### Problem 7.2: $\ell_1$ Norm Minimization

Write a function called `l1Min()` that accepts a matrix  $A$  and vector  $\mathbf{b}$  as NumPy arrays and solves the  $\ell_1$  minimization problem. Return the minimizer  $\mathbf{x}$  and the primal objective value. Remember to first discard the unnecessary  $u$  values from the minimizer.

To test your function consider the matrix  $A$  and vector  $\mathbf{b}$  below.

$$A = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 0 & 3 & -2 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 7 \\ 4 \end{bmatrix}$$

The linear system  $A\mathbf{x} = \mathbf{b}$  has infinitely many solutions. Use `l1Min()` to verify that the solution which minimizes  $\|\mathbf{x}\|_1$  is approximately  $\mathbf{x} = [0., 2.571, 1.857, 0.]^T$  and the minimum objective value is approximately 4.429.

## The Transportation Problem

Consider the following transportation problem: A piano company needs to transport thirteen pianos from their three supply centers (denoted by 1, 2, 3) to two demand centers (4, 5). Transporting a piano from a supply center to a demand center incurs a cost, listed in Table 7.3. The company wants to minimize shipping costs for the pianos while meeting the demand.

Supply Center	Number of pianos available
1	7
2	2
3	4

Table 7.1: Number of pianos available at each supply center

Demand Center	Number of pianos needed
4	5
5	8

Table 7.2: Number of pianos needed at each demand center

Supply Center	Demand Center	Cost of transportation	Number of pianos
1	4	4	$p_1$
1	5	7	$p_2$
2	4	6	$p_3$
2	5	8	$p_4$
3	4	8	$p_5$
3	5	9	$p_6$

**Table 7.3: Cost of transporting one piano from a supply center to a demand center**

A system of constraints is defined for the variables  $p_1, p_2, p_3, p_4, p_5$ , and  $p_6$ . First, there cannot be a negative number of pianos so the variables must be nonnegative. Next, the Tables 7.1 and 7.2 define the following three supply constraints and two demand constraints:

$$\begin{aligned} p_1 + p_2 &= 7 \\ p_3 + p_4 &= 2 \\ p_5 + p_6 &= 4 \\ p_1 + p_3 + p_5 &= 5 \\ p_2 + p_4 + p_6 &= 8 \end{aligned}$$

The objective function is the number of pianos shipped from each location multiplied by the respective cost (found in Table 7.3):

$$4p_1 + 7p_2 + 6p_3 + 8p_4 + 8p_5 + 9p_6.$$

### NOTE

Since our answers must be integers, in general this problem turns out to be an NP-hard problem. There is a whole field devoted to dealing with integer constraints, called *integer linear programming*, which is beyond the scope of this lab. Fortunately, we can treat this particular problem as a standard linear program and still obtain integer solutions.

Recall the variables are nonnegative, so  $p_1, p_2, p_3, p_4, p_5, p_6 \geq 0$ . Thus,  $G$  and  $\mathbf{h}$  constrain the variables to be non-negative.

### Problem 7.3: Transportation problem

Solve the transportation problem by converting the last equality constraint into an inequality constraint. Return the minimizer  $\mathbf{x}$  and the primal objective value.

## Eating on a Budget

---

In 2009, the inmates of Morgan County jail convinced Judge Clemon of the Federal District Court in Birmingham to put Sheriff Barlett in jail for malnutrition. Under Alabama law, in order to encourage less spending, "the chief lawman could go light on prisoners' meals and pocket the leftover change."<sup>1</sup>. Sheriffs had to ensure a minimum amount of nutrition for inmates, but minimizing costs meant more money for the sheriffs themselves. Judge Clemon jailed Sheriff Barlett one night until a plan was made to use all allotted funds, 1.75 per inmate, to feed prisoners more nutritious meals. While this case made national news, the controversy of feeding prisoners in Alabama continues as of 2019<sup>2</sup>.

The problem of minimizing cost while reaching healthy nutritional requirements can be approached as a convex optimization problem. Rather than viewing this problem from the sheriff's perspective, we view it from the perspective of a college student trying to minimize food cost in order to pay for higher education, all while meeting standard nutritional guidelines.

The file `food.npy` contains a dataset with nutritional facts for 18 foods that have been eaten frequently by college students working on this text. A subset of this dataset can be found in Table 7.4, where the "Food" column contains the list of all 18 foods.

The columns of the full dataset are:

- Column 1:  $p$ , price (dollars)
- Column 2:  $s$ , number of servings
- Column 3:  $c$ , calories per serving
- Column 4:  $f$ , fat per serving (grams)
- Column 5:  $\hat{s}$ , sugar per serving (grams)
- Column 6:  $\hat{c}$ , calcium per serving (milligrams)
- Column 7:  $\hat{f}$ , fiber per serving (grams)
- Column 8:  $\hat{p}$ , protein per serving (grams)

---

<sup>1</sup>Nossiter, Adam, 8 Jan 2009, "As His Inmates Grew Thinner, a Sheriff's Wallet Grew Fatter", *New York Times*,<https://www.nytimes.com/2009/01/09/us/09sheriff.html>

<sup>2</sup>Sheets, Connor, 31 January 2019, "Alabama sheriffs urge lawmakers to get them out of the jail food business", <https://www.al.com/news/2019/01/alabama-sheriffs-urge-lawmakers-to-get-them-out-of-the-jail-food-business.html>

Food	Price $p$ dollars	Serving Size $s$	Calories $c$	Fat $f$ g	Sugar $\hat{s}$ g	Calcium $\hat{c}$ mg	Fiber $\hat{f}$ g	Protein $\hat{p}$ g
Ramen	6.88	48	190	7	0	0	0	5
Potatoes	0.48	1	290	0.4	3.2	53.8	6.9	7.9
Milk	1.79	16	130	5	12	250	0	8
Eggs	1.32	12	70	5	0	28	0	6
Pasta	3.88	8	200	1	2	0	2	7
Frozen Pizza	2.78	5	350	11	5	150	2	14
Potato Chips	2.12	14	160	11	1	0	1	1
Frozen Broccoli	0.98	4	25	0	1	25	2	1
Carrots	0.98	2	52.5	0.3	6.1	42.2	3.6	1.2
Bananas	0.24	1	105	0.4	14.4	5.9	3.1	1.3
Tortillas	3.48	18	140	4	0	0	0	3
Cheese	1.88	8	110	8	0	191	0	6
Yogurt	3.47	5	90	0	7	190	0	17
Bread	1.28	6	120	2	2	60	0.01	4
Chicken	9.76	20	110	3	0	0	0	20
Rice	8.43	40	205	0.4	0.1	15.8	0.6	4.2
Pasta Sauce	3.57	15	60	1.5	7	20	2	2
Lettuce	1.78	6	8	0.1	0.6	15.5	1	0.6

**Table 7.4:** Subset of table containing food data

According to the FDA<sup>1</sup> and US Department of Health, someone on a 2000 calorie diet should have no more than 2000 calories, no more than 65 grams of fat, no more than 50 grams of sugar<sup>2</sup>, at least 1000 milligrams of calcium<sup>1</sup>, at least 25 grams of fiber, and at least 46 grams of protein<sup>2</sup> per day.

We can rewrite this as a linear programming problem below.

<sup>1</sup>[urlhttps://www.accessdata.fda.gov/scripts/InteractiveNutritionFactsLabel/pdv.html](https://www.accessdata.fda.gov/scripts/InteractiveNutritionFactsLabel/pdv.html)

<sup>2</sup><https://www.today.com/health/4-rules-added-sugars-how-calculate-your-daily-limit-t34731>

<sup>1</sup>26 Sept 2018, <https://ods.od.nih.gov/factsheets/Calcium-HealthProfessional/>

<sup>2</sup><https://www.accessdata.fda.gov/scripts/InteractiveNutritionFactsLabel/protein.html>

$$\begin{aligned}
 & \text{minimize} \sum_{i=1}^{18} p_i x_i, \\
 & \text{subject to } \sum_{i=1}^{18} c_i x_i \leq 2000, \\
 & \quad \sum_{i=1}^{18} f_i x_i \leq 65, \\
 & \quad \sum_{i=1}^{18} \hat{s}_i x_i \leq 50, \\
 & \quad \sum_{i=1}^{18} \hat{c}_i x_i \geq 1000, \\
 & \quad \sum_{i=1}^{18} \hat{f}_i x_i \geq 25, \\
 & \quad \sum_{i=1}^{18} \hat{p}_i x_i \geq 46, \\
 & \quad x_i \geq 0.
 \end{aligned}$$

#### Problem 7.4: Eating on a Budget

Read in the file `food.npy`. Identify how much of each food item a college student should eat to minimize cost spent each day. Return the minimizing vector and the total amount of money spent. What is the food you should eat most each day? What are the three foods you should eat most each week?

(Hint: Each nutritional value must be multiplied by the number of servings to get the nutrition value of the whole product).

## 7.1 The Simplex Method

---

**The Simplex Method Lab Objective:** *The Simplex Method is a straightforward algorithm for finding optimal solutions to optimization problems with linear constraints and cost functions. Because of its simplicity and applicability, this algorithm has been named one of the most important algorithms invented within the last 100 years. In this lab we implement a standard Simplex solver for the primal problem.*

## Standard Form

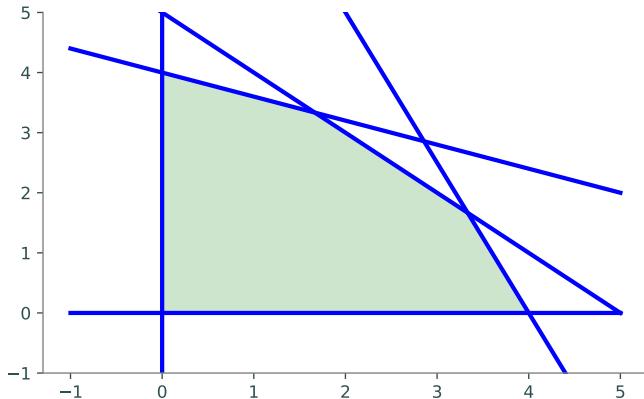
---

The Simplex Algorithm accepts a linear constrained optimization problem, also called a *linear program*, in the form given below:

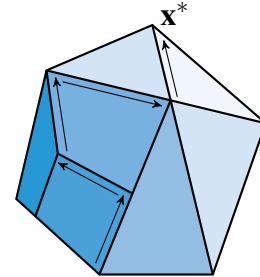
$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

Note that any linear program can be converted to standard form, so there is no loss of generality in restricting our attention to this particular formulation.

Such an optimization problem defines a region in space called the *feasible region*, the set of points satisfying the constraints. Because the constraints are all linear, the feasible region forms a geometric object called a *polytope*, having flat faces and edges (see Figure 7.1). The Simplex Algorithm jumps among the vertices of the feasible region searching for an optimal point. It does this by moving along the edges of the feasible region in such a way that the objective function is always increased after each move.



(a) The feasible region for a linear program with 2-dimensional constraints.



(b) The feasible region for a linear program with 3-dimensional constraints.

**Figure 7.1:** If an optimal point exists, it is one of the vertices of the polyhedron. The simplex algorithm searches for optimal points by moving between adjacent vertices in a direction that increases the value of the objective function until it finds an optimal vertex.

Implementing the Simplex Algorithm is straightforward, provided one carefully follows the procedure. We will break the algorithm into several small steps, and write a function to perform each one. To become familiar with the execution of the Simplex algorithm, it is helpful to work several examples by hand.

## The Simplex Solver

---

Our program will be more lengthy than many other lab exercises and will consist of a collection of functions working together to produce a final result. It is important to clearly define the task of each function and how all the functions will work together. If this program is written haphazardly, it will be much longer and more difficult to read than it needs to be. We will walk you through the steps of implementing the Simplex Algorithm as a Python class.

For demonstration purposes, we will use the following linear program.

$$\begin{array}{ll} \text{minimize} & -3x_0 - 2x_1 \\ \text{subject to} & x_0 - x_1 \leq 2 \\ & 3x_0 + x_1 \leq 5 \\ & 4x_0 + 3x_1 \leq 7 \\ & x_0, x_1 \geq 0. \end{array}$$

## Accepting a Linear Program

---

Our first task is to determine if we can even use the Simplex algorithm. Assuming that the problem is presented to us in standard form, we need to check that the feasible region includes the origin. For now, we only check for feasibility at the origin. A more robust solver sets up the auxiliary problem and solves it to find a starting point if the origin is infeasible.

### Problem 7.5: Check feasibility at the origin.

*Write a class that accepts the arrays  $\mathbf{c}$ ,  $A$ , and  $\mathbf{b}$  of a linear optimization problem in standard form. In the constructor, check that the system is feasible at the origin. That is, check that  $A\mathbf{x} \leq \mathbf{b}$  when  $\mathbf{x} = 0$ . Raise a `ValueError` if the problem is not feasible at the origin.*

## Adding Slack Variables

---

The next step is to convert the inequality constraints  $A\mathbf{x} \leq \mathbf{b}$  into equality constraints by introducing a slack variable for each constraint equation. If the constraint matrix  $A$  is an  $m \times n$  matrix, then there are  $m$  slack variables, one for each row of  $A$ . Grouping all of the slack variables into a vector  $\mathbf{w}$  of length  $m$ , the constraints now take the form  $A\mathbf{x} + \mathbf{w} = \mathbf{b}$ . In our example, we have

$$\mathbf{w} = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

When adding slack variables, it is useful to represent all of your variables, both the original primal variables and the additional slack variables, in a convenient manner. One effective way is to refer to a variable

by its subscript. For example, we can use the integers 0 through  $n - 1$  to refer to the original (non-slack) variables  $x_0$  through  $x_{n-1}$ , and we can use the integers  $n$  through  $n + m - 1$  to track the slack variables (where the slack variable corresponding to the  $i$ th row of the constraint matrix is represented by the index  $n + i - 1$ ).

We also need some way to track which variables are *independent* (non-zero) and which variables are *dependent* (those that have value 0). This can be done using the objective function. At anytime during the optimization process, the non-zero variables in the objective function are *independent* and all other variables are *dependent*.

## Creating a Dictionary

---

After we have determined that our program is feasible, we need to create the *dictionary* (sometimes called the *tableau*), a matrix to track the state of the algorithm.

There are many different ways to build your dictionary. One way is to mimic the dictionary that is often used when performing the Simplex Algorithm by hand. To do this we will set the corresponding dependent variable equations to 0. For example, if  $x_5$  were a dependent variable we would expect to see a -1 in the column that represents  $x_5$ . Define

$$\bar{A} = [ A \ I_m ],$$

where  $I_m$  is the  $m \times m$  identity matrix we will use to represent our slack variables, and define

$$\bar{\mathbf{c}} = \begin{bmatrix} \mathbf{c} \\ 0 \end{bmatrix}.$$

That is,  $\bar{\mathbf{c}} \in \mathbb{R}^{n+m}$  such that the first  $n$  entries are  $\mathbf{c}$  and the final  $m$  entries are zeros. Then the initial dictionary has the form

$$D = \begin{bmatrix} 0 & \bar{\mathbf{c}}^T \\ \mathbf{b} & -\bar{A} \end{bmatrix} \quad (7.1)$$

The columns of the dictionary correspond to each of the variables (both primal and slack), and the rows of the dictionary correspond to the dependent variables.

For our example the initial dictionary is

$$D = \begin{bmatrix} 0 & -3 & -2 & 0 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5 & -3 & -1 & 0 & -1 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{bmatrix}.$$

The advantage of using this kind of dictionary is that it is easy to check the progress of your algorithm by hand.

### Problem 7.6: Initialize the dictionary.

*dd a method to your Simplex solver that takes in arrays  $c$ ,  $A$ , and  $b$  to create the initial dictionary ( $D$ ) as a NumPy array.*

### 7.1.1. Pivoting

---

Pivoting is the mechanism that really makes Simplex useful. Pivoting refers to the act of swapping dependent and independent variables, and transforming the dictionary appropriately. This has the effect of moving from one vertex of the feasible polytope to another vertex in a way that increases the value of the objective function. Depending on how you store your variables, you may need to modify a few different parts of your solver to reflect this swapping.

When initiating a pivot, you need to determine which variables will be swapped. In the dictionary representation, you first find a specific element on which to pivot, and the row and column that contain the pivot element correspond to the variables that need to be swapped. Row operations are then performed on the dictionary so that the pivot column becomes a negative elementary vector.

Let's break it down, starting with the pivot selection. We need to use some care when choosing the pivot element. To find the pivot column, search from left to right along the top row of the dictionary (ignoring the first column), and stop once you encounter the first negative value. The index corresponding to this column will be designated the *entering index*, since after the full pivot operation, it will enter the basis and become a dependent variable.

Using our initial dictionary  $D$  in the example, we stop at the second column:

$$D = \left[ \begin{array}{c|ccccc} 0 & -3 & -2 & 0 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5 & -3 & -1 & 0 & -1 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{array} \right]$$

We now know that our pivot element will be found in the second column. The entering index is thus 1.

Next, we select the pivot element from among the negative entries in the pivot column (ignoring the entry in the first row). *If all entries in the pivot column are non-negative, the problem is unbounded and has no solution.* In this case, the algorithm should terminate. Otherwise, assuming our pivot column is the  $j$ th column of the dictionary and that the negative entries of this column are  $D_{i_1,j}, D_{i_2,j}, \dots, D_{i_k,j}$ , we calculate the ratios

$$\frac{-D_{i_1,0}}{D_{i_1,j}}, \frac{-D_{i_2,0}}{D_{i_2,j}}, \dots, \frac{-D_{i_k,0}}{D_{i_k,j}},$$

and we choose our pivot element to be one that minimizes this ratio. If multiple entries minimize the ratio, then we utilize *Bland's Rule*, which instructs us to choose the entry in the row corresponding to the smallest index (obeying this rule is important, as it prevents the possibility of the algorithm cycling back on itself infinitely). The index corresponding to the pivot row is designated as the *leaving index*, since after the full pivot operation, it will leave the basis and become an independent variable.

In our example, we see that all entries in the pivot column (ignoring the entry in the first row, of course) are negative, and hence they are all potential choices for the pivot element. We then calculate the ratios, and obtain

$$\frac{-2}{-1} = 2, \quad \frac{-5}{-3} = 1.66\dots, \quad \frac{-7}{-4} = 1.75.$$

We see that the entry in the third row minimizes these ratios. Hence, the element in the second column (index 1), third row (index 2) is our designated pivot element.

$$D = \begin{bmatrix} 0 & -3 & -2 & 0 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5 & \boxed{-3} & -1 & 0 & -1 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{bmatrix}$$

**Definition 7.7: Bland's Rule**

choose the independent variable with the smallest index that has a negative coefficient in the objective function as the leaving variable. Choose the dependent variable with the smallest index among all the binding dependent variables.

Bland's Rule is important in avoiding cycles when performing pivots. This rule guarantees that a feasible Simplex problem will terminate in a finite number of pivots.

Finally, we perform row operations on our dictionary in the following way: divide the pivot row by the negative value of the pivot entry. Then use the pivot row to zero out all entries in the pivot column above and below the pivot entry. In our example, we first divide the pivot row by -3, and then zero out the two entries above the pivot element and the single entry below it:

$$\begin{array}{c} \left[ \begin{array}{cccccc} 0 & -3 & -2 & 0 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5 & -3 & -1 & 0 & -1 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{array} \right] \rightarrow \left[ \begin{array}{cccccc} 0 & -3 & -2 & 0 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{array} \right] \rightarrow \\ \left[ \begin{array}{cccccc} -5 & 0 & -1 & 0 & 1 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{array} \right] \rightarrow \left[ \begin{array}{cccccc} -5 & 0 & -1 & 0 & 1 & 0 \\ 1/3 & 0 & -4/3 & 1 & -1/3 & 0 \\ 5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{array} \right] \rightarrow \\ \left[ \begin{array}{cccccc} -5 & 0 & -1 & 0 & 1 & 0 \\ 1/3 & 0 & 4/3 & -1 & 1/3 & 0 \\ 5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\ 1/3 & 0 & -5/3 & 0 & 4/3 & -1 \end{array} \right]. \end{array}$$

The result of these row operations is our updated dictionary, and the pivot operation is complete.

**Problem 7.8: Pivoting**

Add a method to your solver that checks for unboundedness and performs a single pivot operation from start to completion. If the problem is unbounded, raise a `ValueError`.

### 7.1.2. Termination and Reading the Dictionary

---

Up to this point, our algorithm accepts a linear program, adds slack variables, and creates the initial dictionary. After carrying out these initial steps, it then performs the pivoting operation iteratively until the optimal point is found. But how do we determine when the optimal point is found? The answer is to look at the top row of the dictionary, which represents the objective function. More specifically, before each pivoting operation, check whether all of the entries in the top row of the dictionary (ignoring the entry in the first column) are nonnegative. If this is the case, then we have found an optimal solution, and so we terminate the algorithm.

The final step is to report the solution. The ending state of the dictionary and index list tells us everything we need to know. The minimal value attained by the objective function is found in the upper leftmost entry of the dictionary. The dependent variables all have the value 0 in the objective function or first row of our dictionary array. The independent variables have values given by the first column of the dictionary. Specifically, the independent variable whose index is located at the  $i$ th entry of the index list has the value  $T_{i+1,0}$ .

In our example, suppose that our algorithm terminates with the dictionary and index list in the following state:

$$D = \begin{bmatrix} -5.2 & 0 & 0 & 0 & 0.2 & 0.6 \\ 0.6 & 0 & 0 & -1 & 1.4 & -0.8 \\ 1.6 & -1 & 0 & 0 & -0.6 & 0.2 \\ 0.2 & 0 & -1 & 0 & 0.8 & -0.6 \end{bmatrix}$$

Then the minimal value of the objective function is  $-5.2$ . The independent variables have indices 4, 5 and have the value 0. The dependent variables have indices 3, 1, and 2, and have values .6, 1.6, and .2, respectively. In the notation of the original problem statement, the solution is given by

$$\begin{aligned} x_0 &= 1.6 \\ x_1 &= .2. \end{aligned}$$

#### Problem 7.9: SimplexSolver.solve()

*Write an additional method in your solver called `solve()` that obtains the optimal solution, then returns the minimal value, the dependent variables, and the independent variables. The dependent and independent variables should be represented as two dictionaries that map the index of the variable to its corresponding value.*

*For our example, we would return the tuple*

*( $-5.2$ , {0: 1.6, 1: .2, 2: .6}, {3: 0, 4: 0}).*

At this point, you should have a Simplex solver that is ready to use. The following code demonstrates how your solver is expected to behave:

```
>>> import SimplexSolver

# Initialize objective function and constraints.
>>> c = np.array([-3., -2.])
>>> b = np.array([2., 5, 7])
>>> A = np.array([[1., -1], [3, 1], [4, 3]])

# Instantiate the simplex solver, then solve the problem.
>>> solver = SimplexSolver(c, A, b)
>>> sol = solver.solve()
>>> print(sol)
(-5.2,
 {0: 1.6, 1: 0.2, 2: 0.6},
 {3: 0, 4: 0})
```

If the linear program were infeasible at the origin or unbounded, we would expect the solver to alert the user by raising an error.

Note that this simplex solver is *not* fully operational. It can't handle the case of infeasibility at the origin. This can be fixed by adding methods to your class that solve the *auxiliary problem*, that of finding an initial feasible dictionary when the problem is not feasible at the origin. Solving the auxiliary problem involves pivoting operations identical to those you have already implemented, so adding this functionality is not overly difficult.

## 7.2 The Product Mix Problem

---

We now use our Simplex implementation to solve the *product mix problem*, which in its dependent form can be expressed as a simple linear program. Suppose that a manufacturer makes  $n$  products using  $m$  different resources (labor, raw materials, machine time available, etc). The  $i$ th product is sold at a unit price  $p_i$ , and there are at most  $m_j$  units of the  $j$ th resource available. Additionally, each unit of the  $i$ th product requires  $a_{j,i}$  units of resource  $j$ . Given that the demand for product  $i$  is  $d_i$  units per a certain time period, how do we choose the optimal amount of each product to manufacture in that time period so as to maximize revenue, while not exceeding the available resources?

Let  $x_1, x_2, \dots, x_n$  denote the amount of each product to be manufactured. The sale of product  $i$  brings revenue in the amount of  $p_i x_i$ . Therefore our objective function, the profit, is given by

$$\sum_{i=1}^n p_i x_i.$$

Additionally, the manufacture of product  $i$  requires  $a_{j,i} x_i$  units of resource  $j$ . Thus we have the resource constraints

$$\sum_{i=1}^n a_{j,i} x_i \leq m_j \text{ for } j = 1, 2, \dots, m.$$

Finally, we have the demand constraints which tell us not to exceed the demand for the products:

$$x_i \leq d_i \text{ for } i = 1, 2, \dots, n$$

The variables  $x_i$  are constrained to be nonnegative, of course. We therefore have a linear program in the appropriate form that is feasible at the origin. It is a simple task to solve the problem using our Simplex solver.

### Problem 7.10: Product mix problem.

Solve the product mix problem for the data contained in the file `productMix.npz`. In this problem, there are 4 products and 3 resources. The archive file, which you can load using the function `np.load`, contains a dictionary of arrays. The array with key '`A`' gives the resource coefficients  $a_{i,j}$  (i.e. the  $(i, j)$ -th entry of the array give  $a_{i,j}$ ). The array with key '`p`' gives the unit prices  $p_i$ . The array with key '`m`' gives the available resource units  $m_j$ . The array with key '`d`' gives the demand constraints  $d_i$ .

Report the number of units that should be produced for each product. Hint: Because this is a maximization problem and your solver works with minimizations, you will need to change the sign of the array  $c$ .

## Beyond Simplex

The *Computing in Science and Engineering* journal listed Simplex as one of the top ten algorithms of the twentieth century [Nash2000]. However, like any other algorithm, Simplex has its drawbacks.

In 1972, Victor Klee and George Minty Cube published a paper with several examples of worst-case polytopes for the Simplex algorithm [Klee1972]. In their paper, they give several examples of polytopes that the Simplex algorithm struggles to solve.

Consider the following linear program from Klee and Minty.

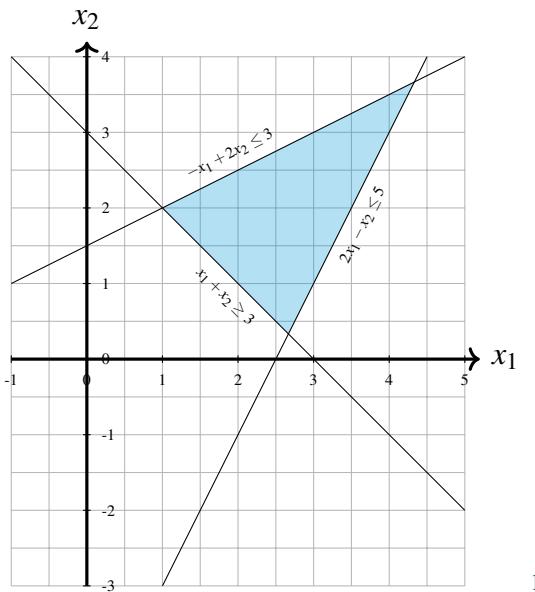
$$\begin{array}{lllll} \max & 2^{n-1}x_1 & + 2^{n-2}x_2 & + \cdots & + 2x_{n-1} & + x_n \\ \text{subject to } & x_1 & & & & \leq 5 \\ & 4x_1 & & + x_2 & & \leq 25 \\ & 8x_1 & & + 4x_2 & + x_3 & \leq 125 \\ & \vdots & & & & \vdots \\ & 2^n x_1 & & + 2^{n-1}x_2 & + \cdots & + 4x_{n-1} & + x_n \leq 5 \end{array}$$

Klee and Minty show that for this example, the worst case scenario has exponential time complexity. With only  $n$  constraints and  $n$  variables, the simplex algorithm goes through  $2^n$  iterations. This is because there are  $2^n$  extreme points, and when starting at the point  $x = 0$ , the simplex algorithm goes through all of the extreme points before reaching the optimal point  $(0, 0, \dots, 0, 5^n)$ . Other algorithms, such as interior point methods, solve this problem much faster because they are not constrained to follow the edges.



# 8. Linear Programming Notes - Hildebrand

---



1

## 8.0.0.1. Simplex Tableau Pivot

---

[http://www.tutor-homework.com/Simplex\\_Tableau\\_Homework\\_Help.html](http://www.tutor-homework.com/Simplex_Tableau_Homework_Help.html)

## 8.0.0.2. Videos

---

Geometry of the simplex method: Fantastic video by Craig Torey of Georgia Tech explaining geometry of pivots and why the simplex method is called the simplex method. There is also a bit of history about Dantzig in the video.

[https://www.youtube.com/watch?v=Ci1vBGn9yRc&ab\\_channel=LouisHolley](https://www.youtube.com/watch?v=Ci1vBGn9yRc&ab_channel=LouisHolley)

For some nice videos of doing simplex method with tableaus, I recommend:

<https://www.youtube.com/watch?v=M8POtpPtQZc>

LPP using simplex method [Minimization with 3 variables]: <https://youtu.be/SNc9NGCJmns>

LPP using Dual simplex method: <https://youtu.be/KLHWtBpPbEc>

LPP using TWO PHASE method: <https://youtu.be/zJhncZ5XUSU>

LPP using BIG M method: <https://youtu.be/MZ843Vvia0A>

[1] LPP using Graphical method [Maximization with 2 constraints]: <https://youtu.be/8IRrgDoV8Eo>

[2] LPP using Graphical method [Minimization with 3 constraints]: [https://youtu.be/06Q03J\\_85as](https://youtu.be/06Q03J_85as)

<sup>1</sup><https://tex.stackexchange.com/questions/75933/how-to-draw-the-region-of-inequality>

## 8.1 Linear Programming Forms

---

## 8.2 Linear Programming Dual

---

Consider the linear program in standard form. The dual is the following problem

**Dual of LP in Standard Form:**

*Polynomial time (P)*

**Primal**

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

**Dual**

$$\begin{aligned} \min \quad & b^\top y \\ \text{s.t.} \quad & A^\top y \geq c \\ & y \text{ free} \end{aligned} \tag{8.1}$$

## 8.3 Weak and Strong Duality

---

**Theorem 8.1: Weak Duality**

Let  $x$  be feasible for the primal LP and  $y$  feasible for the dual LP. Then

$$c^\top x \leq b^\top y. \tag{8.1}$$

**Theorem 8.2: Strong Duality**

The primal LP is feasible and has a bounded objective value if and only if the dual LP is also feasible and has a bounded objective value. In this case, the optimal values to both problems coincide.

In particular, suppose  $x^*$  is optimal for the primal LP and  $y^*$  is optimal for the dual LP.

Then

$$c^\top x^* = b^\top y^*. \tag{8.2}$$

### 8.3.1. Reduced Costs

---

Consider the LP in standard form (2.2) given by

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{8.3}$$

A basis  $B$  is a subset of the columns of  $A$  that form an invertible matrix. The remaining columns for the matrix  $N$ , that is,  $A = (A_B | A_N)$  (after permuting the columns of  $A$ ).

The *basic variables*  $x_B$  are the variables corresponding to the columns of  $A_B$  and the *non-basic variables* are those corresponding to the columns of  $A_N$ .

Since  $A_B$  is invertible we can convert the formulation by multiplying through by  $A_B^{-1}$ . This produces

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & A_B^{-1}Ax = A_B^{-1}b \\ & x \geq 0 \end{aligned} \tag{8.4}$$

Since  $A = (A_B | A_N)$ , we have

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & (A_B^{-1}A_B, A_B^{-1}A_N)x = A_B^{-1}b \\ & x \geq 0 \end{aligned} \tag{8.5}$$

which becomes

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & x_B + A_B^{-1}A_Nx_N = A_B^{-1}b \\ & x \geq 0 \end{aligned} \tag{8.6}$$

$B^{-1}b \geq 0$ , then  $x_B = B^{-1}b, x_N = 0$  called a *basic feasible solution*.

Manipulating the formulation again, we can multiply the equations by  $c_B$  and subtract that from the objective function. This leaves us with

$$\begin{aligned} \max \quad & c_Nx_N - c_BA_B^{-1}A_Nx_N + c_BA_B^{-1}b \\ \text{s.t.} \quad & x_B + A_B^{-1}A_Nx_N = A_B^{-1}b \\ & x \geq 0 \end{aligned} \tag{8.7}$$

combining terms creates

$$\begin{aligned} \max \quad & (c_N - c_BA_B^{-1}A_N)x_N + c_BA_B^{-1}b \\ \text{s.t.} \quad & x_B + A_B^{-1}A_Nx_N = A_B^{-1}b \\ & x \geq 0 \end{aligned} \tag{8.8}$$

Now clearly we see that if  $A_B^{-1}b \geq 0$ , then setting  $x = (x_B, x_N) = (A_B^{-1}b, 0)$  is a feasible solution with objective value  $c_B A_B^{-1}b$ .

We say that the quantity

$$\tilde{c}_N = c_N - c_B A_B^{-1} A_N.$$

are the *reduced costs*.

Notice that we can re-write the equation above as

$$\begin{aligned} \max \quad & \tilde{c}_N x_N + c_B A_B^{-1} b \\ \text{s.t.} \quad & x_B + A_B^{-1} A_N x_N = A_B^{-1} b \\ & x \geq 0 \end{aligned} \tag{8.9}$$

Hence, viewing this LP from the basis  $B$  illuminates that change in objective function as we increase the non-basic variables from 0.

### 8.3.2. Tableau Based Pivoting

---

In this section, we discuss how to solve a linear program using a *tableau*. A tableau is just a table to record calculations in a convenient way.

#### Example 8.3

Solve this linear program using a tableau based approach.

$$\begin{aligned} \text{Minimize } Z &= 2x_1 + 3x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 16 \\ & x_1 + 3x_2 \geq 20 \\ & x_1 + x_2 = 10 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{8.10}$$

We will use the *Big-M* method to solve this problem. We begin by converting the problem into standard form.

$$\begin{aligned} \text{Maximize } -Z &= -2x_1 - 3x_2 - M\bar{x}_5 - M\bar{x}_6 \\ \text{s.t.} \quad & 2x_1 + x_2 + x_3 = 16 \\ & x_1 + 3x_2 - x_4 + \bar{x}_5 = 20 \\ & x_1 + x_2 + \bar{x}_6 = 10 \\ & x_1, x_2, x_3, x_4, \bar{x}_5, \bar{x}_6 \geq 0 \end{aligned}$$

Initial Set-up

Basic Variable	Z	Coefficient of:						Right Side
		$x_1$	$x_2$	$x_3$	$x_4$	$\bar{x}_5$	$\bar{x}_6$	
Z	-1	2	3	0	0	M	M	0
$x_3$	0	2	1	1	0	0	0	16
$\bar{x}_5$	0	1	3	0	-1	1	0	20
$\bar{x}_6$	0	1	1	0	0	0	1	10

Standard Form

Basic Variable	Z	Coefficient of:						Right Side
		$x_1$	$x_2$	$x_3$	$x_4$	$\bar{x}_5$	$\bar{x}_6$	
Z	-1	2 - 2M	3 - 4M	0	M	0	0	-30M
$x_3$	0	2		1	1	0	0	16
$\bar{x}_5$	0	1		3	0	-1	1	20
$\bar{x}_6$	0	1		1	0	0	1	10

Iteration 1 - Let  $x_2$  enter and  $\bar{x}_5$  leaves.

Basic Variable	Z	Coefficient of:						Right Side
		$x_1$	$x_2$	$x_3$	$x_4$	$\bar{x}_5$	$\bar{x}_6$	
Z	-1	1 - 2M/3	0	0	1 - M/3	-1 + 4M/3	0	-20 - 10M/3
$x_3$	0	2/3	0	1	1/3	-1/3	0	-28/3
$x_2$	0	1/3	1	0	-1/3	1/3	0	20/3
$\bar{x}_6$	0	2/3	0	0	1/3	-1/3	1	10/3

Iteration 2 - Let  $x_1$  enter and  $\bar{x}_6$  leaves.

Basic Variable	Z	Coefficient of:						Right Side
		$x_1$	$x_2$	$x_3$	$x_4$	$\bar{x}_5$	$\bar{x}_6$	
Z	-1	0	0	0	1/2	-1/2 + M	-3/2 + M	-25
$x_3$	0	0	0	1	-1/2	1/2	-5/2	1
$x_2$	0	0	1	0	-1/2	1/2	-1/2	5
$x_1$	0	1	0	0	1/2	-1/2	3/2	5

We have reached an optimal solution since all coefficients in the objective function are positive. Thus our solution to the initial minimization problem is

$$x_1 = 5, \quad x_2 = 5, \quad Z(5,5) = 25 \quad \text{with slack } x_3 = 1$$



# 9. Linear Programming Book - Cheung

---

## Preface

---

This book covers the fundamentals of linear programming through studying systems of linear inequalities using only basic facts from linear algebra. It is suitable for a crash course on linear programming that emphasizes theoretical aspects of the subject. Discussion on practical solution methods such as the simplex method and interior point methods, though not present in this book, is planned for a future book.

Two excellent references for further study are [**Bertsimas:1997**] and [**Schrijver:1986**].



The book is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

## Notation

---

The set of real numbers is denoted by  $\mathbb{R}$ . The set of rational numbers is denoted by  $\mathbb{Q}$ . The set of integers is denoted by  $\mathbb{Z}$ .

The set of  $n$ -tuples with real entries is denoted by  $\mathbb{R}^n$ . Similar definitions hold for  $\mathbb{Q}^n$  and  $\mathbb{Z}^n$ .

The set of  $m \times n$  matrices (that is, matrices with  $m$  rows and  $n$  columns) with real entries is denoted  $\mathbb{R}^{m \times n}$ . Similar definitions hold for  $\mathbb{Q}^{m \times n}$  and  $\mathbb{Z}^{m \times n}$ .

All  $n$ -tuples are written as columns (that is, as  $n \times 1$  matrices). An  $n$ -tuple is normally represented by a lowercase Roman letter in boldface; for example,  $\mathbf{x}$ . For an  $n$ -tuple  $\mathbf{x}$ ,  $x_i$  denotes the  $i$ th entry (or component) of  $\mathbf{x}$  for  $i = 1, \dots, n$ .

Matrices are normally represented by an uppercase Roman letter in boldface; for example,  $\mathbf{A}$ . The  $j$ th column of a matrix  $\mathbf{A}$  is denoted by  $A_j$  and the  $(i, j)$ -entry (that is, the entry in row  $i$  and column  $j$ ) is denoted by  $a_{ij}$ .

Scalars are usually represented by lowercase Greek letters; for example,  $\lambda, \alpha, \beta$  etc.

An  $n$ -tuple consisting of all zeros is denoted by  $0$ . The dimension of the tuple is inferred from the context.

For a matrix  $\mathbf{A}$ ,  $\mathbf{A}^\top$  denotes the transpose of  $\mathbf{A}$ . For an  $n$ -tuple  $\mathbf{x}$ ,  $\mathbf{x}^\top$  denotes the transpose of  $\mathbf{x}$ .

If  $\mathbf{A}$  and  $\mathbf{B}$  are  $m \times n$  matrices,  $\mathbf{A} \geq \mathbf{B}$  means  $a_{ij} \geq b_{ij}$  for all  $i = 1, \dots, m$ ,  $j = 1, \dots, n$ . Similar definitions hold for  $\mathbf{A} \leq \mathbf{B}$ ,  $\mathbf{A} = \mathbf{B}$ ,  $\mathbf{A} < \mathbf{B}$  and  $\mathbf{A} > \mathbf{B}$ . In particular, if  $\mathbf{u}$  and  $\mathbf{v}$  are  $n$ -tuples,  $\mathbf{u} \geq \mathbf{v}$  means  $u_i \geq v_i$  for  $i = 1, \dots, n$  and  $\mathbf{u} > \mathbf{0}$  means  $u_i > 0$  for  $i = 1, \dots, n$ .

Superscripts in brackets are used for indexing tuples. For example, we can write  $\mathbf{u}^{(1)}, \mathbf{u}^{(2)} \in \mathbb{R}^3$ . Then  $\mathbf{u}^{(1)}$  and  $\mathbf{u}^{(2)}$  are elements of  $\mathbb{R}^3$ . The second entry of  $\mathbf{u}^{(1)}$  is denoted by  $u_2^{(1)}$ .

## 9.1 Graphical example

To motivate the subject of linear programming (LP), we begin with a planning problem that can be solved graphically.

### Example 9.1: Lemonade Vendor

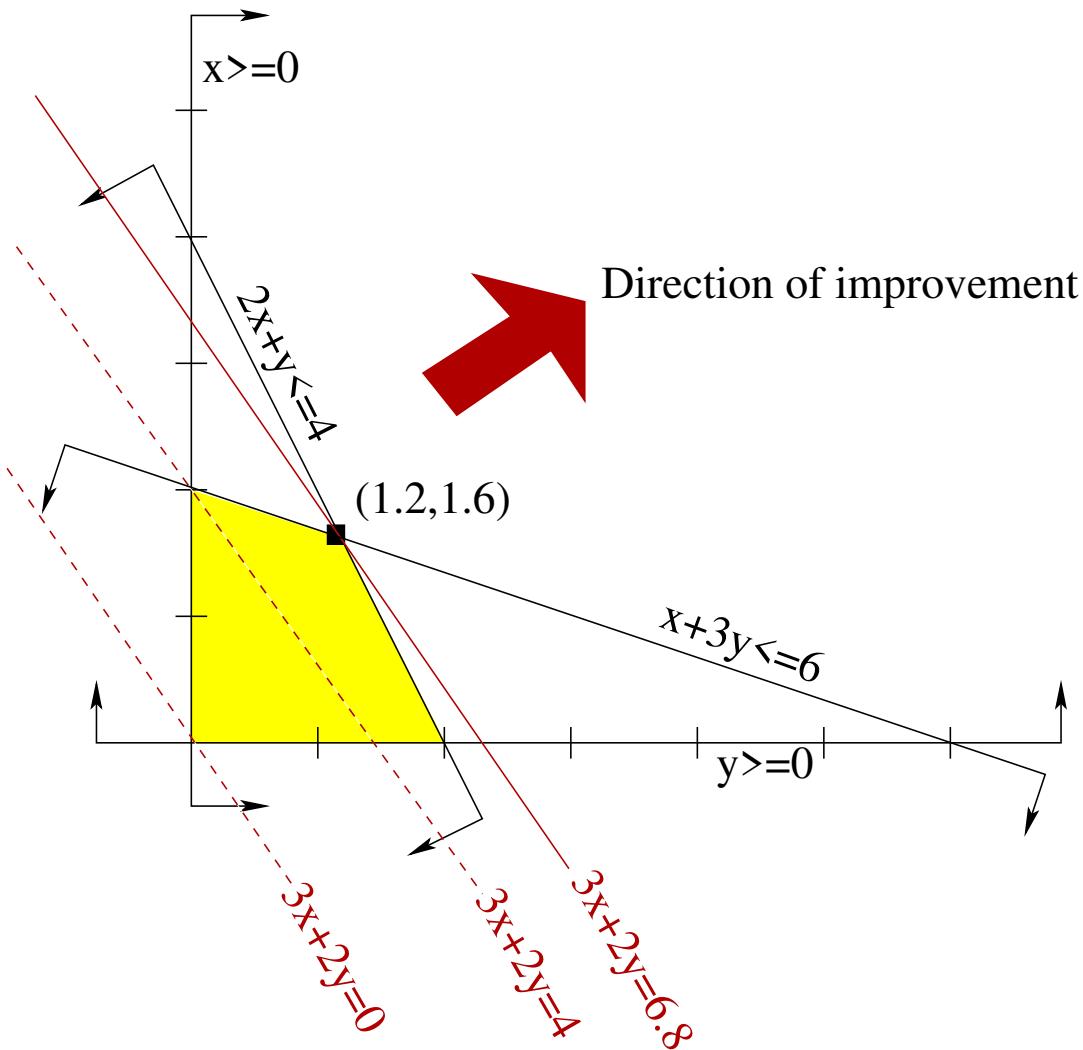
Say you are a vendor of lemonade and lemon juice. Each unit of lemonade requires 1 lemon and 2 litres of water. Each unit of lemon juice requires 3 lemons and 1 litre of water. Each unit of lemonade gives a profit of three dollars. Each unit of lemon juice gives a profit of two dollars. You have 6 lemons and 4 litres of water available. How many units of lemonade and lemon juice should you make to maximize profit?

If we let  $x$  denote the number of units of lemonade to be made and let  $y$  denote the number of units of lemon juice to be made, then the profit is given by  $3x + 2y$  dollars. We call  $3x + 2y$  the objective function. Note that there are a number of constraints that  $x$  and  $y$  must satisfy. First of all,  $x$  and  $y$  should be nonnegative. The number of lemons needed to make  $x$  units of lemonade and  $y$  units of lemon juice is  $x + 3y$  and cannot exceed 6. The number of litres of water needed to make  $x$  units of lemonade and  $y$  units of lemon juice is  $2x + y$  and cannot exceed 4. Hence, to determine the maximum profit, we need to maximize  $3x + 2y$  subject to  $x$  and  $y$  satisfying the constraints  $x + 3y \leq 6$ ,  $2x + y \leq 4$ ,  $x \geq 0$ , and  $y \geq 0$ .

A more compact way to write the problem is as follows:

$$\begin{aligned} & \text{maximize} && 3x + 2y \\ & \text{subject to} && x + 3y \leq 6 \\ & && 2x + y \leq 4 \\ & && x \geq 0 \\ & && y \geq 0. \end{aligned}$$

We can solve this maximization problem graphically as follows. We first sketch the set of  $\begin{bmatrix} x \\ y \end{bmatrix}$  satisfying the constraints, called the feasible region, on the  $(x, y)$ -plane. We then take the objective function  $3x + 2y$  and turn it into an equation of a line  $3x + 2y = z$  where  $z$  is a parameter. Note that as the value of  $z$  increases, the line defined by the equation  $3x + 2y = z$  moves in the direction of the normal vector  $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$ . We call this direction the direction of improvement. Determining the maximum value of the objective function, called the optimal value, subject to the constraints amounts to finding the maximum value of  $z$  so that the line defined by the equation  $3x + 2y = z$  still intersects the feasible region.



In the figure above, the lines with  $z$  at 0, 4 and 6.8 have been drawn. From the picture, we can see that if  $z$  is greater than 6.8, the line defined by  $3x + 2y = z$  will not intersect the feasible region. Hence, the profit cannot exceed 6.8 dollars.

As the line  $3x + 2y = 6.8$  does intersect the feasible region, 6.8 is the maximum value for the objective function. Note that there is only one point in the feasible region that intersects the line  $3x + 2y = 6.8$ , namely  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.6 \end{bmatrix}$ . In other words, to maximize profit, we want to make 1.2 units of lemonade and 1.6 units of lemon juice.

The above solution method can hardly be regarded as rigorous because we relied on a picture to conclude that  $3x + 2y \leq 6.8$  for all  $\begin{bmatrix} x \\ y \end{bmatrix}$  satisfying the constraints. But we can actually show this *algebraically*.

Note that multiplying both sides of the constraint  $x + 3y \leq 6$  gives  $0.2x + 0.6y \leq 1.2$ , and multiplying both sides of the constraint  $2x + y \leq 4$  gives  $2.8x + 1.4y \leq 5.6$ . Hence, any  $\begin{bmatrix} x \\ y \end{bmatrix}$  that satisfies both  $x + 3y \leq 6$  and  $2x + y \leq 4$  must also satisfy  $(0.2x + 0.6y) + (2.8x + 1.4y) \leq 1.2 + 5.6$ , which simplifies to  $3x + 2y \leq 6.8$  as desired! (Here, we used the fact that if  $a \leq b$  and  $c \leq d$ , then  $a + c \leq b + d$ .)

Now, one might ask if it is always possible to find an algebraic proof like the one above for similar

problems. If the answer is yes, how does one find such a proof? We will see answers to this question later on.

Before we end this segment, let us consider the following problem:

$$\begin{array}{lll} \text{minimize} & -2x + y \\ \text{subject to} & -x + y \leq 3 \\ & x - 2y \leq 2 \\ & x \geq 0 \\ & y \geq 0. \end{array}$$

Note that for any  $t \geq 0$ ,  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t \\ t \end{bmatrix}$  satisfies all the constraints. The value of the objective function at  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t \\ t \end{bmatrix}$  is  $-t$ . As  $t \rightarrow \infty$ , the value of the objective function tends to  $-\infty$ . Therefore, there is no minimum value for the objective function. The problem is said to be unbounded. Later on, we will see how to detect unboundedness algorithmically.

As an exercise, check that unboundedness can also be established by using  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2t+2 \\ t \end{bmatrix}$  for  $t \geq 0$ .

## Exercises

---

1. Sketch all  $\begin{bmatrix} x \\ y \end{bmatrix}$  satisfying

$$x - 2y \leq 2$$

on the  $(x, y)$ -plane.

2. Determine the optimal value of

$$\begin{array}{ll} \text{Minimize} & x + y \\ \text{Subject to} & 2x + y \geq 4 \\ & x + 3y \geq 1. \end{array}$$

3. Show that the problem

$$\begin{array}{ll} \text{Minimize} & -x + y \\ \text{Subject to} & 2x - y \geq 0 \\ & x + 3y \geq 3 \end{array}$$

is unbounded.

4. Suppose that you are shopping for dietary supplements to satisfy your required daily intake of 0.40mg of nutrient  $M$  and 0.30mg of nutrient  $N$ . There are three popular products on the market. The costs and the amounts of the two nutrients are given in the following table:

	Product 1	Product 2	Product 3
Cost	\$27	\$31	\$24
Daily amount of $M$	0.16 mg	0.21 mg	0.11 mg

	Product 1	Product 2	Product 3
Daily amount of $N$	0.19 mg	0.13 mg	0.15 mg

You want to determine how much of each product you should buy so that the daily intake requirements of the two nutrients are satisfied at minimum cost. Formulate your problem as a linear programming problem, assuming that you can buy a fractional number of each product.

## Solutions

---

1. The points  $(x,y)$  satisfying  $x - 2y \leq 2$  are precisely those above the line passing through  $(2,0)$  and  $(0,-1)$ .
2. We want to determine the minimum value  $z$  so that  $x + y = z$  defines a line that has a nonempty intersection with the feasible region. However, we can avoid referring to a sketch by setting  $x = z - y$  and substituting for  $x$  in the inequalities to obtain:

$$\begin{aligned} 2(z-y) + y &\geq 4 \\ (z-y) + 3y &\geq 1, \end{aligned}$$

or equivalently,

$$\begin{aligned} z &\geq 2 + \frac{1}{2}y \\ z &\geq 1 - 2y, \end{aligned}$$

Thus, the minimum value for  $z$  is  $\min\{2 + \frac{1}{2}y, 1 - 2y\}$ , which occurs at  $y = -\frac{2}{5}$ . Hence, the optimal value is  $\frac{9}{5}$ .

We can verify our work by doing the following. If our calculations above are correct, then an optimal solution is given by  $x = \frac{11}{5}$ ,  $y = -\frac{2}{5}$  since  $x = z - y$ . It is easy to check that this satisfies both inequalities and therefore is a feasible solution.

Now, taking  $\frac{2}{5}$  times the first inequality and  $\frac{1}{5}$  times the second inequality, we can infer the inequality  $x + y \geq \frac{9}{5}$ . The left-hand side of this inequality is precisely the objective function. Hence, no feasible solution can have objective function value less than  $\frac{9}{5}$ . But  $x = \frac{11}{5}$ ,  $y = -\frac{2}{5}$  is a feasible solution with objective function value equal to  $\frac{9}{5}$ . As a result, it must be an optimal solution.

**Remark.** We have not yet discussed how to obtain the multipliers  $\frac{2}{5}$  and  $\frac{1}{5}$  for inferring the inequality  $x + y \geq \frac{9}{5}$ . This is an issue that will be taken up later. In the meantime, think about how one could have obtained these multipliers for this particular exercise.

3. We could glean some insight by first making a sketch on the  $(x,y)$ -plane.

The line defined by  $-x + y = z$  has  $x$ -intercept  $-z$ . Note that for  $z \leq -3$ ,  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -z \\ 0 \end{bmatrix}$  satisfies both

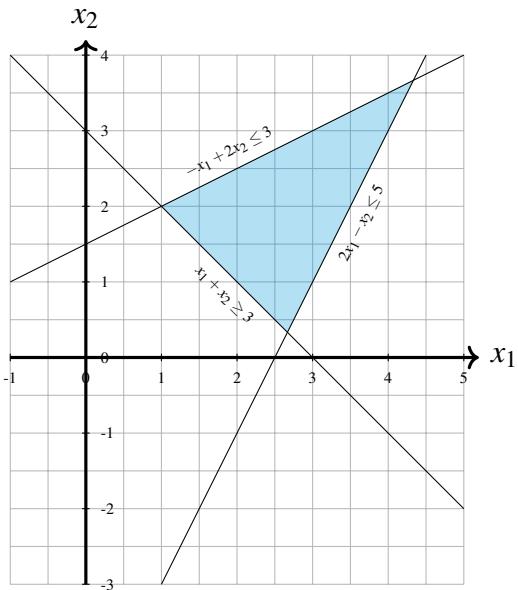
inequalities and the value of the objective function at  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -z \\ 0 \end{bmatrix}$  is  $z$ . Hence, there is no lower bound on the value of objective function.

4. Let  $x_i$  denote the amount of Product  $i$  to buy for  $i = 1, 2, 3$ . Then, the problem can be formulated as

$$\begin{aligned} & \text{minimize} && 27x_1 + 31x_2 + 24x_3 \\ & \text{subject to} && 0.16x_1 + 0.21x_2 + 0.11x_3 \geq 0.30 \\ & && 0.19x_1 + 0.13x_2 + 0.15x_3 \geq 0.40 \\ & && x_1, x_2, x_3 \geq 0. \end{aligned}$$

**Remark.** If one cannot buy fractional amounts of the products, the problem can be formulated as

$$\begin{aligned} & \text{minimize} && 27x_1 + 31x_2 + 24x_3 \\ & \text{subject to} && 0.16x_1 + 0.21x_2 + 0.11x_3 \geq 0.30 \\ & && 0.19x_1 + 0.13x_2 + 0.15x_3 \geq 0.40 \\ & && x_1, x_2, x_3 \geq 0. \\ & && x_1, x_2, x_3 \in \mathbb{Z}. \end{aligned}$$



1

## 9.2 Definitions

---

The following is an example of a problem in **linear programming**:

$$\begin{aligned} & \text{Maximize} && x + y - 2z \\ & \text{Subject to} && 2x + y + z \leq 4 \\ & && 3x - y + z = 0 \\ & && x, y, z \geq 0 \end{aligned}$$

<sup>1</sup><https://tex.stackexchange.com/questions/75933/how-to-draw-the-region-of-inequality>

**Solving** this problem means finding real values for the **variables**  $x, y, z$  satisfying the **constraints**  $2x + y + z \leq 4$ ,  $3x - y + z = 0$ , and  $x, y, z \geq 0$  that gives the maximum possible value (if it exists) for the **objective function**  $x + y - 2z$ .

For example,  $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$  satisfies all the constraints and is called a **feasible solution**. Its **objective**

**function value**, obtained by evaluating the objective function at  $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$ , is  $0 + 1 - 2(1) = -1$ . The set of feasible solutions to a linear programming problem is called the **feasible region**.

More formally, a linear programming problem is an optimization problem of the following form:

$$\begin{aligned} & \text{Maximize (or Minimize)} \quad \sum_{j=1}^n c_j x_j \\ & \text{Subject to} \quad P_i(x_1, \dots, x_n) \quad i = 1, \dots, m \end{aligned}$$

where  $m$  and  $n$  are positive integers,  $c_j \in \mathbb{R}$  for  $j = 1, \dots, n$ , and for each  $i = 1, \dots, m$ ,  $P_i(x_1, \dots, x_n)$  is a **linear constraint** on the (**decision**) **variables**  $x_1, \dots, x_n$  having one of the following forms:

- $a_1 x_1 + \dots + a_n x_n \geq \beta$
- $a_1 x_1 + \dots + a_n x_n \leq \beta$
- $a_1 x_1 + \dots + a_n x_n = \beta$

where  $\beta, a_1, \dots, a_n \in \mathbb{R}$ . To save writing, the word “Minimize” (“Maximize”) is replaced with “min” (“max”) and “Subject to” is abbreviated as “s.t.”.

A feasible solution  $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$  that gives the maximum possible objective function value in the case of a maximization problem is called an **optimal solution** and its objective function value is the **optimal value** of the problem.

The following example shows that it is possible to have multiple optimal solutions:

$$\begin{aligned} & \max \quad x + y \\ & \text{s.t.} \quad 2x + 2y \leq 1 \end{aligned}$$

The constraint says that  $x + y$  cannot exceed  $\frac{1}{2}$ . Now, both  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix}$  and  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{2} \end{bmatrix}$  are feasible solutions having objective function value  $\frac{1}{2}$ . Hence, they are both optimal solutions. (In fact, this problem has infinitely many optimal solutions. Can you specify all of them?)

Not all linear programming problems have optimal solutions. For example, a problem can have no feasible solution. Such a problem is said to be **infeasible**. Here is an example of an infeasible problem:

$$\begin{aligned} & \min \quad x \\ & \text{s.t.} \quad x \leq 1 \\ & \quad x \geq 2 \end{aligned}$$

There is no value for  $x$  that is at the same time at most 1 and at least 2.

Even if a problem is not infeasible, it might not have an optimal solution as the following example shows:

$$\begin{aligned} \min \quad & x \\ \text{s.t.} \quad & x \leq 0 \end{aligned}$$

Note that now matter what real number  $M$  we are given, we can always find a feasible solution whose objective function value is less than  $M$ . Such a problem is said to be **unbounded**. (For a maximization problem, it is unbounded if one can find feasible solutions who objective function value is larger than any given real number.)

---

So far, we have seen that a linear programming problem can have an optimal solution, be infeasible, or be unbounded. Is it possible for a linear programming problem to be not infeasible, not unbounded, and with no optimal solution?

The following optimization problem, though not a linear programming problem, is not infeasible, not unbounded, and has no optimal solution:

$$\begin{aligned} \min \quad & 2^x \\ \text{s.t.} \quad & x \leq 0 \end{aligned}$$

The objective function value is never negative and can get arbitrarily close to 0 but can never attain 0.

A main result in linear programming states that if a linear programming problem is not infeasible and is not unbounded, then it must have an optimal solution. This result is known as the **Fundamental Theorem of Linear Programming** (Theorem 9.4) and we will see a proof of this important result. In the meantime, we will consider the seemingly easier problem of determining if a system of linear constraints has a solution.

## Exercises

---

- Determine all values of  $a$  such that the problem

$$\begin{aligned} \min \quad & x + y \\ \text{s.t.} \quad & -3x + y \geq a \\ & 2x - y \geq 0 \\ & x + 2y \geq 2 \end{aligned}$$

is infeasible.

- Show that the problem

$$\begin{aligned} \min \quad & 2^x \cdot 4^y \\ \text{s.t.} \quad & e^{-3x+y} \geq 1 \\ & |2x - y| \leq 4 \end{aligned}$$

can be solved by solving a linear programming problem.

## Solutions

---

1. Adding the first two inequalities gives  $-x \geq a$ . Adding 2 times the second inequality and the third inequality gives  $5x \geq 2$ , implying that  $x \geq \frac{2}{5}$ . Hence, if  $a > -\frac{2}{5}$ , there is no solution.

Note that if  $a \leq -\frac{2}{5}$ , then  $(x, y) = (\frac{2}{5}, \frac{4}{5})$  satisfies all the inequalities. Hence, the problem is infeasible if and only if  $a > -\frac{2}{5}$ .

2. Note that the constraint  $|2x - y| \leq 4$  is equivalent to the constraints  $2x - y \leq 4$  and  $2x - y \geq -4$  taken together, and the constraint  $e^{-3x+y} \geq 1$  is equivalent to  $-3x + y \geq 0$ . Hence, we can rewrite the problem with linear constraints.

Finally, minimizing  $2^x \cdot 4^y$  is the same as minimizing  $2^{x+2y}$ , which is equivalent to minimizing  $x + 2y$ .

## 9.3 Farkas' Lemma

---

A well-known result in linear algebra states that a system of linear equations  $\mathbf{Ax} = \mathbf{b}$ , where  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,

$\mathbf{b} \in \mathbb{R}^m$ , and  $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$  is a tuple of variables, has no solution if and only if there exists  $\mathbf{y} \in \mathbb{R}^m$  such that  $\mathbf{y}^\top \mathbf{A} = 0$  and  $\mathbf{y}^\top \mathbf{b} \neq 0$ .

It is easily seen that if such a  $\mathbf{y}$  exists, then the system  $\mathbf{Ax} = \mathbf{b}$  cannot have a solution. (Simply multiply both sides of  $\mathbf{Ax} = \mathbf{b}$  on the left by  $\mathbf{y}^\top$ .) However, proving the converse requires a bit of work. A standard elementary proof involves using Gauss-Jordan elimination to reduce the original system to an equivalent system  $\mathbf{Qx} = \mathbf{d}$  such that  $\mathbf{Q}$  has a row of zero, say in row  $i$ , with  $\mathbf{d}_i \neq 0$ . The process can be captured by a square matrix  $\mathbf{M}$  satisfying  $\mathbf{MA} = \mathbf{Q}$ . We can then take  $\mathbf{y}^\top$  to be the  $i$ th row of  $\mathbf{M}$ .

An analogous result holds for systems of linear inequalities. The following result is one of the many variants of a result known as the **Farkas' Lemma**:

### Theorem 9.2: Farkas' Lemma

With  $\mathbf{A}$ ,  $\mathbf{x}$ , and  $\mathbf{b}$  as above, the system  $\mathbf{Ax} \geq \mathbf{b}$  has no solution if and only if there exists  $\mathbf{y} \in \mathbb{R}^m$  such that

$$\mathbf{y} \geq 0, \mathbf{y}^\top \mathbf{A} = 0, \mathbf{y}^\top \mathbf{b} > 0.$$

In other words, the system  $\mathbf{Ax} \geq \mathbf{b}$  has no solution if and only if one can infer the inequality  $0 \geq \gamma$  for some  $\gamma > 0$  by taking a nonnegative linear combination of the inequalities.

This result essentially says that there is always a certificate (the  $m$ -tuple  $\mathbf{y}$  with the prescribed properties) for the infeasibility of the system  $\mathbf{Ax} \geq \mathbf{b}$ . This allows third parties to verify the claim of infeasibility without having to solve the system from scratch.

**Example 9.3**

For the system

$$\begin{aligned} 2x - y + z &\geq 2 \\ -x + y - z &\geq 0 \\ -y + z &\geq 0, \end{aligned}$$

adding two times the second inequality and the third inequality to the first inequality gives  $0 \geq 2$ .

Hence,  $\mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$  is a certificate of infeasibility for this example.

We now give a proof of Theorem 9.3. It is easy to see that if such a  $\mathbf{y}$  exists, then the system  $\mathbf{Ax} \geq \mathbf{b}$  has no solution.

## 9.4 Fundamental Theorem of Linear Programming

Having used Fourier-Motzkin elimination to solve a linear programming problem, we now will go one step further and use the same technique to prove the following important result.

### Theorem 9.4: Fundamental Theorem of Linear Programming

For any given linear programming problem, exactly one of the following holds:

1. the problem is infeasible;
2. the problem is unbounded;
3. the problem has an optimal solution.

*Proof.* Without loss of generality, we may assume that the linear programming problem is of the form

$$\begin{array}{ll} \min & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} & \mathbf{Ax} \geq \mathbf{b} \end{array} \quad (9.1)$$

where  $m$  and  $n$  are positive integers,  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,  $\mathbf{c} \in \mathbb{R}^n$ , and  $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$  is a tuple of variables.

Indeed, any linear programming problem can be converted to a linear programming problem in the form of (9.1) having the same feasible region and optimal solution set. To see this, note that a constraint of the form  $\mathbf{a}^T \mathbf{x} \leq \beta$  can be written as  $-\mathbf{a}^T \mathbf{x} \geq -\beta$ ; a constraint of the form  $\mathbf{a}^T \mathbf{x} = \beta$  written as a pair of constraints  $\mathbf{a}^T \mathbf{x} \geq \beta$  and  $-\mathbf{a}^T \mathbf{x} \geq -\beta$ ; and a maximization problem is equivalent to the problem that minimizes the negative of the objective function subject to the same constraints.

Suppose that (9.1) is not infeasible. Form the system

$$\begin{aligned} z - \mathbf{c}^T \mathbf{x} &\geq 0 \\ -z + \mathbf{c}^T \mathbf{x} &\geq 0 \\ \mathbf{A}\mathbf{x} &\geq \mathbf{b}. \end{aligned} \tag{9.2}$$

Solving (9.1) is equivalent to finding among all the solutions to (9.2) one that minimizes  $z$ , if it exists. Eliminating the variables  $x_1, \dots, x_n$  (in any order) using Fourier-Motzkin elimination gives a system of linear inequalities (S) containing at most the variable  $z$ . By scaling, we may assume that the each coefficient of  $z$  in (S) is 1,  $-1$ , or 0. Note that any  $z$  satisfying (S) can be extended to a solution to (9.2) and the  $z$  value from any solution to (9.2) must satisfy (S).

That (9.1) is not unbounded implies that (S) must contain an inequality of the form  $z \geq \beta$  for some  $\beta \in \mathbb{R}$ . (Why?) Let all the inequalities in which the coefficient of  $z$  is positive be

$$z \geq \beta_i$$

where  $\beta_i \in \mathbb{R}$  for  $i = 1, \dots, p$  for some positive integer  $p$ . Let  $\gamma = \max\{\beta_1, \dots, \beta_p\}$ . Then for any solution  $x, z$  to (9.2),  $z$  is at least  $\gamma$ . But we can set  $z = \gamma$  and extend it to a solution to (9.2). Hence, we obtain an optimal solution for (9.1) and  $\gamma$  is the optimal value. This completes the proof of the theorem.

□

**Remark.** We can construct multipliers to infer the inequality  $\mathbf{c}^T \mathbf{x} \geq \gamma$  from the system  $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ . Because we obtained the inequality  $z \geq \gamma$  using Fourier-Motzkin elimination, there must exist real numbers  $\alpha, \beta, y_1^*, \dots, y_m^* \geq 0$  such that

$$[\alpha \ \beta \ y_1^* \ \dots \ y_m^*] \begin{bmatrix} 1 & -\mathbf{c}^T \\ -1 & \mathbf{c}^T \\ 0 & \mathbf{A} \end{bmatrix} \begin{bmatrix} z \\ \mathbf{x} \end{bmatrix} \geq [\alpha \ \beta \ y_1^* \ \dots \ y_m^*] \begin{bmatrix} 0 \\ 0 \\ \mathbf{b} \end{bmatrix}$$

is identically  $z \geq \gamma$ . Note that we must have  $\alpha - \beta = 1$  and

$$\mathbf{y}^* \geq 0, \mathbf{y}^{*\top} \mathbf{A} = \mathbf{c}^T, \text{ and } \mathbf{y}^{*\top} \mathbf{b} = \gamma$$

where  $\mathbf{y}^* = [y_1^*, \dots, y_m^*]^T$ . Hence,  $y_1^*, \dots, y_m^*$  are the desired multipliers.,

The significance of the fact that we can infer  $\mathbf{c}^T \mathbf{x} \geq \gamma$  where  $\gamma$  will be discussed in more details when we look at duality theory for linear programming.

## Exercises

---

- Determine the optimal value of the following linear programming problem:

$$\begin{aligned} \min \quad & x \\ \text{s.t.} \quad & x + y \geq 2 \\ & x - 2y + z \geq 0 \\ & y - 2z \geq -1. \end{aligned}$$

2. Determine if the following linear programming problem has an optimal solution:

$$\begin{aligned} \min \quad & x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 + 3x_2 \geq 4 \\ & -x_1 + x_2 \geq 0. \end{aligned}$$

3. A set  $S \subset \mathbb{R}^n$  is said to be bounded if there exists a real number  $M > 0$  such that for every  $\mathbf{x} \in S$ ,  $|x_i| < M$  for all  $i = 1, \dots, n$ . Prove that every linear programming problem with a bounded nonempty feasible region has an optimal solution.

## Solutions

---

1. The problem is equivalent to determining the minimum value for  $x$  among all  $x, y, z$  satisfying

$$\begin{aligned} x + y &\geq 2 & (1) \\ x - 2y + z &\geq 0 & (2) \\ y - 2z &\geq -1. & (3) \end{aligned}$$

We use Fourier-Motzkin Elimination Method to eliminate  $z$ . Multiplying (3) by  $\frac{1}{2}$ , we get

$$\begin{aligned} x + y &\geq 2 & (1) \\ x - 2y + z &\geq 0 & (2) \\ \frac{1}{2}y - z &\geq -\frac{1}{2}. & (4) \end{aligned}$$

Eliminating  $z$ , we obtain

$$\begin{aligned} x + y &\geq 2 & (1) \\ x - \frac{3}{2}y &\geq -\frac{1}{2} & (5) \end{aligned}$$

where (5) is given by (2) + (4).

Multiplying (5) by  $\frac{2}{3}$ , we get

$$\begin{aligned} x + y &\geq 2 & (1) \\ \frac{2}{3}x - y &\geq -\frac{1}{3} & (6) \end{aligned}$$

Eliminating  $y$ , we get

$$\frac{5}{3}x \geq \frac{5}{3} \quad (7)$$

where (7) is given by (1) + (6). Multiplying (7) by  $\frac{3}{5}$ , we obtain  $x \geq 1$ . Hence, the minimum possible value for  $x$  is 1.

Note that setting  $x = 1$ , the system (1) and (6) forces  $y = 1$ . And (2) and (3) together force  $z = 1$ . One can check that  $(x, y, z) = (1, 1, 1)$  is a feasible solution.

**Remark.** Note that the inequality  $x \geq 1$  is given by

$$\begin{aligned}
\frac{3}{5}(7) &\Leftarrow \frac{3}{5}(1) + \frac{3}{5}(6) \\
&\Leftarrow \frac{3}{5}(1) + \frac{2}{5}(5) \\
&\Leftarrow \frac{3}{5}(1) + \frac{2}{5}(2) + \frac{2}{5}(4) \\
&\Leftarrow \frac{3}{5}(1) + \frac{2}{5}(2) + \frac{1}{5}(3)
\end{aligned}$$

2. It suffices to determine if there exists a minimum value for  $z$  among all the solutions to the system

$$\begin{aligned}
z - x_1 - 2x_2 &\geq 0 & (1) \\
-z + x_1 + 2x_2 &\geq 0 & (2) \\
x_1 + 3x_2 &\geq 4 & (3) \\
-x_1 + x_2 &\geq 0 & (4)
\end{aligned}$$

Using Fourier-Motzkin elimination to eliminate  $x_1$ , we obtain:

$$\begin{aligned}
(1) + (2) : \quad 0 &\geq 0 \\
(1) + (3) : \quad z + x_2 &\geq 4 & (5) \\
(2) + (4) : \quad -z + 3x_2 &\geq 0 & (6) \\
(3) + (4) : \quad 4x_2 &\geq 4 & (7)
\end{aligned}$$

Note that all the coefficients of  $x_2$  is nonnegative. Hence, eliminating  $x_2$  will result in a system with no constraints. Therefore, there is no lower bound on the value of  $z$ . In particular, if  $z = t$  for  $t \leq 0$ , then from (5)–(6), we need  $x_2 \geq 4 - t$ ,  $3x_2 \geq t$ , and  $x_2 \geq 1$ . Hence, we can set  $x_2 = 4 - t$  and  $x_1 = -8 + 3t$ . This gives a feasible solution for all  $t \leq 0$  with objective function value that approaches  $-\infty$  as  $t \rightarrow -\infty$ . Hence, the linear programming problem is unbounded.

3. Let  $(P)$  denote a linear programming problem with a bounded nonempty feasible region with objective function  $\mathbf{c}^T \mathbf{x}$ . By assumption,  $(P)$  is not infeasible. Note that  $(P)$  is not unbounded because  $|\mathbf{c}^T \mathbf{x}| \leq \sum_i |c_i| |x_i| \leq M \sum_i |c_i|$ . Thus, by Theorem 9.4,  $(P)$  has an optimal solution.

## 9.5 Linear programming duality

---

Consider the following problem:

$$\begin{aligned}
\min \quad & \mathbf{c}^T \mathbf{x} \\
\text{s.t.} \quad & \mathbf{A} \mathbf{x} \geq \mathbf{b}.
\end{aligned} \tag{9.1}$$

In the remark at the end of Chapter ??, we saw that if (12.1) has an optimal solution, then there exists  $\mathbf{y}^* \in \mathbb{R}^m$  such that  $\mathbf{y}^* \geq 0$ ,  $\mathbf{y}^{*\top} \mathbf{A} = \mathbf{c}^T$ , and  $\mathbf{y}^{*\top} \mathbf{b} = \gamma$  where  $\gamma$  denotes the optimal value of (12.1).

Take any  $\mathbf{y} \in \mathbb{R}^m$  satisfying  $\mathbf{y} \geq 0$  and  $\mathbf{y}^T \mathbf{A} = \mathbf{c}^T$ . Then we can infer from  $\mathbf{A} \mathbf{x} \geq \mathbf{b}$  the inequality  $\mathbf{y}^T \mathbf{A} \mathbf{x} \geq \mathbf{y}^T \mathbf{b}$ , or more simply,  $\mathbf{c}^T \mathbf{x} \geq \mathbf{y}^T \mathbf{b}$ . Thus, for any such  $\mathbf{y}$ ,  $\mathbf{y}^T \mathbf{b}$  gives a lower bound for the objective function value of any feasible solution to (12.1). Since  $\gamma$  is the optimal value of  $(P)$ , we must have  $\gamma \geq \mathbf{y}^T \mathbf{b}$ .

As  $\mathbf{y}^* \mathbf{b} = \gamma$ , we see that  $\gamma$  is the optimal value of

$$\begin{aligned} \max \quad & \mathbf{y}^T \mathbf{b} \\ \text{s.t.} \quad & \mathbf{y}^T \mathbf{A} = \mathbf{c}^T \\ & \mathbf{y} \geq 0. \end{aligned} \tag{9.2}$$

Note that (12.2) is a linear programming problem! We call it the **dual problem** of the **primal problem** (12.1). We say that the dual variable  $y_i$  is **associated** with the constraint  $\mathbf{a}^{(i)\top} \mathbf{x} \geq b_i$  where  $\mathbf{a}^{(i)\top}$  denotes the  $i$ th row of  $\mathbf{A}$ .

In other words, we define the dual problem of (12.1) to be the linear programming problem (12.2). In the discussion above, we saw that if the primal problem has an optimal solution, then so does the dual problem and the optimal values of the two problems are equal. Thus, we have the following result:

### Theorem 9.5: strong-duality-special

Suppose that (12.1) has an optimal solution. Then (12.2) also has an optimal solution and the optimal values of the two problems are equal.

At first glance, requiring all the constraints to be  $\geq$ -inequalities as in (12.1) before forming the dual problem seems a bit restrictive. We now see how the dual problem of a primal problem in general form can be defined. We first make two observations that motivate the definition.

#### Observation 1

Suppose that our primal problem contains a mixture of all types of linear constraints:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} \geq \mathbf{b} \\ & \mathbf{A}' \mathbf{x} \leq \mathbf{b}' \\ & \mathbf{A}'' \mathbf{x} = \mathbf{b}'' \end{aligned} \tag{9.3}$$

where  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,  $\mathbf{A}' \in \mathbb{R}^{m' \times n}$ ,  $\mathbf{b}' \in \mathbb{R}^{m'}$ ,  $\mathbf{A}'' \in \mathbb{R}^{m'' \times n}$ , and  $\mathbf{b}'' \in \mathbb{R}^{m''}$ .

We can of course convert this into an equivalent problem in the form of (12.1) and form its dual. However, if we take the point of view that the function of the dual is to infer from the constraints of (12.3) an inequality of the form  $\mathbf{c}^T \mathbf{x} \geq \gamma$  with  $\gamma$  as large as possible by taking an appropriate linear combination of the constraints, we are effectively looking for  $\mathbf{y} \in \mathbb{R}^m$ ,  $\mathbf{y} \geq 0$ ,  $\mathbf{y}' \in \mathbb{R}^{m'}$ ,  $\mathbf{y}' \leq 0$ , and  $\mathbf{y}'' \in \mathbb{R}^{m''}$ , such that

$$\mathbf{y}^T \mathbf{A} + \mathbf{y}'^T \mathbf{A}' + \mathbf{y}''^T \mathbf{A}'' = \mathbf{c}^T$$

with  $\mathbf{y}^T \mathbf{b} + \mathbf{y}'^T \mathbf{b}' + \mathbf{y}''^T \mathbf{b}''$  to be maximized.

(The reason why we need  $\mathbf{y}' \leq 0$  is because inferring a  $\geq$ -inequality from  $\mathbf{A}' \mathbf{x} \leq \mathbf{b}'$  requires nonpositive multipliers. There is no restriction on  $\mathbf{y}''$  because the constraints  $\mathbf{A}'' \mathbf{x} = \mathbf{b}''$  are equalities.)

This leads to the dual problem:

$$\begin{aligned}
\max \quad & \mathbf{y}^T \mathbf{b} + \mathbf{y}'^T \mathbf{b}' + \mathbf{y}''^T \mathbf{b}'' \\
\text{s.t.} \quad & \mathbf{y}^T \mathbf{A} + \mathbf{y}'^T \mathbf{A}' + \mathbf{y}''^T \mathbf{A}'' = \mathbf{c}^T \\
& \mathbf{y} \geq 0 \\
& \mathbf{y}' \leq 0.
\end{aligned} \tag{9.4}$$

In fact, we could have derived this dual by applying the definition of the dual problem to

$$\begin{aligned}
\min \quad & \mathbf{c}^T \mathbf{x} \\
\text{s.t.} \quad & \begin{bmatrix} \mathbf{A} \\ -\mathbf{A}' \\ \mathbf{A}'' \\ -\mathbf{A}'' \end{bmatrix} \mathbf{x} \geq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b}' \\ \mathbf{b}'' \\ -\mathbf{b}'' \end{bmatrix},
\end{aligned}$$

which is equivalent to (12.3). The details are left as an exercise.

### Observation 2

Consider the primal problem of the following form:

$$\begin{aligned}
\min \quad & \mathbf{c}^T \mathbf{x} \\
\text{s.t.} \quad & \mathbf{Ax} \geq \mathbf{b} \\
& x_i \geq 0 \quad i \in P \\
& x_i \leq 0 \quad i \in N
\end{aligned} \tag{9.5}$$

where  $P$  and  $N$  are disjoint subsets of  $\{1, \dots, n\}$ . In other words, constraints of the form  $x_i \geq 0$  or  $x_i \leq 0$  are separated out from the rest of the inequalities.

Forming the dual of (12.5) as defined under Observation 1, we obtain the dual problem

$$\begin{aligned}
\max \quad & \mathbf{y}^T \mathbf{b} \\
\text{s.t.} \quad & \mathbf{y}^T \mathbf{a}^{(i)} = c_i \quad i \in \{1, \dots, n\} \setminus (P \cup N) \\
& \mathbf{y}^T \mathbf{a}^{(i)} + p_i = c_i \quad i \in P \\
& \mathbf{y}^T \mathbf{a}^{(i)} + q_i = c_i \quad i \in N \\
& p_i \geq 0 \quad i \in P \\
& q_i \leq 0 \quad i \in N
\end{aligned} \tag{9.6}$$

where  $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$ . Note that this problem is equivalent to the following without the variables  $p_i$ ,  $i \in P$  and  $q_i$ ,  $i \in N$ :

$$\begin{aligned}
\max \quad & \mathbf{y}^T \mathbf{b} \\
\text{s.t.} \quad & \mathbf{y}^T \mathbf{a}^{(i)} = c_i \quad i \in \{1, \dots, n\} \setminus (P \cup N) \\
& \mathbf{y}^T \mathbf{a}^{(i)} \leq c_i \quad i \in P \\
& \mathbf{y}^T \mathbf{a}^{(i)} \geq c_i \quad i \in N,
\end{aligned} \tag{9.7}$$

which can be taken as the dual problem of (12.5) instead of (12.6). The advantage here is that it has fewer variables than (12.6).

Hence, the dual problem of

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A} \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

is simply

$$\begin{aligned} \max \quad & \mathbf{y}^T \mathbf{b} \\ \text{s.t.} \quad & \mathbf{y}^T \mathbf{A} \leq \mathbf{c}^T \\ & \mathbf{y} \geq 0. \end{aligned}$$

As we can see from above, there is no need to associate dual variables to constraints of the form  $x_i \geq 0$  or  $x_i \leq 0$  provided we have the appropriate types of constraints in the dual problem. Combining all the observations lead to the definition of the dual problem for a primal problem in general form as discussed next.

### 9.5.1. The dual problem

---

Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,  $\mathbf{c} \in \mathbb{R}^n$ . Let  $\mathbf{a}^{(i)T}$  denote the  $i$ th row of  $\mathbf{A}$ . Let  $\mathbf{A}_j$  denote the  $j$ th column of  $\mathbf{A}$ .

Let  $(P)$  denote the minimization problem with variables in the tuple  $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$  given as follows:

- The objective function to be minimized is  $\mathbf{c}^T \mathbf{x}$
- The constraints are

$$\mathbf{a}^{(i)T} \mathbf{x} \sqcup_i b_i$$

where  $\sqcup_i$  is  $\leq$ ,  $\geq$ , or  $=$  for  $i = 1, \dots, m$ .

- For each  $j \in \{1, \dots, n\}$ ,  $x_j$  is constrained to be nonnegative, nonpositive, or free (i.e. not constrained to be nonnegative or nonpositive.)

Then the **dual problem** is defined to be the maximization problem with variables in the tuple  $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$  given as follows:

- The objective function to be maximized is  $\mathbf{y}^T \mathbf{b}$
- For  $j = 1, \dots, n$ , the  $j$ th constraint is

$$\left\{ \begin{array}{ll} \mathbf{y}^T \mathbf{A}_j \leq c_j & \text{if } x_j \text{ is constrained to be nonnegative} \\ \mathbf{y}^T \mathbf{A}_j \geq c_j & \text{if } x_j \text{ is constrained to be nonpositive} \\ \mathbf{y}^T \mathbf{A}_j = c_j & \text{if } x_j \text{ is free.} \end{array} \right.$$

- For each  $i \in \{1, \dots, m\}$ ,  $y_i$  is constrained to be nonnegative if  $\sqcup_i$  is  $\geq$ ;  $y_i$  is constrained to be nonpositive if  $\sqcup_i$  is  $\leq$ ;  $y_i$  is free if  $\sqcup_i$  is  $=$ .

The following table can help remember the above.

Primal (min)	Dual (max)
$\geq$ constraint	$\geq 0$ variable
$\leq$ constraint	$\leq 0$ variable
$=$ constraint	free variable
$\geq 0$ variable	$\leq$ constraint
$\leq 0$ variable	$\geq$ constraint
free variable	$=$ constraint

Below is an example of a primal-dual pair of problems based on the above definition:

Consider the primal problem:

$$\begin{array}{lllll} \min & x_1 & - & 2x_2 & + & 3x_3 \\ \text{s.t.} & -x_1 & & & + & 4x_3 = 5 \\ & 2x_1 & + & 3x_2 & - & 5x_3 \geq 6 \\ & & & & 7x_2 & \leq 8 \\ & x_1 & & & & \geq 0 \\ & & x_2 & & & \text{free} \\ & & & x_3 & & \leq 0. \end{array}$$

Here,  $\mathbf{A} = \begin{bmatrix} -1 & 0 & 4 \\ 2 & 3 & -5 \\ 0 & 7 & 0 \end{bmatrix}$ ,  $\mathbf{b} = \begin{bmatrix} 5 \\ 6 \\ 8 \end{bmatrix}$ , and  $\mathbf{c} = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$ .

The primal problem has three constraints. So the dual problem has three variables. As the first constraint in the primal is an equation, the corresponding variable in the dual is free. As the second constraint in the primal is a  $\geq$ -inequality, the corresponding variable in the dual is nonnegative. As the third constraint in the primal is a  $\leq$ -inequality, the corresponding variable in the dual is nonpositive. Now, the primal problem has three variables. So the dual problem has three constraints. As the first variable in the primal is nonnegative, the corresponding constraint in the dual is a  $\leq$ -inequality. As the second variable in the primal is free, the corresponding constraint in the dual is an equation. As the third variable in the primal is nonpositive, the corresponding constraint in the dual is a  $\geq$ -inequality. Hence, the dual problem is:

$$\begin{array}{lllll} \max & 5y_1 & + & 6y_2 & + & 8y_3 \\ \text{s.t.} & -y_1 & + & 2y_2 & & \leq 1 \\ & & & 3y_2 & + & 7y_3 = -2 \\ & 4y_1 & - & 5y_2 & & \geq 3 \\ & y_1 & & & & \text{free} \\ & & y_2 & & & \geq 0 \\ & & & y_3 & & \leq 0. \end{array}$$

**Remarks.** Note that in some books, the primal problem is always a maximization problem. In that case, what is our primal problem is their dual problem and what is our dual problem is their primal problem.

One can now prove a more general version of Theorem 12.2 as stated below. The details are left as an exercise.

### Theorem 9.6: Duality Theorem for Linear Programming

*Let  $(P)$  and  $(D)$  denote a primal-dual pair of linear programming problems. If either  $(P)$  or  $(D)$  has an optimal solution, then so does the other. Furthermore, the optimal values of the two problems are equal.*

Theorem 12.2.1 is also known informally as **strong duality**.

## Exercises

---

1. Write down the dual problem of

$$\begin{array}{lll} \min & 4x_1 - 2x_2 \\ \text{s.t.} & x_1 + 2x_2 \geq 3 \\ & 3x_1 - 4x_2 = 0 \\ & x_2 \geq 0. \end{array}$$

2. Write down the dual problem of the following:

$$\begin{array}{lll} \min & 3x_2 + x_3 \\ \text{s.t.} & x_1 + x_2 + 2x_3 = 1 \\ & x_1 - 3x_3 \leq 0 \\ & x_1, x_2, x_3 \geq 0. \end{array}$$

3. Write down the dual problem of the following:

$$\begin{array}{lll} \min & x_1 - 9x_3 \\ \text{s.t.} & x_1 - 3x_2 + 2x_3 = 1 \\ & x_1 \leq 0 \\ & x_2 \text{ free} \\ & x_3 \geq 0. \end{array}$$

4. Determine all values  $c_1, c_2$  such that the linear programming problem

$$\begin{array}{lll} \min & c_1x_1 + c_2x_2 \\ \text{s.t.} & 2x_1 + x_2 \geq 2 \\ & x_1 + 3x_2 \geq 1. \end{array}$$

has an optimal solution. Justify your answer

## Solutions

---

1. The dual is

$$\begin{array}{ll} \max & 3y_1 \\ \text{s.t.} & y_1 + 3y_2 = 4 \\ & 2y_1 - 4y_2 \leq -2 \\ & y_1 \geq 0. \end{array}$$

2. The dual is

$$\begin{array}{ll} \max & y_1 \\ \text{s.t.} & y_1 + y_2 \leq 0 \\ & y_1 \leq 3 \\ & 2y_1 - 3y_2 \leq 1 \\ & y_1 \quad \text{free} \\ & y_2 \leq 0. \end{array}$$

3. The dual is

$$\begin{array}{ll} \max & y_1 \\ \text{s.t.} & y_1 \geq 1 \\ & -3y_1 = 0 \\ & 2y_1 \leq -9 \\ & y_1 \quad \text{free.} \end{array}$$

4. Let (P) denote the given linear programming problem.

Note that  $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  is a feasible solution to (P). Therefore, by Theorem ??, it suffices to find all values  $c_1, c_2$  such that

(P) is not unbounded. This amounts to finding all values  $c_1, c_2$  such that the dual problem of (P) has a feasible solution.

The dual problem of (P) is

$$\begin{array}{ll} \max & 2y_1 + y_2 \\ & 2y_1 + y_2 = c_1 \\ & y_1 + 3y_2 = c_2 \\ & y_1, y_2 \geq 0. \end{array}$$

The two equality constraints gives  $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{3}{5}c_1 - \frac{1}{5}c_2 \\ -\frac{1}{5}c_1 + \frac{2}{5}c_2 \end{bmatrix}$ . Thus, the dual problem is feasible if and only if  $c_1$  and  $c_2$  are real numbers satisfying

$$\begin{array}{ll} \frac{3}{5}c_1 - \frac{1}{5}c_2 \geq & 0 \\ -\frac{1}{5}c_1 + \frac{2}{5}c_2 \geq & 0, \end{array}$$

or more simply,

$$\frac{1}{3}c_2 \leq c_1 \leq 2c_2.$$

## 9.6 Complementary slackness

### Theorem 9.7: Weak Duality

Let  $(P)$  and  $(D)$  denote a primal-dual pair of linear programming problems in generic form as defined previously. Let  $\mathbf{x}^*$  be a feasible solution to  $(P)$  and  $\mathbf{y}^*$  is a feasible solution to  $(D)$ . Then the following hold:

1.  $\mathbf{c}^\top \mathbf{x}^* \geq \mathbf{y}^{*\top} \mathbf{b}$ .
2.  $\mathbf{x}^*$  and  $\mathbf{y}^*$  are optimal solutions to the respective problems if and only if the following conditions (known as the **complementary slackness conditions**) hold:

$$\begin{aligned} x_j^* = 0 &\quad \text{or} \quad \mathbf{y}^{*\top} \mathbf{A}_j = c_j \quad \text{for } j = 1, \dots, n \\ y_i^* = 0 &\quad \text{or} \quad \mathbf{a}^{(i)\top} \mathbf{x}^* = b_i \quad \text{for } i = 1, \dots, m \end{aligned}$$

Part 1 of the theorem is known as **weak duality**. Part 2 of the theorem is often called the **Complementary Slackness Theorem**.

#### Proof. [Proof of Theorem 9.6]

Note that if  $x_j^*$  is constrained to be nonnegative, its corresponding dual constraint is  $\mathbf{y}^{*\top} \mathbf{A}_j \leq c_j$ . Hence,  $(c_j - \mathbf{y}^{*\top} \mathbf{A}_j)x_j^* \geq 0$  with equality if and only if  $x_j^* = 0$  or  $\mathbf{y}^{*\top} \mathbf{A}_j = c_j$  (or both).

If  $x_j^*$  is constrained to be nonpositive, its corresponding dual constraint is  $\mathbf{y}^{*\top} \mathbf{A}_j \geq c_j$ . Hence,  $(c_j - \mathbf{y}^{*\top} \mathbf{A}_j)x_j^* \geq 0$  with equality if and only if  $x_j^* = 0$  or  $\mathbf{y}^{*\top} \mathbf{A}_j = c_j$  (or both).

If  $x_j^*$  is free, its corresponding dual constraint is  $\mathbf{y}^{*\top} \mathbf{A}_j = c_j$ . Hence,  $(c_j - \mathbf{y}^{*\top} \mathbf{A}_j)x_j^* = 0$ .

We can combine these three cases and obtain that  $(\mathbf{c}^\top - \mathbf{y}^{*\top} \mathbf{A})\mathbf{x}^* = \sum_{j=1}^n (c_j - \mathbf{y}^{*\top} \mathbf{A}_j)x_j^* \geq 0$  with equality if and only if for each  $j = 1, \dots, n$ ,

$$x_j^* = 0 \text{ or } \mathbf{y}^{*\top} \mathbf{A}_j = c_j.$$

(Here, the usage of “or” is not exclusive.)

Similarly,  $\mathbf{y}^{*\top}(\mathbf{A}\mathbf{x}^* - \mathbf{b}) = \sum_{i=1}^n y_i^*(\mathbf{a}^{(i)\top} \mathbf{x}^* - b_i) \geq 0$  with equality if and only if for each  $i = 1, \dots, n$ ,

$$y_i^* = 0 \text{ or } \mathbf{a}^{(i)\top} \mathbf{x}^* = b_i.$$

(Again, the usage of “or” is not exclusive.)

Adding the inequalities  $(\mathbf{c}^\top - \mathbf{y}^{*\top} \mathbf{A})\mathbf{x}^* \geq 0$  and  $\mathbf{y}^{*\top}(\mathbf{A}\mathbf{x}^* - \mathbf{b}) \geq 0$ , we obtain  $\mathbf{c}^\top \mathbf{x}^* - \mathbf{y}^{*\top} \mathbf{b} \geq 0$  with equality if and only if the complementary slackness conditions hold. By strong duality,  $\mathbf{x}^*$  is optimal  $(P)$  and  $\mathbf{y}^*$  is optimal for  $(D)$  if and only if  $\mathbf{c}^\top \mathbf{x}^* = \mathbf{y}^{*\top} \mathbf{b}$ . The result now follows.



The complementary slackness conditions give a characterization of optimality which can be useful in solving certain problems as illustrated by the following example.

**Example 9.8: Checking Optimality**

Let  $(P)$  denote the following linear programming problem:

$$\begin{array}{lll} \min & 2x_1 + 4x_2 + 2x_3 \\ \text{s.t.} & x_1 + x_2 + 3x_3 \leq 1 \\ & -x_1 + 2x_2 + x_3 \geq 1 \\ & 3x_2 - 6x_3 = 0 \\ & x_1, x_3 \geq 0 \\ & x_2 \quad \text{free.} \end{array}$$

Is  $\mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{2}{5} \\ \frac{1}{5} \end{bmatrix}$  an optimal solution to  $(P)$ ?

**Solution.** One could answer this question by solving  $(P)$  and then see if the objective function value of  $\mathbf{x}^*$ , assuming that its feasibility has already been verified, is equal to the optimal value. However, there is a way to make use of the given information to save some work.

Let  $(D)$  denote the dual problem of  $(P)$ :

$$\begin{array}{lll} \max & y_1 + y_2 \\ \text{s.t.} & y_1 - y_2 \leq 2 \\ & y_1 + 2y_2 + 3y_3 = 4 \\ & 3y_1 + y_2 - 6y_3 \leq 2 \\ & y_1 \leq 0 \\ & y_2 \geq 0 \\ & y_3 \quad \text{free.} \end{array}$$

One can check that  $\mathbf{x}^*$  is a feasible solution to  $(P)$ . If  $\mathbf{x}^*$  is optimal, then there must exist a feasible solution  $\mathbf{y}^*$  to  $(D)$  satisfying together with  $\mathbf{x}^*$  the complementary slackness conditions:

$$\begin{array}{ll} y_1^* = 0 & \text{or} \quad x_1^* + x_2^* + 3x_3^* = 1 \\ y_2^* = 0 & \text{or} \quad -x_1^* + 2x_2^* + x_3^* = 1 \\ y_3^* = 0 & \text{or} \quad 3x_2^* - 6x_3^* = 0 \\ x_1^* = 0 & \text{or} \quad y_1^* - y_2^* = 2 \\ x_2^* = 0 & \text{or} \quad y_1^* + 2y_2^* + 3y_3^* = 4 \\ x_3^* = 0 & \text{or} \quad 3y_1^* + y_2^* - 6y_3^* = 2. \end{array}$$

As  $x_2^*, x_3^* > 0$ , satisfying the above conditions require that

$$\begin{aligned} y_1^* + 2y_2^* + 3y_3^* &= 4 \\ 3y_1^* + y_2^* - 6y_3^* &= 2. \end{aligned}$$

Solving for  $y_2^*$  and  $y_3^*$  in terms of  $y_1^*$  gives  $y_2^* = 2 - y_1^*$ ,  $y_3^* = \frac{1}{3}y_1^*$ . To make  $\mathbf{y}^*$  feasible to  $(D)$ , we can set  $y_1^* = 0$  to obtain the feasible solution  $y_1^* = 0, y_2^* = 2, y_3^* = 0$ . We can check that this  $\mathbf{y}^*$  satisfies the complementary slackness conditions with  $\mathbf{x}^*$ . Hence,  $\mathbf{x}^*$  is an optimal solution to  $(P)$  by Theorem 9.6, part 2. ♠

## Exercises

---

1. Let (P) and (D) denote a primal-dual pair of linear programming problems. Prove that if (P) is not infeasible and (D) is infeasible, then (P) is unbounded.

2. Let (P) denote the following linear programming problem:

$$\begin{array}{ll} \min & 4x_2 + 2x_3 \\ \text{s.t.} & x_1 + x_2 + 3x_3 \leq 1 \\ & x_1 - 2x_2 + x_3 \geq 1 \\ & x_1 + 3x_2 - 6x_3 = 0 \\ & x_1, x_3 \geq 0 \\ & x_2 \quad \text{free.} \end{array}$$

Determine if  $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{3}{5} \\ -\frac{1}{5} \\ 0 \end{bmatrix}$  is an optimal solution to (P).

3. Let (P) denote the following linear programming problem:

$$\begin{array}{ll} \min & x_1 + 2x_2 - 3x_3 \\ \text{s.t.} & x_1 + 2x_2 + 2x_3 = 2 \\ & -x_1 + x_2 + x_3 = 1 \\ & -x_1 + x_2 - x_3 \geq 0 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

Determine if  $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$  is an optimal solution to (P).

4. Let  $m$  and  $n$  be positive integers. Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$ . Let  $\mathbf{b} \in \mathbb{R}^m$ . Let  $\mathbf{c} \in \mathbb{R}^n$ . Let (P) denote the linear programming problem

$$\begin{array}{ll} \min & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0. \end{array}$$

Let (D) denote the dual problem of (P):

$$\begin{array}{ll} \max & \mathbf{y}^\top \mathbf{b} \\ \text{s.t.} & \mathbf{y}^\top \mathbf{A} \leq \mathbf{c}^\top. \end{array}$$

Suppose that  $\mathbf{A}$  has rank  $m$  and that (P) has at least one optimal solution. Prove that if  $x_j^* = 0$  for every optimal solution  $\mathbf{x}^*$  to (P), then there exists an optimal solution  $\mathbf{y}^*$  to (D) such that  $\mathbf{y}^{*\top} \mathbf{A}_j < c_i$  where  $\mathbf{A}_j$  denotes the  $j$ th column of  $\mathbf{A}$ .

## Solutions

---

1. By the Fundamental Theorem of Linear Programming, (P) either is unbounded or has an optimal solution. If it is the latter, then by strong duality, (D) has an optimal solution, which contradicts that (D) is infeasible. Hence, (P) must be unbounded.
2. We show that it is not an optimal solution to (P). First, note that the dual problem of (P) is

$$\begin{array}{ll} \max & y_1 + y_2 \\ \text{s.t.} & y_1 + y_2 + y_3 \leq 0 \\ & y_1 - 2y_2 + 3y_3 = 4 \\ & 3y_1 + y_2 - 6y_3 \leq 2 \\ & y_1 \leq 0 \\ & y_2 \geq 0 \\ & y_3 \text{ free.} \end{array}$$

\end{bmatrix}) were an optimal solution, there would exist  $\mathbf{y}^*$  feasible to (D) satisfying the complementary slackness conditions with  $\mathbf{x}^*$ :

$$\begin{aligned} y_1^* = 0 & \quad \text{or} \quad x_1^* + x_2^* + 3x_3^* = 1 \\ y_2^* = 0 & \quad \text{or} \quad x_1^* - 2x_2^* + x_3^* = 1 \\ y_3^* = 0 & \quad \text{or} \quad x_1^* + 3x_2^* - 6x_3^* = 0 \\ x_1^* = 0 & \quad \text{or} \quad y_1^* + y_2^* + y_3^* = 0 \\ x_2^* = 0 & \quad \text{or} \quad y_1^* - 2y_2^* + 3y_3^* = 4 \\ x_3^* = 0 & \quad \text{or} \quad 3y_1^* + y_2^* - 6y_3^* = 2. \end{aligned}$$

Since  $x_1^* + x_2^* + 3x_3^* < 1$ , we must have  $y_1^* = 0$ . Also,  $x_1^*, x_2^*$  are both nonzero. Hence,

$$\begin{aligned} y_1^* + y_2^* + y_3^* &= 0 \\ y_1^* - 2y_2^* + 3y_3^* &= 4, \end{aligned}$$

implying that

$$\begin{aligned} y_2^* + y_3^* &= 0 \\ -2y_2^* + 3y_3^* &= 4. \end{aligned}$$

Solving gives  $y_2^* = -\frac{4}{5}$  and  $y_3^* = \frac{4}{5}$ . But this implies that  $\mathbf{y}^*$  is not a feasible solution to the dual problem since we need  $y_2^* \geq 0$ . Hence,  $\mathbf{x}^*$  is not an optimal solution to (P).

3. We show that it is not an optimal solution to (P). First, note that the dual problem of (P) is

$$\begin{aligned} \max \quad & 2y_1 + y_2 \\ \text{s.t.} \quad & y_1 - y_2 - y_3 \leq 1 \\ & 2y_1 + y_2 + y_3 \leq 2 \\ & 2y_1 + y_2 - y_3 \leq -3 \\ & y_1, y_2 \quad \text{free.} \\ & y_3 \geq 0 \end{aligned}$$

Note that  $\mathbf{x}^* = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$  is a feasible solution to (P). If it were an optimal solution to (P), there would exist  $\mathbf{y}^*$  feasible to the dual problem (D) satisfying the complementary slackness conditions with  $\mathbf{x}^*$ :

$$\begin{aligned} y_1^* = 0 \quad & \text{or} \quad x_1^* + 2x_2^* + 2x_3^* = 2 \\ y_2^* = 0 \quad & \text{or} \quad -x_1^* + x_2^* + x_3^* = 1 \\ y_3^* = 0 \quad & \text{or} \quad -x_1^* + x_2^* - x_3^* = 0 \\ x_1^* = 0 \quad & \text{or} \quad y_1^* - y_2^* - y_3^* = 1 \\ x_2^* = 0 \quad & \text{or} \quad 2y_1^* + y_2^* + y_3^* = 2 \\ x_3^* = 0 \quad & \text{or} \quad 2y_1^* + y_2^* - y_3^* = -3. \end{aligned}$$

Since  $-x_1^* + x_2^* - x_3^* > 0$ , we must have  $y_3^* = 0$ . Also,  $x_2^* > 0$  implies that  $2y_1^* + y_2^* + y_3^* = 2$ . Simplifying gives  $y_2^* = 2 - 2y_1^*$ .

Hence, for  $\mathbf{y}^*$  to be feasible to the dual problem, it needs to satisfy the third constraint,  $2y_1^* + (2 - 2y_1^*) \leq -3$ , which simplifies to the absurdity  $2 \leq -3$ . Hence,  $\mathbf{x}^*$  is not an optimal solution to (P).

4. Let  $v$  denote the optimal value of (P). Let (P') denote the problem

$$\begin{aligned} \min \quad & -x_i \\ \text{s.t.} \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{c}^\top \mathbf{x} \leq v \\ & \mathbf{x} \geq 0 \end{aligned}$$

Note that  $x^*$  is a feasible solution to (P') if and only if it is an optimal solution to (P). Since  $x_i^* = 0$  for every optimal solution to (P), we see that the optimal value of (P') is 0.

Let (D') denote the dual problem of (P'):

$$\begin{aligned} \max \quad & \mathbf{y}^\top \mathbf{b} + vu \\ \text{s.t.} \quad & \mathbf{y}^\top \mathbf{A}_p + c_p u \leq 0 \quad \text{for all } p \neq i \\ & \mathbf{y}^\top \mathbf{A}_i + c_i u \leq -1 \\ & u \leq 0. \end{aligned}$$

Suppose that an optimal solution to (D') is given by  $\mathbf{y}', u'$ . Let  $\bar{\mathbf{y}}$  be an optimal solution to (D). We consider two cases.

**Case 1:**  $u' = 0$ .

Then  $\mathbf{y}'^\top \mathbf{b} = 0$ . Hence,  $\mathbf{y}^* = \bar{\mathbf{y}} + \mathbf{y}'$  is an optimal solution to (D) with  $\mathbf{y}^{*\top} \mathbf{A}_i < c_i$ .

**Case 2:**  $u' < 0$ .

Then  $\mathbf{y}'^\top \mathbf{b} + vu' = 0$ , implying that  $\frac{1}{|u'|} \mathbf{y}'^\top \mathbf{b} = v$ . Let  $\mathbf{y}^* = \frac{1}{|u|} \mathbf{y}'$ . Then  $\mathbf{y}^*$  is an optimal solution to (D) with  $\mathbf{y}^{*\top} \mathbf{A}_i < c_i$ .

## 9.7 Basic feasible solution

For a linear constraint  $\mathbf{a}^\top \mathbf{x} \sqcup \gamma$  where  $\sqcup$  is  $\geq$ ,  $\leq$ , or  $=$ , we call  $\mathbf{a}^\top$  the **coefficient row-vector** of the constraint.

Let  $S$  denote a system of linear constraints with  $n$  variables and  $m$  constraints given by  $\mathbf{a}^{(i)\top} \mathbf{x} \sqcup_i b_i$  where  $\sqcup_i$  is  $\geq$ ,  $\leq$ , or  $=$  for  $i = 1, \dots, m$ .

For  $\mathbf{x}' \in \mathbb{R}^n$ , let  $J(S, \mathbf{x}')$  denote the set  $\{i : \mathbf{a}^{(i)\top} \mathbf{x}' = b_i\}$  and define  $\mathbf{A}_{S, \mathbf{x}'}$  to be the matrix whose rows are precisely the coefficient row-vectors of the constraints indexed by  $J(S, \mathbf{x}')$ .

### Example 9.9

Suppose that  $S$  is the system

$$x_1 + x_2 - x_3 \geq 2$$

$$3x_1 - x_2 + x_3 = 2$$

$$2x_1 - x_2 \leq 1$$

If  $\mathbf{x}' = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$ , then  $J(S, \mathbf{x}') = \{1, 2\}$  since  $\mathbf{x}'$  satisfies the first two constraints with equality but not the third. Hence,  $\mathbf{A}_{S, \mathbf{x}'} = \begin{bmatrix} 1 & 1 & -1 \\ 3 & -1 & 1 \end{bmatrix}$ .

### Definition 9.10

A solution  $\mathbf{x}^*$  to  $S$  is called a **basic feasible solution** if the rank of  $\mathbf{A}_{S, \mathbf{x}^*}$  is  $n$ .

A basic feasible solution to the system in Example 9.7 is  $\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ .

It is not difficult to see that in two dimensions, basic feasible solutions correspond to “corner points” of the set of all solutions. Therefore, the notion of a basic feasible solution generalizes the idea of a corner point to higher dimensions.

The following result is the basis for what is commonly known as the **corner method** for solving linear programming problems in two variables.

### Theorem 9.11: Basic Feasible Optimal Solution

*Let  $(P)$  be a linear programming problem. Suppose that  $(P)$  has an optimal solution and there exists a basic feasible solution to its constraints. Then there exists an optimal solution that is a basic feasible solution.*

We first state the following simple fact from linear algebra:

### Lemma 9.12

*Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{d} \in \mathbb{R}^n$  be such that  $\mathbf{A}\mathbf{d} = 0$ . If  $\mathbf{q} \in \mathbb{R}^m$  satisfies  $\mathbf{q}^\top \mathbf{d} \neq 0$  then  $\mathbf{q}^\top$  is not in the row space of  $\mathbf{A}$ .*

### Proof.

*Proof of Theorem 9.7.*

Suppose that the system of constraints in  $(P)$ , call it  $S$ , has  $m$  constraints and  $n$  variables. Let the objective function be  $\mathbf{c}^\top \mathbf{x}$ . Let  $v$  denote the optimal value.

Let  $\mathbf{x}^*$  be an optimal solution to  $(P)$  such that the rank of  $\mathbf{A}_{S,\mathbf{x}^*}$  is as large as possible. We claim that  $\mathbf{x}^*$  must be a basic feasible solution.

To ease notation, let  $J = J(S, \mathbf{x}^*)$ . Let  $N = \{1, \dots, m\} \setminus J$ .

Suppose to the contrary that the rank of  $\mathbf{A}_{S,\mathbf{x}^*}$  is less than  $n$ . Let  $\mathbf{Px} = \mathbf{q}$  denote the system of equations obtained by setting the constraints indexed by  $J$  to equalities. Then  $\mathbf{Px} = \mathbf{A}_{S,\mathbf{x}^*}$ . Since  $\mathbf{P}$  has  $n$  columns and its rank is less than  $n$ , there exists a nonzero  $\mathbf{d}$  such that  $\mathbf{Pd} = 0$ .

As  $\mathbf{x}^*$  satisfies each constraint indexed by  $N$  strictly, for a sufficiently small  $\varepsilon > 0$ ,  $\mathbf{x}^* + \varepsilon \mathbf{d}$  and  $\mathbf{x}^* - \varepsilon \mathbf{d}$  are solutions to  $S$  and therefore are feasible to  $(P)$ . Thus,

$$\begin{aligned} \mathbf{c}^\top (\mathbf{x}^* + \varepsilon \mathbf{d}) &\geq v \\ \mathbf{c}^\top (\mathbf{x}^* - \varepsilon \mathbf{d}) &\geq v. \end{aligned} \tag{9.1}$$

Since  $\mathbf{x}^*$  is an optimal solution, we have  $\mathbf{c}^\top \mathbf{x}^* = v$ . Hence, (9.1) simplifies to

$$\begin{aligned} \varepsilon \mathbf{c}^\top \mathbf{d} &\geq 0 \\ -\varepsilon \mathbf{c}^\top \mathbf{d} &\geq 0, \end{aligned}$$

giving us  $\mathbf{c}^\top \mathbf{d} = 0$  since  $\varepsilon > 0$ .

Without loss of generality, assume that the constraints indexed by  $N$  are  $\mathbf{Qx} \geq \mathbf{r}$ . As  $(P)$  does have a basic feasible solution, implying that the rank of  $\begin{bmatrix} \mathbf{P} \\ \mathbf{Q} \end{bmatrix}$  is  $n$ , at least one row of  $\mathbf{Q}$ , which we denote by  $\mathbf{t}^\top$ , must satisfy  $\mathbf{t}^\top \mathbf{d} \neq 0$ . Without loss of generality, we may assume that  $\mathbf{t}^\top \mathbf{d} > 0$ , replacing  $\mathbf{d}$  with  $-\mathbf{d}$  if necessary. Consider the linear programming problem

$$\begin{aligned} \min \quad & \lambda \\ \text{s.t.} \quad & \mathbf{Q}(\mathbf{x}^* + \lambda \mathbf{d}) \geq \mathbf{p} \end{aligned}$$

Since at least one entry of  $\mathbf{Q}\mathbf{d}$  is positive (namely,  $\mathbf{t}^\top \mathbf{d}$ ), this problem must have an optimal solution, say  $\lambda'$ . Setting  $\mathbf{x}' = \mathbf{x}^* + \lambda' \mathbf{d}$ , we have that  $\mathbf{x}'$  is an optimal solution since  $\mathbf{c}^\top \mathbf{x}' = v$ .

Now,  $\mathbf{x}'$  must satisfy at least one constraint in  $\mathbf{Q} \geq \mathbf{p}$  with equality. Let  $\mathbf{q}^\top$  be the coefficient row-vector of one such constraint. Then the rows of  $\mathbf{A}_{S,\mathbf{x}'}$  must have all the rows of  $\mathbf{A}_{S,\mathbf{x}^*}$  and  $\mathbf{q}^\top$ . Since  $\mathbf{q}^\top \mathbf{d} \neq 0$ , by Lemma 9.7, the rank of  $\mathbf{A}_{S,\mathbf{x}'}$  is larger than the rank of  $\mathbf{A}_{S,\mathbf{x}^*}$ , contradicting our choice of  $\mathbf{x}^*$ . Thus,  $\mathbf{x}^*$  must be a basic feasible solution. ♠

## Exercises

---

1. Find all basic feasible solutions to

$$\begin{aligned} x_1 + 2x_2 - x_3 &\geq 1 \\ x_2 + 2x_3 &\geq 3 \\ -x_1 + 2x_2 + x_3 &\geq 3 \\ -x_1 + x_2 + x_3 &\geq 0. \end{aligned}$$

2. A set  $S \subset \mathbb{R}^n$  is said to be bounded if there exists a real number  $M > 0$  such that for every  $\mathbf{x} \in S$ ,  $|x_i| < M$  for all  $i = 1, \dots, n$ . Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{b} \in \mathbb{R}^m$ . Prove that if  $\{\mathbf{x} : \mathbf{Ax} \geq \mathbf{b}\}$  is nonempty and bounded, then there is a basic feasible solution to  $\mathbf{Ax} \geq \mathbf{b}$ .
3. Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and  $\mathbf{b} \in \mathbb{R}^m$  where  $m$  and  $n$  are positive integers with  $m \leq n$ . Suppose that the rank of  $\mathbf{A}$  is  $m$  and  $\mathbf{x}'$  is a basic feasible solution to

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq 0. \end{aligned}$$

Let  $J = \{i : x'_i > 0\}$ . Prove that the columns of  $\mathbf{A}$  indexed by  $J$  are linearly independent.

## Solutions

---

1. To obtain all the basic feasible solutions, it suffices to enumerate all subsystems  $\mathbf{A}'\mathbf{x} \geq \mathbf{b}'$  of the given system such that the rank of  $\mathbf{A}'$  is three and solve  $\mathbf{A}'\mathbf{x} = \mathbf{b}'$  for  $\mathbf{x}$  and see if it is a solution to the system, in which case it is a basic feasible solution. Observe that every basic feasible solution can be discovered in this manner.

We have at most four subsystems to consider.

Setting the first three inequalities to equality gives the unique solution  $\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$  which satisfies the given

system.. Hence,  $\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$  is a basic feasible solution.

Setting the first, second, and fourth inequalities to equality gives the unique solution  $\begin{bmatrix} 5 \\ \frac{5}{3} \\ \frac{1}{3} \\ \frac{4}{3} \\ 3 \end{bmatrix}$  which violates the third inequality of the given system.

Setting the first, third, and fourth inequalities to equality leads to no solution. (In fact, the coefficient matrix of the system does not have rank 3 and therefore this case can be ignored.)

Setting the last three inequalities to equality gives the unique solution  $\begin{bmatrix} 3 \\ 3 \\ 0 \end{bmatrix}$  which satisfies the given

system. Hence,  $\begin{bmatrix} 3 \\ 3 \\ 0 \end{bmatrix}$  is a basic feasible solution.

Thus,  $\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 3 \\ 1 \\ 0 \end{bmatrix}$  are the only basic feasible solutions.

2. Let  $S$  denote the system  $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ . Let  $\mathbf{x}'$  be a solution to  $S$  such that the rank of  $\mathbf{A}_{S,\mathbf{x}'}$  is as large as possible. If the rank is  $n$ , then we are done. Otherwise, there exists nonzero  $\mathbf{d} \in \mathbb{R}^n$  such  $\mathbf{A}_{S,\mathbf{x}'}\mathbf{d} = 0$ . Since the set of solutions to  $S$  is a bounded set, at least one of the following values is finite:

- $\max\{\lambda : \mathbf{A}(\mathbf{x}' + \lambda\mathbf{d}) \geq \mathbf{b}\}$
- $\min\{\lambda : \mathbf{A}(\mathbf{x}' + \lambda\mathbf{d}) \geq \mathbf{b}\}$

Without loss of generality, assume that the maximum is finite and is equal to  $\lambda^*$ . Setting  $\mathbf{x}^*$  to  $\mathbf{x}' + \lambda^*\mathbf{d}$ , we have that the rows of  $\mathbf{A}_{S,\mathbf{x}^*}$  contains all the rows of  $\mathbf{A}_{S,\mathbf{x}'}$  plus at least one additional row, say  $\mathbf{q}^\top$ . Since  $\mathbf{q}^\top\mathbf{d} \neq 0$ , by Lemma 9.7, the rank of  $\mathbf{A}_{S,\mathbf{x}^*}$  is larger than the rank of  $\mathbf{A}_{S,\mathbf{x}'}$ , contradicting our choice of  $\mathbf{x}'$ .

3. The system of equations obtained from taking all the constraints satisfied with equality by  $\mathbf{x}'$  is

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ x_j &= 0 \quad j \notin J. \end{aligned} \tag{9.2}$$

Note that the coefficient matrix of this system has rank  $n$  if and only if it has a unique solution. Now, (9.2) simplifies to

$$\sum_{j \in J} x_j \mathbf{A}_j = \mathbf{b},$$

which has a unique solution if and only if the columns of  $\mathbf{A}$  indexed by  $J$  are linearly independent.

# 10. LP Notes from ISE 5405

---

## 10.1 Introduction to Optimization

---

Optimization (i.e., Mathematical Programming) seeks to select, from a set of alternative solutions (decisions), a solution that is “best” for a given performance criteria (i.e., maximize or minimizes the criteria). The following is a general optimization problem:

$$\max\{f(\mathbf{x}, \mathbf{y}) : A(\mathbf{x}) + G(\mathbf{y}) \leq \mathbf{b}_1, H(\mathbf{x}) + W(\mathbf{y}) = \mathbf{b}_2, \mathbf{x} \in \mathcal{X}^+, \mathbf{y} \in \mathcal{R}^+\},$$

where  $\mathbf{x}$  and  $\mathbf{y}$  are vectors of decision variables,  $f(\mathbf{x}, \mathbf{y})$  is the *objective function*, which defines the “best” solution (in this case the optimization problem seeks to maximize the objective function), and  $A(\mathbf{x}) + G(\mathbf{y}) \leq \mathbf{b}_1$ ,  $H(\mathbf{x}) + W(\mathbf{y}) = \mathbf{b}_2$ ,  $\mathbf{x} \in \mathcal{X}^+$ , and  $\mathbf{y} \in \mathcal{R}^+$  are the *constraints* that define the set of possible solutions.

Depending on the nature of the objective function, the constraints, and the input parameters, we can make some broad classifications of optimization problems, as follows:

**Linear Optimization:** Linear optimization, i.e., a linear program (LP), has a linear objective function subject to a set of linear constraints and continuous decision variables.

**Definition:** A function  $f(x_1, x_2, \dots, x_n)$  is linear if, and only if, we have  $f(x_1, x_2, \dots, x_n) = c_1x_1 + c_2x_2 + \dots + c_nx_n$ , where the  $c_1, c_2, \dots, c_n$  coefficients are constants.

An LP has the following general form:

$$\max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} = \mathbf{b}, \mathbf{x} \in \mathcal{R}\},$$

where  $\mathbf{x}$  is a vector of decision variables, and the vectors  $\mathbf{c}$  and  $\mathbf{b}$ , as well as the matrix  $\mathbf{A}$ , are constant problem parameters.

**Nonlinear Optimization:** Nonlinear optimization, i.e., a nonlinear program, is similar to an LP, but objective function and/or the constraints are nonlinear.

**Integer Optimization:** Integer optimization, , i.e., an integer program (IP), is much like an LP, but some variables restricted to take only integer values.

To use optimization, first you must formulate your model, based on the system of interest and any simplifications required (i.e., assumptions). Formulating the model is not enough, we are also interested in solving the problem, and in a reasonable amount of time (however that is determined). To solve these problems, algorithms are developed. An algorithms is a step-by-step process for finding a solution. We can broadly define different types of algorithms as follows:

- Optimal algorithms - processes that solve the model to optimality (and proves optimality).
- Near-optimal algorithms with bounds (heuristics), processes that do not guarantee optimality, but provides "good" solutions with known bounds.
- Other heuristic algorithms, processes that provide a "good" solution, but bounds are not provided, or are not that useful.

Algorithms can also be categorized based on performance, for instance, usually a *polynomial time algorithm* is better than an *exponential time algorithm*.

The class will almost exclusively focus on Linear Programs (LP) because: 1) LP are useful for many problems; 2) LPs are, relatively, easy to solve; and most importantly 3) LP are an important foundation for further courses in optimization.

### 10.1.1. Notation

---

- We use bold text to indicate a matrix or vector, e.g., the matrix **A** or the vector **x**.

## 10.2 Linear Optimization

---

In this section, we study on linear optimization problems, i.e., linear programs (LPs).

### 10.2.1. Problem Formulation

---

Remember, for a linear program (LP), we want to maximize or minimize a linear **objective function** of the continuous decision variables, while considering linear constraints on the values of the decision variables.

#### Definition 10.1: Linear Function

*function  $f(x_1, x_2, \dots, x_n)$  is linear if, and only if, we have  $f(x_1, x_2, \dots, x_n) = c_1x_1 + c_2x_2 + \dots + c_nx_n$ , where the  $c_1, c_2, \dots, c_n$  coefficients are constants.*

#### A Generic Linear Program (LP)

---

Decision Variables:

$x_i$  : continuous variables ( $x_i \in \mathcal{R}$ , i.e., a real number),  $\forall i = 1, \dots, 3$ .

Parameters (known input parameters):

$c_i$  : cost coefficients  $\forall i = 1, \dots, 3$

$a_{ij}$  : constraint coefficients  $\forall i = 1, \dots, 3, j = 1, \dots, 4$

$b_j$  : right hand side coefficient for constraint  $j$ ,  $j = 1, \dots, 4$

$$\text{Min } z = c_1x_1 + c_2x_2 + c_3x_3 \quad (10.1)$$

$$\text{s.t. } a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \geq b_1 \quad (10.2)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \leq b_2 \quad (10.3)$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \quad (10.4)$$

$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 \geq b_4 \quad (10.5)$$

$$x_1 \geq 0, x_2 \leq 0, x_3 \text{ urs.} \quad (10.6)$$

Eq. (10.1) is the objective function, (10.2)-(10.5) are the functional constraints, while (10.6) is the sign restrictions (*urs* signifies that the variable is unrestricted). If we were to add any one of these following constraints  $x_2 \in \{0, 1\}$  ( $x_2$  is binary-valued) or  $x_3 \in \mathcal{Z}$  ( $x_3$  is integer-valued) we would have an Integer Program. For the purposes of this class, an Integer Program (IP) is just an LP with added integer restrictions on (some) variables.

While, in general, solvers will take any form of the LP, there are some special forms we use in analysis:

**LP Standard Form:** The standard form has all constraints as equalities, and all variables as non-negative. The generic LP is not in standard form, but any LP can be converted to standard form.

Since  $x_2$  is non-positive and  $x_3$  unrestricted, perform the following substitutions  $x_2 = -\hat{x}_2$  and  $x_3 = x_3^+ - x_3^-$ , where  $\hat{x}_2, x_3^+, x_3^- \geq 0$ . Eqs. (10.2) and (10.5) are in the form left-hand side (LHS)  $\geq$  right-hand side (RHS), so to make an equality, subtract a non-negative slack variable from the LHS ( $s_1$  and  $s_4$ ). Eq. (10.3) is in the form LHS  $\leq$  RHS, so add a non-negative slack variable to the LHS.

$$\begin{aligned} \text{Min } z &= c_1x_1 - c_2\hat{x}_2 + c_3(x_3^+ - x_3^-) \\ \text{s.t. } a_{11}x_1 - a_{12}x_2 + a_{13}(x_3^+ - x_3^-) - s_1 &= b_1 \\ a_{21}x_1 - a_{22}\hat{x}_2 + a_{23}(x_3^+ - x_3^-) + s_2 &= b_2 \\ a_{31}x_1 - a_{32}\hat{x}_2 + a_{33}(x_3^+ - x_3^-) &= b_3 \\ a_{41}x_1 - a_{42}\hat{x}_2 + a_{43}x_3 - s_4 &= b_4 \\ x_1, \hat{x}_2, x_3^+, x_3^-, s_1, s_2, s_4 &\geq 0. \end{aligned}$$

**LP Canonical Form:** For a minimization problem the canonical form of the LP has the LHS of each constraint greater than or equal to the the RHS, and a maximization the LHS less than or equal to the RHS, and non-negative variables.

Next we consider some formulation examples:

**Production Problem:** You have 21 units of transparent aluminum alloy (TAA), LazWeld1, a joining robot leased for 23 hours, and CrumCut1, a cutting robot leased for 17 hours of aluminum cutting. You also have production code for a bookcase, desk, and cabinet, along with commitments to buy any of these you can produce for \$18, \$16, and \$10 apiece, respectively. A bookcase requires 2 units of TAA, 3 hours of joining, and 1 hour of cutting, a desk requires 2 units of TAA, 2 hours of joining, and 2 hour of cutting, and a cabinet requires 1 unit of TAA, 2 hours of joining, and 1 hour of cutting. Formulate an LP to maximize your revenue given your current resources.

Decision variables:

$x_i$  : number of units of product  $i$  to produce,

$\forall i = \{\text{bookcase, desk, cabinet}\}$ .

$$\begin{aligned} \max z &= 18x_1 + 16x_2 + 10x_3 : \\ 2x_1 + 2x_2 + 1x_3 &\leq 21 && (\text{TAA}) \\ 3x_1 + 2x_2 + 2x_3 &\leq 23 && (\text{LazWeld1}) \\ 1x_1 + 2x_2 + 1x_3 &\leq 17 && (\text{CrumCut1}) \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

**Work Scheduling Problem:** You are the manager of LP Burger. The following table shows the minimum number of employees required to staff the restaurant on each day of the week. Each employees must work for five consecutive days. Formulate an LP to find the minimum number of employees required to staff the restaurant.

Decision variables:

$x_i$  : the number of workers that start 5 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$

Day of Week	Workers Required
1 = Monday	6
2 = Tuesday	4
3 = Wednesday	5
4 = Thursday	4
5 = Friday	3
6 = Saturday	7
7 = Sunday	7

$$\begin{aligned}
 \text{Min } z &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \\
 \text{s.t. } x_1 + x_4 + x_5 + x_6 + x_7 &\geq 6 \\
 x_2 + x_5 + x_6 + x_7 + x_1 &\geq 4 \\
 x_3 + x_6 + x_7 + x_1 + x_2 &\geq 5 \\
 x_4 + x_7 + x_1 + x_2 + x_3 &\geq 4 \\
 x_5 + x_1 + x_2 + x_3 + x_4 &\geq 3 \\
 x_6 + x_2 + x_3 + x_4 + x_5 &\geq 7 \\
 x_7 + x_3 + x_4 + x_5 + x_6 &\geq 7 \\
 x_1, x_2, x_3, x_4, x_5, x_6, x_7 &\geq 0.
 \end{aligned}$$

The solution is as follows:

LP Solution	IP Solution
$z_{LP} = 7.333$	$z_I = 8.0$
$x_1 = 0$	$x_1 = 0$
$x_2 = 0.333$	$x_2 = 0$
$x_3 = 1$	$x_3 = 0$
$x_4 = 2.333$	$x_4 = 3$
$x_5 = 0$	$x_5 = 0$
$x_6 = 3.333$	$x_6 = 4$
$x_7 = 0.333$	$x_7 = 1$

LP Burger has changed its policy, and allows, at most, two part time workers, who work for two consecutive days in a week. Formulate this problem.

Decision variables:

$x_i$  : the number of workers that start 5 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$

$y_i$  : the number of workers that start 2 consecutive days of work on day  $i$ ,  $i = 1, \dots, 7$ .

$$\begin{aligned}
\text{Min } z &= 5(x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7) \\
&\quad + 2(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7) \\
\text{s.t. } x_1 + x_4 + x_5 + x_6 + x_7 + y_1 + y_7 &\geq 6 \\
x_2 + x_5 + x_6 + x_7 + x_1 + y_2 + y_1 &\geq 4 \\
x_3 + x_6 + x_7 + x_1 + x_2 + y_3 + y_2 &\geq 5 \\
x_4 + x_7 + x_1 + x_2 + x_3 + y_4 + y_3 &\geq 4 \\
x_5 + x_1 + x_2 + x_3 + x_4 + y_5 + y_4 &\geq 3 \\
x_6 + x_2 + x_3 + x_4 + x_5 + y_6 + y_5 &\geq 7 \\
x_7 + x_3 + x_4 + x_5 + x_6 + y_7 + y_6 &\geq 7 \\
y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 &\leq 2 \\
x_i &\geq 0, y_i \geq 0, \forall i = 1, \dots, 7.
\end{aligned}$$

**The Diet Problem:** In the future (as envisioned in a bad 70's science fiction film) all food is in tablet form, and there are four types, green, blue, yellow, and red. A balanced, futuristic diet requires, at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D. Formulate an LP that ensures a balanced diet at the minimum possible cost.

Tablet	Iron	B	C	D	Cost (\$)
green (1)	6	6	7	4	1.25
blue (2)	4	5	4	9	1.05
yellow (3)	5	2	5	6	0.85
red (4)	3	6	3	2	0.65

Now we formulate the problem:

Decision variables:

$x_i$  : number of tablet of type  $i$  to include in the diet,  $\forall i \in \{1, 2, 3, 4\}$ .

$$\begin{aligned}
\text{Min } z &= 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4 \\
\text{s.t. } 6x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 20 \\
6x_1 + 5x_2 + 2x_3 + 6x_4 &\geq 25 \\
7x_1 + 4x_2 + 5x_3 + 3x_4 &\geq 30 \\
4x_1 + 9x_2 + 6x_3 + 2x_4 &\geq 15 \\
x_1, x_2, x_3, x_4 &\geq 0.
\end{aligned}$$

**The Next Diet Problem:** Progress is important, and our last problem had too many tablets, so we are going to produce a single, purple, 10 gram tablet for our futuristic diet requires, which are at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D, and 2000 calories. The tablet is made from blending 4 nutritious chemicals; the following table shows the units of our nutrients

Tablet	Iron	B	C	D	Calories	Cost (\$)
Chem 1	6	6	7	4	1000	1.25
Chem 2	4	5	4	9	250	1.05
Chem 3	5	2	5	6	850	0.85
Chem 4	3	6	3	2	750	0.65

per, and cost of, grams of each chemical. Formulate an LP that ensures a balanced diet at the minimum possible cost.

#### Decision variables:

$x_i$  : grams of chemical  $i$  to include in the purple tablet,  $\forall i = 1, 2, 3, 4$ .

$$\begin{aligned}
 \text{Min} z &= 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4 \\
 \text{s.t. } &6x_1 + 4x_2 + 5x_3 + 3x_4 \geq 20 \\
 &6x_1 + 5x_2 + 2x_3 + 6x_4 \geq 25 \\
 &7x_1 + 4x_2 + 5x_3 + 3x_4 \geq 30 \\
 &4x_1 + 9x_2 + 6x_3 + 2x_4 \geq 15 \\
 &1000x_1 + 250x_2 + 850x_3 + 750x_4 \geq 2000 \\
 &x_1 + x_2 + x_3 + x_4 = 10 \\
 &x_1, x_2, x_3, x_4 \geq 0.
 \end{aligned}$$

**The Assignment Problem:** Consider the assignment of  $n$  teams to  $n$  projects, where each team ranks the projects, where their favorite project is given a rank of  $n$ , their next favorite  $n - 1$ , and their least favorite project is given a rank of 1. The assignment problem is formulated as follows (we denote ranks using the  $R$ -parameter):

#### Variables:

$x_{ij}$  : 1 if project  $i$  assigned to team  $j$ , else 0.

$$\begin{aligned}
 \text{Max } z &= \sum_{i=1}^n \sum_{j=1}^n R_{ij}x_{ij} \\
 \text{s.t. } &\sum_{i=1}^n x_{ij} = 1, \quad \forall j = 1, \dots, n \\
 &\sum_{j=1}^n x_{ij} = 1, \quad \forall i = 1, \dots, n \\
 &x_{ij} \geq 0, \quad \forall i = 1, \dots, n, j = 1, \dots, n.
 \end{aligned}$$

The assignment problem has an integrality property, such that if we remove the binary restriction on the  $x$  variables (now just non-negative, i.e.,  $x_{ij} \geq 0$ ) then we still get binary assignments, despite the fact that it is now an LP. This property is very interesting and useful. Of course, the objective function might not quite what we want, we might be interested ensuring that the team with the worst assignment is as good as possible (a fairness criteria). One way of doing this is to modify the assignment problem using a max-min objective:

### Max-min Assignment-like Formulation

$$\begin{aligned}
 & \text{Max} \quad z \\
 \text{s.t.} \quad & \sum_{i=1}^n x_{ij} = 1, \quad \forall j = 1, \dots, n \\
 & \sum_{j=1}^n x_{ij} = 1, \quad \forall i = 1, \dots, n \\
 & x_{ij} \geq 0, \quad \forall i = 1, \dots, n, J = 1, \dots, n \\
 & z \leq \sum_{i=1}^n R_{ij} x_{ij}, \quad \forall j = 1, \dots, n.
 \end{aligned}$$

Does this formulation have the integrality property (it is not an assignment problem)? Consider a very simple example where two teams are to be assigned to two projects and the teams give the projects the following rankings: Both teams prefer Project 2. For both problems, if we remove the binary restriction on

	Project 1	Project 2
Team 1	2	1
Team 2	2	1

the  $x$ -variable, they can take values between (and including) zero and one. For the assignment problem the optimal solution will have  $z = 3$ , and fractional  $x$ -values will not improve  $z$ . For the max-min assignment problem this is not the case, the optimal solution will have  $z = 1.5$ , which occurs when each team is assigned half of each project (i.e., for Team 1 we have  $x_{11} = 0.5$  and  $x_{21} = 0.5$ ).

**Linear Data Models:** Consider a data set that consists of  $n$  data points  $(x_i, y_i)$ . We want to fit the best line to this data, such that given an  $x$ -value, we can predict the associated  $y$ -value. Thus, the form is  $y_i = \alpha x_i + \beta$  and we want to choose the  $\alpha$  and  $\beta$  values such that we minimize the error for our  $n$  data points.

**Variables:**

$e_i$  : error for data point  $i$ ,  $i = 1, \dots, n$ .

$\alpha$  : slope of fitted line.

$\beta$  : intercept of fitted line.

$$\begin{aligned} \text{Min } & \sum_{i=1}^n |e_i| \\ \text{s.t. } & \alpha x_i + \beta - y_i = e_i, \quad i = 1, \dots, n \\ & e_i, \alpha, \beta \text{ urs.} \end{aligned}$$

Of course, absolute values are not linear function, so we can linearize as follows:

**Decision variables:**

$e_i^+$  : positive error for data point  $i$ ,  $i = 1, \dots, n$ .

$e_i^-$  : negative error for data point  $i$ ,  $i = 1, \dots, n$ .

$\alpha$  : slope of fitted line.

$\beta$  : intercept of fitted line.

$$\begin{aligned} \text{Min } & \sum_{i=1}^n e_i^+ + e_i^- \\ \text{s.t. } & \alpha x_i + \beta - y_i = e_i^+ - e_i^-, \quad i = 1, \dots, n \\ & e_i^+, e_i^- \geq 0, \alpha, \beta \text{ urs.} \end{aligned}$$

**Two-Person Zero-Sum Games:** Consider a game with two players,  $\mathcal{A}$  and  $\mathcal{B}$ . In each round of the game,  $\mathcal{A}$  chooses one out of  $m$  possible actions, while  $\mathcal{B}$  chooses one out of  $n$  actions. If  $\mathcal{A}$  takes action  $j$  while  $\mathcal{B}$  takes action  $i$ , then  $c_{ij}$  is the payoff for  $\mathcal{A}$ , if  $c_{ij} > 0$ ,  $\mathcal{A}$  "wins"  $c_{ij}$  (and  $\mathcal{B}$  losses that amount), and if  $c_{ij} < 0$  if  $\mathcal{B}$  "wins"  $-c_{ij}$  (and  $\mathcal{A}$  losses that amount). This is a two-person zero-sum game.

Rock, Paper, Scissors is a two-person zero-sum game, with the following payoff matrix.

		$\mathcal{A}$		
		R	P	S
$\mathcal{B}$	R	0	1	-1
	P	-1	0	1
	S	1	-1	0

We can have a similar game, but with a different payoff matrix, as follows:

		$\mathcal{A}$		
		R	P	S
$\mathcal{B}$	R	4	-1	-1
	P	-2	4	-2
	S	-3	-3	4

What is the optimal strategy for  $\mathcal{A}$  (for either game)? We define  $x_j$  as the probability that  $\mathcal{A}$  takes action  $j$  (related to the columns). Then the payoff for  $\mathcal{A}$ , if  $\mathcal{B}$  takes action  $i$  is  $\sum_{j=1}^m c_{ij}x_j$ . Of course,  $\mathcal{A}$  does not know what action  $\mathcal{B}$  will take, so let's find a strategy that maximizes the minimum expected winnings of  $\mathcal{A}$  given any random strategy of  $\mathcal{B}$ , which we can formulate as follows:

$$\begin{aligned} \text{Max } & \left( \min_{i=1,\dots,n} \sum_{j=1}^m c_{ij}x_j \right) \\ \text{s.t. } & \sum_{j=1}^m x_j = 1 \\ & x_j \geq 0, \quad i = 1, \dots, m, \end{aligned}$$

which can be linearized as follows:

$$\begin{aligned} \text{Max } & z \\ \text{s.t. } & z \leq \sum_{j=1}^m c_{ij}x_j, \quad i = 1, \dots, n \\ & \sum_{j=1}^m x_j = 1 \\ & x_j \geq 0, \quad i = 1, \dots, m. \end{aligned}$$

The last two constraints ensure the that  $x_i$ -variables are valid probabilities. If you solved this LP for the first game (i.e., payoff matrix) you find the best strategy is  $x_1 = 1/3$ ,  $x_2 = 1/3$ , and  $x_3 = 1/3$  and there is no expected gain for player  $\mathcal{A}$ . For the second game, the best strategy is  $x_1 = 23/107$ ,  $x_2 = 37/107$ , and  $x_3 = 47/107$ , with  $\mathcal{A}$  gaining, on average,  $8/107$  per round.

### 10.2.2. Linear Algebra Review

---

#### Vectors and Linear and Convex Combinations

**Vectors:** Vector  $\mathbf{n}$  has  $n$ -elements and represents a point (or an arrow from the origin to the point, denoting a direction) in  $\mathbb{R}^n$  space (Euclidean or real space). Vectors can be expressed as either row or column vectors.

**Vector Addition:** Two vectors of the same size can be added, componentwise, e.g., for vectors  $\mathbf{a} = (2, 3)$  and  $\mathbf{b} = (3, 2)$ ,  $\mathbf{a} + \mathbf{b} = (2+3, 3+2) = (5, 5)$ .

**Scalar Multiplication:** A vector can be multiplied by a scalar  $k$  (constant) component-wise. If  $k > 0$  then this does not change the direction represented by the vector, it just scales the vector.

**Inner or Dot Product:** Two vectors of the same size can be multiplied to produce a real number. For example,  $\mathbf{ab} = 2 * 3 + 3 * 2 = 10$ .

**Linear Combination:** The vector  $\mathbf{b}$  is a **linear combination** of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  if  $\mathbf{b} = \sum_{i=1}^k \lambda_i \mathbf{a}_i$  for  $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathbb{R}$ . If  $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathbb{R}_{\geq 0}$  then  $\mathbf{b}$  is a *non-negative linear combination* of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$ .

**Convex Combination:** The vector  $\mathbf{b}$  is a **convex combination** of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  if  $\mathbf{b} = \sum_{i=1}^k \lambda_i \mathbf{a}_i$ , for  $\lambda_1, \lambda_2, \dots, \lambda_k \in \mathbb{R}_{\geq 0}$  and  $\sum_{i=1}^k \lambda_i = 1$ . For example, any convex combination of two points will lie on the

line segment between the points.

**Linear Independence:** Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  are *linearly independent* if the following linear combination  $\sum_{i=1}^k \lambda_i \mathbf{a}_i = 0$  implies that  $\lambda_i = 0$ ,  $i = 1, 2, \dots, k$ . In  $\mathcal{R}^2$  two vectors are only linearly dependent if they lie on the same line. Can you have three linearly independent vectors in  $\mathcal{R}^2$ ?

**Spanning Set:** Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  span  $\mathcal{R}^m$  if any vector in  $\mathcal{R}^m$  can be represented as a linear combination of  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$ , i.e.,  $\sum_{i=1}^m \lambda_i \mathbf{a}_i$  can represent any vector in  $\mathcal{R}^m$ .

**Basis:** Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  form a basis of  $\mathcal{R}^m$  if they span  $\mathcal{R}^m$  and any smaller subset of these vectors does not span  $\mathcal{R}^m$ . Vectors  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$  can only form a basis of  $\mathcal{R}^m$  if  $k = m$  and they are linearly independent.

## Convex and Polyhedral Sets

**Convex Set:** Set  $\mathcal{S}$  in  $\mathbb{R}^n$  is a *convex set* if a line segment joining any pair of points  $\mathbf{a}_1$  and  $\mathbf{a}_2$  in  $\mathcal{S}$  is completely contained in  $\mathcal{S}$ , that is,  $\lambda\mathbf{a}_1 + (1 - \lambda)\mathbf{a}_2 \in \mathcal{S}, \forall \lambda \in [0, 1]$ .

**Hyperplanes and Half-Spaces:** A hyperplane in  $\mathbb{R}^n$  divides  $\mathbb{R}^n$  into 2 half-spaces (like a line does in  $\mathbb{R}^2$ ). A hyperplane is the set  $\{\mathbf{x} : \mathbf{p}\mathbf{x} = k\}$ , where  $\mathbf{p}$  is the gradient to the hyperplane (i.e., the coefficients of our linear expression). The corresponding half-spaces is the set of points  $\{\mathbf{x} : \mathbf{p}\mathbf{x} \geq k\}$  and  $\{\mathbf{x} : \mathbf{p}\mathbf{x} \leq k\}$ .

**Polyhedral Set:** A *polyhedral set* (or polyhedron) is the set of points in the intersection of a finite set of half-spaces. Set  $\mathcal{S} = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ , where  $\mathbf{A}$  is an  $m \times n$  matrix,  $\mathbf{x}$  is an  $n$ -vector, and  $\mathbf{b}$  is an  $m$ -vector, is a *polyhedral set* defined by  $m+n$  hyperplanes (i.e., the intersection of  $m+n$  half-spaces).

- Polyhedral sets are convex.
- A polytope is a bounded polyhedral set.
- A polyhedral cone is a polyhedral set where the hyperplanes (that define the half-spaces) pass through the origin, thus  $\mathcal{C} = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq 0\}$  is a polyhedral cone.

**Edges and Faces:** An *edge* of a polyhedral set  $\mathcal{S}$  is defined by  $n-1$  hyperplanes, and a *face* of  $\mathcal{S}$  by one of more defining hyperplanes of  $\mathcal{S}$ , thus an extreme point and an edge are faces (an extreme point is a zero-dimensional face and an edge a one-dimensional face). In  $\mathbb{R}^2$  faces are only edges and extreme points, but in  $\mathbb{R}^3$  there is a third type of face, and so on...

**Extreme Points:**  $\mathbf{x} \in \mathcal{S}$  is an extreme point of  $\mathcal{S}$  if:

**Definition 1:**  $\mathbf{x}$  is not a convex combination of two other points in  $\mathcal{S}$ , that is, all line segments that are completely in  $\mathcal{S}$  that contain  $\mathbf{x}$  must have  $\mathbf{x}$  as an endpoint.

**Definition 2:**  $\mathbf{x}$  lies on  $n$  linearly independent defining hyperplanes of  $\mathcal{S}$ .

If more than  $n$  hyperplanes pass through an extreme points then it is a degenerate extreme point, and the polyhedral set is considered degenerate. This just adds a bit of complexity to the algorithms we will study, but it is quite common.

## Unbounded Sets:

**Rays:** A ray in  $\mathbb{R}^n$  is the set of points  $\{\mathbf{x} : \mathbf{x}_0 + \lambda\mathbf{d}, \lambda \geq 0\}$ , where  $\mathbf{x}_0$  is the vertex and  $\mathbf{d}$  is the direction of the ray.

**Convex Cone:** A *Convex Cone* is a convex set that consists of rays emanating from the origin. A convex cone is completely specified by its extreme directions. If  $\mathcal{C}$  is convex cone, then for any  $\mathbf{x} \in \mathcal{C}$  we have  $\lambda\mathbf{x} \in \mathcal{C}, \lambda \geq 0$ .

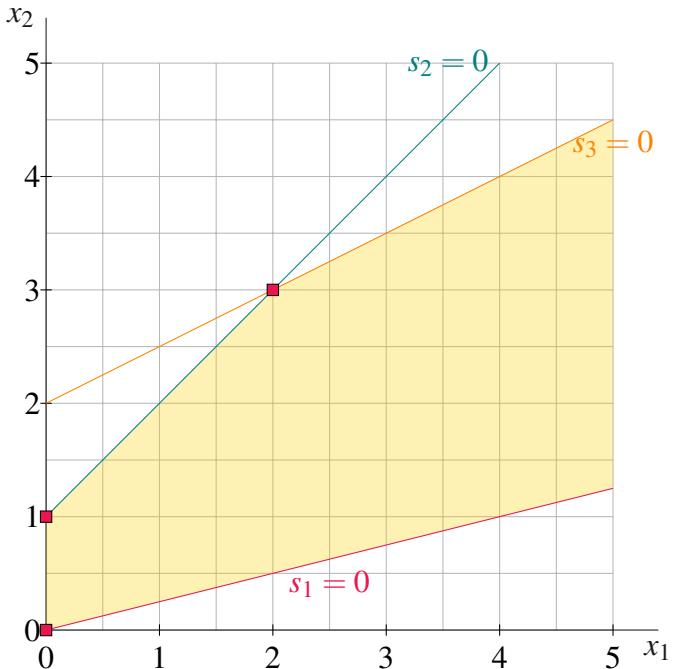
**Unbounded Polyhedral Sets:** If  $\mathcal{S}$  is unbounded, it will have *directions*.  $\mathbf{d}$  is a direction of  $\mathcal{S}$  only if  $\mathbf{Ax} + \lambda\mathbf{d} \leq \mathbf{b}, \mathbf{x} + \lambda\mathbf{d} \geq 0$  for all  $\lambda \geq 0$  and all  $\mathbf{x} \in \mathcal{S}$ . In other words, consider the ray  $\{\mathbf{x} : \mathbf{x}_0 + \lambda\mathbf{d}, \lambda \geq 0\}$

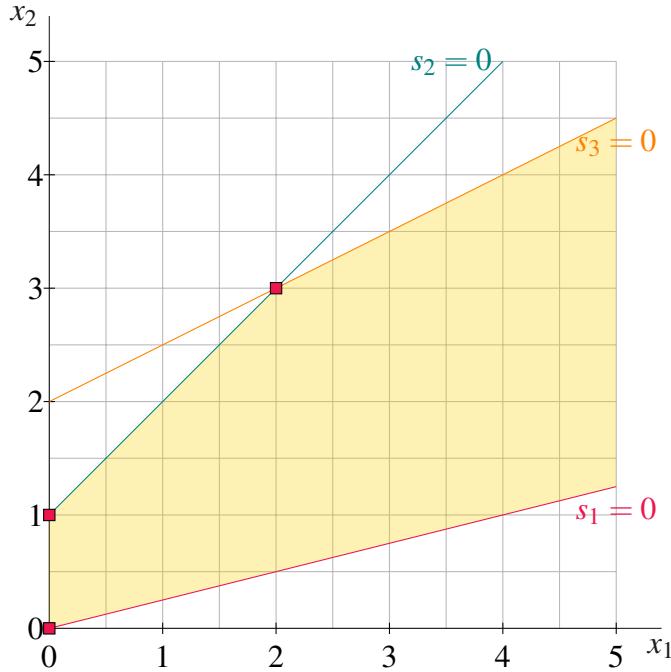
in  $\mathbb{R}^n$ , where  $\mathbf{x}_0$  is the vertex and  $\mathbf{d}$  is the direction of the ray.  $\mathbf{d} \neq 0$  is a **direction** of set  $\mathcal{S}$  if for each  $\mathbf{x}_0$  in  $\mathcal{S}$  the ray  $\{\mathbf{x}_0 + \lambda \mathbf{d}, \lambda \geq 0\}$  also belongs to  $\mathcal{S}$ .

**Extreme Directions:** An *extreme direction* of  $\mathcal{S}$  is a direction that *cannot* be represented as positive linear combination of other directions of  $\mathcal{S}$ . A non-negative linear combination of extreme directions can be used to represent all other directions of  $\mathcal{S}$ . A polyhedral cone is completely specified by its extreme directions.

Let's define a procedure for finding the extreme directions, using the following LP's feasible region. Graphically, we can see that the extreme directions should follow the the  $s_1 = 0$  (red) line and the  $s_3 = 0$  (orange) line.

$$\begin{aligned} \max \quad & z = -5x_1 - x_2 \\ \text{s.t.} \quad & x_1 - 4x_2 + s_1 = 0 \\ & -x_1 + x_2 + s_2 = 1 \\ & -x_1 + 2x_2 + s_3 = 4 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0. \end{aligned}$$



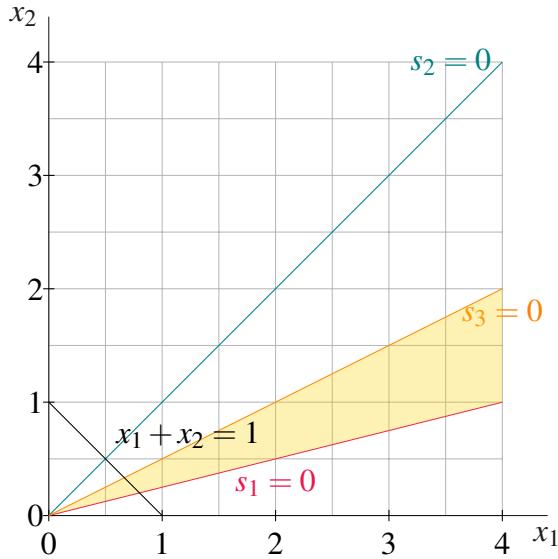


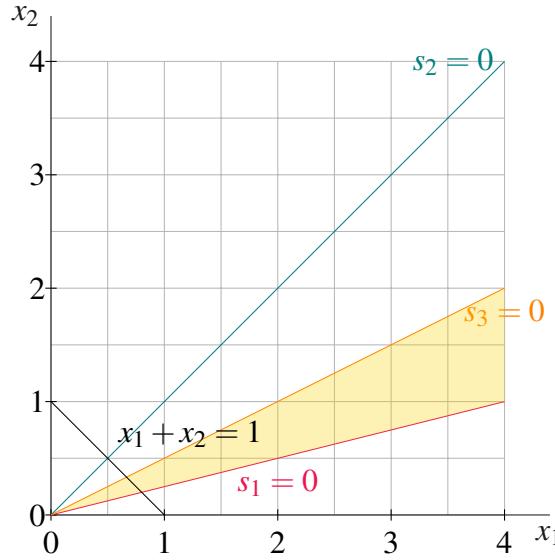
E.g., consider the  $s_3 = 0$  (orange) line, to find the extreme direction start at extreme point  $(2,3)$  and find another feasible point on the orange line, say  $(4,4)$  and subtract  $(2,3)$  from  $(4,4)$ , which yields  $(2,1)$ .

This is related to the slope in two-dimensions, as discussed in class, the rise is 1 and the run is 2. So this direction has a slope of  $1/2$ , but this does not carry over easily to higher dimensions where directions cannot be defined by a single number.

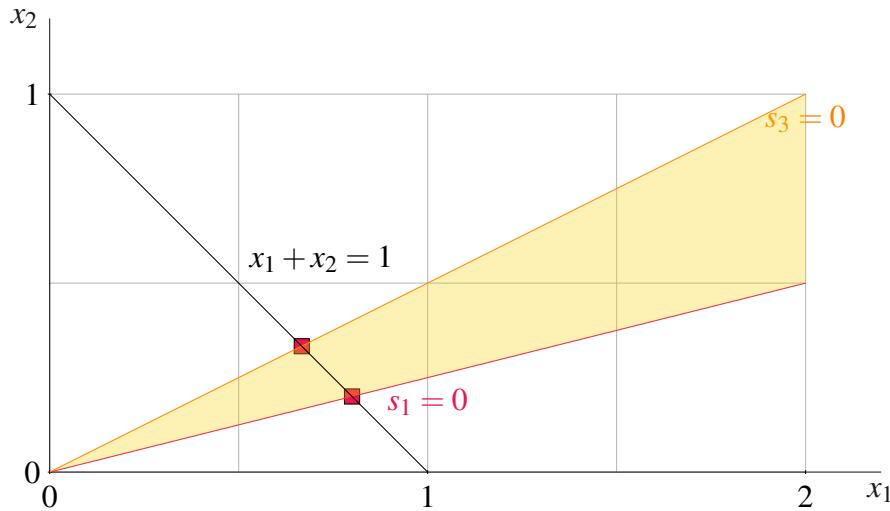
To find the extreme directions we can change the right-hand-side to  $\mathbf{b} = 0$ , which forms a polyhedral cone (in yellow), and then add the constraint  $x_1 + x_2 = 1$ . The intersection of the cone and  $x_1 + x_2 = 1$  form a line segment.

$$\begin{aligned} \max \quad & z = -5x_1 - x_2 \\ \text{s.t.} \quad & x_1 - 4x_2 + s_1 = 0 \\ & -x_1 + x_2 + s_2 = 0 \\ & -x_1 + 2x_2 + s_3 = 0 \\ & x_1 + x_2 = 1 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0. \end{aligned}$$





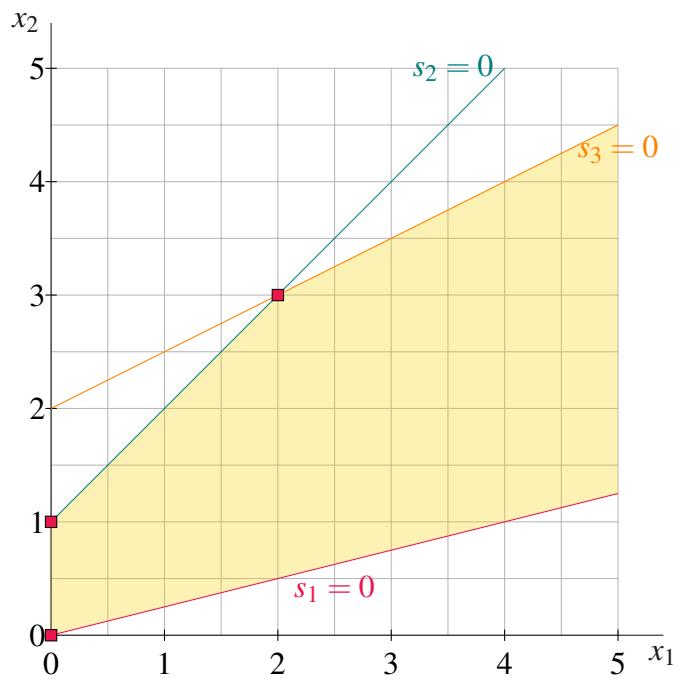
Magnifying for clarity, and removing the  $s_2 = 0$  (teal) line, as it is redundant, and marking the extreme points of the new feasible region,  $(4/5, 1/5)$  and  $(2/3, 1/3)$ , with red boxes, we have:



The extreme directions are thus  $(4/5, 1/5)$  and  $(2/3, 1/3)$ .

**Representation Theorem:** Let  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$  be the set of extreme points of  $\mathcal{S}$ , and if  $\mathcal{S}$  is unbounded,  $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_l$  be the set of extreme directions. Then any  $\mathbf{x} \in \mathcal{S}$  is equal to a convex combination of the extreme points and a non-negative linear combination of the extreme directions:  $\mathbf{x} = \sum_{j=1}^k \lambda_j \mathbf{x}_j + \sum_{j=1}^l \mu_j \mathbf{d}_j$ , where  $\sum_{j=1}^k \lambda_j = 1$ ,  $\lambda_j \geq 0$ ,  $\forall j = 1, 2, \dots, k$ , and  $\mu_j \geq 0$ ,  $\forall j = 1, 2, \dots, l$ .

$$\begin{aligned}
 \max \quad & z = -5x_1 - x_2 \\
 \text{s.t.} \quad & x_1 - 4x_2 + s_1 = 0 \\
 & -x_1 + x_2 + s_2 = 1 \\
 & -x_1 + 2x_2 + s_3 = 4 \\
 & x_1, x_2, s_1, s_2, s_3 \geq 0.
 \end{aligned}$$

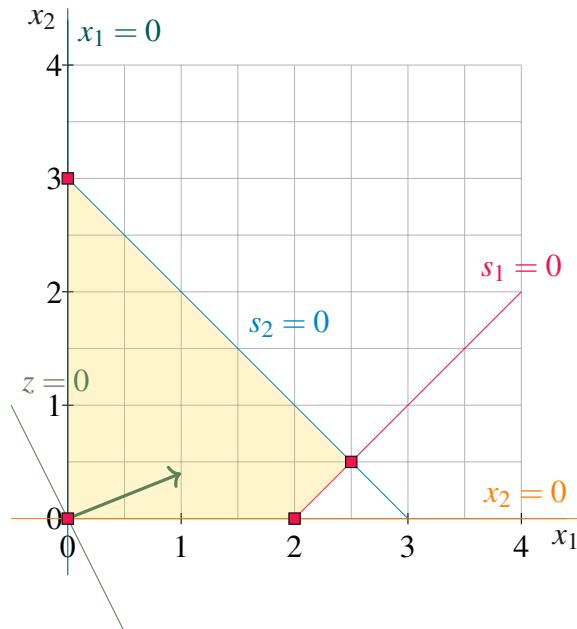


Represent point  $(1/2, 1)$  as a convex combination of the extreme points of the above LP. Find  $\lambda$ s to solve the following system of equations:

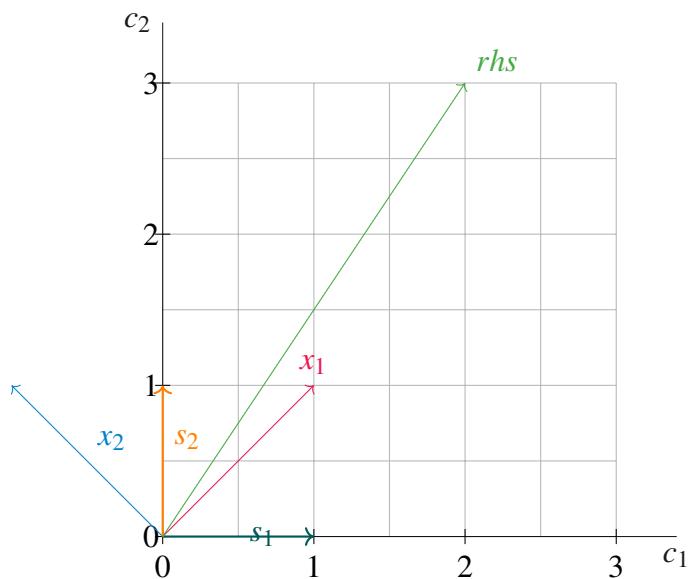
$$\lambda_1 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \lambda_3 \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$$

The Variable (Canonical Form) and Requirement Space

$$\begin{aligned} \max \quad & z = 2x_1 + x_2 \\ \text{s.t.} \quad & x_1 - x_2 + s_1 = 2 \\ & x_1 + x_2 + s_2 = 3 \\ & x_1, x_2, s_1, s_2 \geq 0. \end{aligned}$$



$$\begin{aligned} \max \quad & z = 2x_1 + x_2 \\ \text{s.t.} \quad & x_1 - x_2 + s_1 = 2 \\ & x_1 + x_2 + s_2 = 3 \\ & x_1, x_2, s_1, s_2 \geq 0. \end{aligned}$$



## Tableaus

After putting an LP into standard form, we can put the system of equations into a table form, the “tableau”.

$$\begin{aligned} \text{max } z &= 2x_1 + x_2 \\ \text{s.t. } x_1 - x_2 + s_1 &= 2 \\ x_1 + x_2 + s_2 &= 3 \\ x_1, x_2, s_1, s_2 &\geq 0. \end{aligned}$$

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
(r0 - z)	1	-2	-1	0	0	0
(r1 - $x_3$ )	0	1	-1	1	0	2
(r2 - $x_4$ )	0	1	1	0	1	3

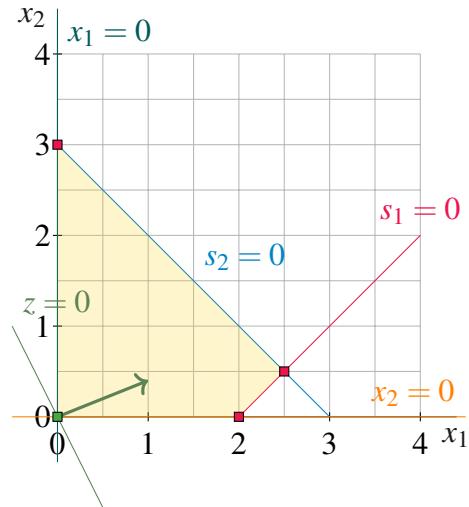
Why are the coefficients negative in row zero, we change  $z = 2x_1 + x_2$  to  $z - 2x_1 - x_2 = 0$  so we have only constants on the right-hand-side (rhs).

This tableau represents a basic solution, because it contains an identity matrix. The basic variables are those variables having columns in the identity matrix (here,  $x_3$  and  $x_4$ ), and it is feasible because the rhs for row 1 – m are non-negative.

We can consider  $z$  a permanent member of an expanded basis if we treat row zero like any other row (although the rhs of row 0 can be negative).

### Basic Solutions and Extreme Points

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
(r0 - z)	1	-2	-1	0	0	0
(r1 - $x_3$ )	0	1	-1	1	0	2
(r2 - $x_4$ )	0	1	1	0	1	3



Here the basic variables are  $x_3 = 2$  and  $x_4 = 3$ , and the  $z$ -value of objective function value is 0.

Let's go to the extreme point  $(2,0)$  which has basic variables  $x_1$  and  $x_4$  (this new extreme point is adjacent to the extreme point  $(0,0)$ ). Why?

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
(r0 - z)	1	-2	-1	0	0	0
(r1 - $x_3$ )	0	1	-1	1	0	2
(r2 - $x_4$ )	0	1	1	0	1	3

First get a 1 coefficient in row 1 in the  $x_1$  column by multiplying row 1 by a scalar (no action needed, already equals one).

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
(r0 - z)	1	-2	-1	0	0	0
(r1 - $x_3$ )	0	1	-1	1	0	2
(r2 - $x_4$ )	0	1	1	0	1	3

Then use row 1 to zero out the row 0 coefficient for  $x_1$  by multiplying row 1 by 2 and adding it to row 0 to get a new row 0.

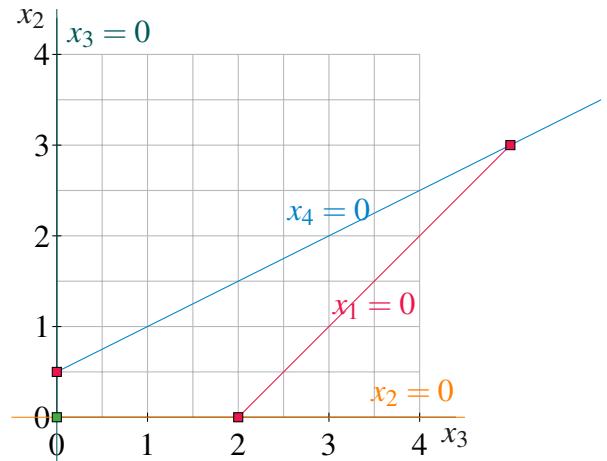
max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
(r0 - z)	1	0	-3	2	0	4
(r1 - $x_3$ )	0	1	-1	1	0	2
(r2 - $x_4$ )	0	1	1	0	1	3

Lastly use row 1 to zero out the row 2 coefficient for  $x_1$  by multiplying row 1 by -1 and adding it to row 2 to get a new row 2.

	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
(r0 -z)	1	0	-3	2	0	4
(r1 - $x_3$ )	0	1	-1	1	0	2
(r2 - $x_4$ )	0	0	2	-1	1	1

The nonbasic variables for this tableau are  $x_2$  and  $x_3$ , so we can graph the new LP in the nonbasic variable space.

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
(r0 -z)	1	0	-3	2	0	4
(r1 - $x_3$ )	0	1	-1	1	0	2
(r2 - $x_4$ )	0	0	2	-1	1	1



## Matrix Math

An  $m \times n$  matrix is an array of real numbers with  $m$  rows and  $n$  columns. Any matrix can be represented by its constituent set of row or column vectors.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 6 & 4 \end{bmatrix} = [\mathbf{a}_1 \quad \mathbf{a}_2] = \begin{bmatrix} \mathbf{a}^1 \\ \mathbf{a}^2 \end{bmatrix},$$

where  $\mathbf{a}_1 = \begin{bmatrix} 1 \\ 6 \end{bmatrix}$ ,  $\mathbf{a}_2 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ ,  $\mathbf{a}^1 = [1 \ 3]$ , and  $\mathbf{a}^2 = [6 \ 4]$ . Additionally,  $a_{11} = 1$ ,  $a_{12} = 3$ ,  $a_{21} = 6$ ,  $a_{22} = 4$ .

**Matrix Addition:** Two matrices of the same dimension can be added componentwise, thus  $\mathbf{C} = \mathbf{A} + \mathbf{B}$  means that  $c_{ij} = a_{ij} + b_{ij}$ .

**Scalar Multiplication:** Just like it sounds. If  $k$  is a scalar, then  $k\mathbf{A}$  means that every component of  $\mathbf{A}$  is multiplied by  $k$ .

**Matrix Multiplication:**  $\mathbf{A}$  is a  $m \times n$  matrix and  $\mathbf{B}$  is a  $p \times q$  matrix.  $\mathbf{AB}$  (matrix multiplication) is only defined if  $n = p$  and the result is a  $m \times q$  matrix,  $\mathbf{BA}$  is only defined if  $q = m$  and the result is a  $p \times n$ .  $\mathbf{AB}$  is not necessarily equal to  $\mathbf{BA}$ , thus  $\mathbf{C} = \mathbf{AB}$  where  $c_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, p$ , e.g.,  $c_{11}$  is the sum of the componentwise multiplication of the first row of  $\mathbf{A}$  and the first column of  $\mathbf{B}$ .

**Identity Matrix:** A square matrix (denoted by  $\mathbf{I}$ ) with all zero components, except for the diagonal:

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

**Elementary Matrix Operations:** These operations are used to solve systems of linear equations or inverting a matrix. The three operations are as follows (for any matrix  $\mathbf{A}$ ):

- Interchange two rows of  $\mathbf{A}$ .
- Multiply a row by a nonzero scalar.
- Replace row  $i$  with row  $i$  plus row  $j$  multiplied by a nonzero scalar.

### Inverting a Matrix:

$$\mathbf{A} = \begin{bmatrix} 3 & 9 & 2 \\ 1 & 1 & 1 \\ 5 & 4 & 7 \end{bmatrix} \text{ and } \mathbf{A}^{-1} = \begin{bmatrix} -\frac{3}{11} & \frac{5}{11} & -\frac{7}{11} \\ \frac{2}{11} & -1 & \frac{1}{11} \\ \frac{1}{11} & -3 & \frac{6}{11} \end{bmatrix}$$

To find  $\mathbf{A}^{-1}$  using elementary row operations to transform  $\mathbf{A}$  into an identity matrix, while performing these same operations on the attached identity matrix.

$$\left[ \begin{array}{ccc|ccc} 3 & 9 & 2 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 5 & 4 & 7 & 0 & 0 & 1 \end{array} \right] \Rightarrow \left[ \begin{array}{ccc|ccc} 1 & 0 & 0 & -\frac{3}{11} & 5 & -\frac{7}{11} \\ 0 & 1 & 0 & \frac{2}{11} & -1 & \frac{1}{11} \\ 0 & 0 & 1 & \frac{1}{11} & -3 & \frac{6}{11} \end{array} \right].$$

**Rank of a Matrix:**  $\mathbf{A}$  is a  $m \times n$  matrix then  $\text{rank}(\mathbf{A}) \leq \min\{m, n\}$ , if  $\text{rank}(\mathbf{A}) = \min\{m, n\}$ , then  $\mathbf{A}$  is of full rank. If  $\mathbf{A}$  is not full rank, but of rank  $k$ , where  $k < \min\{m, n\}$ , then using the elementary row operations, we can transform  $\mathbf{A}$  to the following:  $\left[ \begin{array}{cc} \mathbf{I}_k & \mathbf{Q} \\ 0 & 0 \end{array} \right]$

### 10.2.3. Linear Optimization Theory

---

Consider an arbitrary LP, which we will call the primal ( $P$ ):

$$(P) : \max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\},$$

where  $\mathbf{A}$  is an  $m \times n$  matrix, and  $\mathbf{x}$  is a  $n$  element column vector. Every primal LP has a related LP, which we call the dual, the dual of ( $P$ ) is:

$$(D) : \min\{\mathbf{w}\mathbf{b} : \mathbf{w}\mathbf{A} \geq \mathbf{c}, \mathbf{w} \geq 0\}.$$

Before we discuss properties of duality, and why it is important, we start with how to formulate the dual for any given LP. If the LP has a different form like  $P$ , we find the dual based on the  $P$  and  $D$  example above. If the LP does not have this form, we can transform it to this form, or use the rules in the following table, first noting that:

- The dual of problem  $D$  is problem  $P$ .
- Each primal constraint has an associated dual variable ( $w_i$ ) and each dual constraint has an associated primal variable ( $x_i$ ).
- When the primal is a maximization, the dual is a minimization, and vice versa.

$\max \mathbf{c}\mathbf{x} :$

$$\begin{aligned} \mathbf{a}_{1*}\mathbf{x} &\leq b_1 \quad (w_1 \geq 0) \\ \mathbf{a}_{2*}\mathbf{x} &= b_2 \quad (w_2 \text{ urs}) \\ \mathbf{a}_{3*}\mathbf{x} &\geq b_3 \quad (w_3 \leq 0) \\ &\vdots \\ x_1 &\geq 0, x_2 \text{ urs}, x_3 \leq 0, \dots \end{aligned}$$

$\min \mathbf{w}\mathbf{b} :$

$$\begin{aligned} \mathbf{w}\mathbf{a}_{*1} &\geq c_1 \quad (x_1 \geq 0) \\ \mathbf{w}\mathbf{a}_{*2} &= c_2 \quad (x_2 \text{ urs}) \\ \mathbf{w}\mathbf{a}_{*3} &\leq c_3 \quad (x_3 \leq 0) \\ &\vdots \\ w_1 &\geq 0, w_2 \text{ urs}, w_3 \leq 0, \dots \end{aligned}$$

To illustrate the relationship between the primal and dual, consider this production problem we previously formulated:

**Production Problem:** You have 21 units of transparent aluminum alloy (TAA), LazWeld1, a joining robot leased for 23 hours, and CrumCut1, a cutting robot leased for 17 hours of aluminum cutting. You also have production code for a bookcase, desk, and cabinet, along with commitments to buy any of these you

can produce for \$18, \$16, and \$10 apiece, respectively. A bookcase requires 2 units of TAA, 3 hours of joining, and 1 hour of cutting, a desk requires 2 units of TAA, 2 hours of joining, and 2 hour of cutting, and a cabinet requires 1 unit of TAA, 2 hours of joining, and 1 hour of cutting. Formulate an LP to maximize your revenue given your current resources.

Decision variables:

$x_i$  : number of units of product  $i$  to produce,  
 $\forall i = \{\text{bookcase, desk, cabinet}\}$ .

$$\begin{aligned} \max z &= 18x_1 + 16x_2 + 10x_3 : \\ 2x_1 + 2x_2 + 1x_3 &\leq 21 && (\text{TAA}) \\ 3x_1 + 2x_2 + 2x_3 &\leq 23 && (\text{LazWeld1}) \\ 1x_1 + 2x_2 + 1x_3 &\leq 17 && (\text{CrumCut1}) \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

Considering the formulation above as the primal, consider a new, related, problem: You have an offer to buy all your resources (the leased hours for the two robots, and the TAA). Formulate an LP to find the minimum value of the resources given the above plans for the three products and commitments to buy them.

Decision variables:

$w_i$  : selling price, per unit, for resource  $i$ ,  $\forall i = \{\text{TAA, LazWeld1, CrumCut1}\}$ .

$$\begin{aligned} \min 21w_1 + 23w_2 + 17w_3 &: \\ 2w_1 + 3w_2 + 1w_3 &\geq 18 \\ 2w_1 + 2w_2 + 2w_3 &\geq 16 \\ 1w_1 + 2w_2 + 1w_3 &\geq 10 \\ w_1, w_2, w_3 &\geq 0. \end{aligned}$$

Define  $\mathbf{w} = \mathbf{c}_B \mathbf{B}^{-1}$  as the vector of *shadow prices*, where  $w_i$  represents the change in the objective function value caused by a unit change to the associated  $b_i$  parameter (i.e., increasing the amount of resource  $i$  by one unit, see dual objective function).

Consider the following primal tableau (where  $z_p$  is the primal objective function value) for  $(P)$  : ( $\max \{\mathbf{c}\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ )

	$z_P$	$x_i$	rhs
$z_P$	1	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - c_i$	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{b}$
BV	0	$\mathbf{B}^{-1} \mathbf{a}_i$	$\mathbf{B}^{-1} \mathbf{b}$

Observe that if a basis for  $P$  is optimal, then the row zero coefficients for the variables are greater than, or equal to, zero, that is,  $c_B B^{-1} a_i - c_i \geq 0$  for each  $x_i$  (if the variable is a slack, this simplifies to  $c_B B^{-1} \geq 0$ ).

Substituting  $w = c_B B^{-1}$  we get  $\mathbf{w}\mathbf{A} \geq \mathbf{c}, \mathbf{w} \geq 0$  which corresponds to dual feasibility.

$$(D) : \min\{\mathbf{w}\mathbf{b} : \mathbf{w}\mathbf{A} \geq \mathbf{c}, \mathbf{w} \geq 0\}.$$

### Weak Duality Property

If  $\mathbf{x}$  and  $\mathbf{w}$  are feasible solutions to  $P$  and  $D$ , respectively, then  $\mathbf{c}\mathbf{x} \leq \mathbf{w}\mathbf{A}\mathbf{x} \leq \mathbf{w}\mathbf{b}$ .

$$(P) : \max\{\mathbf{c}\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}.$$

$$(D) : \min\{\mathbf{w}\mathbf{b} : \mathbf{w}\mathbf{A} \geq \mathbf{c}, \mathbf{w} \geq 0\}.$$

This implies that the objective function value for a feasible solution to  $P$  is a lower bound on the objective function value for the optimal solution to  $D$ , and the objective function value for a feasible solution to  $D$  is an upper bound on the objective function value for the optimal solution to  $P$ .

Thus if the objective function values are equal, i.e.,  $\mathbf{c}\mathbf{x} = \mathbf{w}\mathbf{b}$ , then the solutions  $\mathbf{x}$  and  $\mathbf{w}$  are optimal.

### Fundamental Theorem of Duality

For problems  $P$  and  $D$  (i.e., any primal dual set) exactly one of the following is true:

1. Both have optimal solutions  $\mathbf{x}$  and  $\mathbf{w}$  where  $\mathbf{c}\mathbf{x} = \mathbf{w}\mathbf{b}$ .
2. One problem is unbounded (i.e., the objective function value can become arbitrarily large for a maximization, or arbitrarily small for a minimization), and the other is infeasible.
3. Both are infeasible.

#### 10.2.3.1. Optimality Conditions

---

### Farka's Lemma

Consider the following two systems:

1.  $\mathbf{A}\mathbf{x} \geq 0, \mathbf{c}\mathbf{x} < 0$ .
2.  $\mathbf{w}\mathbf{A} = \mathbf{c}, \mathbf{w} \geq 0$ .

Farka's Lemma - exactly one of these systems has a solution.

#### Suppose system 1 has $\mathbf{x}$ as a solution:

- If  $\mathbf{w}$  were a solution to system 2, then post-multiplying each side of  $\mathbf{w}\mathbf{A} = \mathbf{c}$  by  $\mathbf{x}$  would yield  $\mathbf{w}\mathbf{A}\mathbf{x} = \mathbf{c}\mathbf{x}$ .
- Since  $\mathbf{A}\mathbf{x} \geq 0$  and  $\mathbf{w} \geq 0$ , this implies that  $\mathbf{c}\mathbf{x} \geq 0$ , which violates  $\mathbf{c}\mathbf{x} < 0$ .
- Thus we show that if system 1 has a solution, system 2 cannot have one.

#### Suppose system 1 has no solution:

- Consider the following LP:  $\min\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \geq 0\}$ .
- The optimal solution is  $\mathbf{cx} = 0$  and  $\mathbf{x} = 0$ .
- The LP in standard form (substitute  $\mathbf{x} = \mathbf{x}' - \mathbf{x}''$ ,  $\mathbf{x}' \geq 0$  and  $\mathbf{x}'' \geq 0$  and add  $\mathbf{x}^s \geq 0$ ) follows:  

$$\min\{\mathbf{c}\mathbf{x}' - \mathbf{c}\mathbf{x}'' : \mathbf{Ax}' - \mathbf{Ax}'' - \mathbf{x}^s = 0, \mathbf{x}', \mathbf{x}'', \mathbf{x}^s \geq 0\}$$
- $\mathbf{x}' = 0, \mathbf{x}'' = 0, \mathbf{x}^s = 0$  is an optimal extreme point solution.
- Using  $\mathbf{x}^s$  as an initial feasible basis, solve with the simplex algorithm (with cycling prevention) to find a basis where  $\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - c_i \leq 0$  for all variables. Define  $\mathbf{w} = \mathbf{c}_B \mathbf{B}^{-1}$ .
- This yields  $\mathbf{wA} - \mathbf{c} \leq 0, -\mathbf{wA} + \mathbf{c} \leq 0, -\mathbf{w} \leq 0\}$ , from the columns for variables  $\mathbf{x}', \mathbf{x}'', \mathbf{x}^s$ , respectively. Thus,  $\mathbf{w} \geq 0$  and  $\mathbf{wA} = \mathbf{c}$ , and system 2 has a solution.

### Karush-Kuhn-Tucker (KKT) Conditions

$$(P) : \max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\}.$$

$$(D) : \min\{\mathbf{wb} : \mathbf{wA} \geq \mathbf{c}, \mathbf{w} \geq 0\}.$$

For problems  $P$  and  $D$ , with solutions  $\mathbf{x}$  and  $\mathbf{w}$ , respectively, we have the following conditions, which for LPs are necessary and sufficient conditions for optimality:

1.  $\mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0$  (primal feasibility).
2.  $\mathbf{wA} \geq \mathbf{c}, \mathbf{w} \geq 0$  (dual feasibility).
3.  $\mathbf{w}(\mathbf{Ax} - \mathbf{b}) = 0$  and  $\mathbf{x}(\mathbf{c} - \mathbf{wA}) = 0$  (complementary slackness).

Note we can rewrite the third condition as  $\mathbf{w}(\mathbf{Ax} - \mathbf{b}) = \mathbf{wx}^s = 0$  and  $\mathbf{x}(\mathbf{c} - \mathbf{wA}) = \mathbf{xw}^s = 0$ , where  $\mathbf{x}^s$  and  $\mathbf{w}^s$  are the slack variables for the primal and dual problems, respectively.

### Why do the KKT conditions hold?

Suppose that the LP  $\min\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq 0\}$  has an optimal solution  $\mathbf{x}^*$  (the dual is  $\max\{\mathbf{wb} : \mathbf{wA} \leq \mathbf{c}, \mathbf{w} \geq 0\}$ ).

- Since  $\mathbf{x}^*$  is optimal there is no direction  $\mathbf{d}$  such that  $\mathbf{c}(\mathbf{x}^* + \lambda\mathbf{d}) < \mathbf{c}\mathbf{x}^*$ ,  $\mathbf{A}(\mathbf{x}^* + \lambda\mathbf{d}) \geq \mathbf{b}$ , and  $\mathbf{x}^* + \lambda\mathbf{d} \geq 0$  for  $\lambda > 0$ .
- Let  $\mathbf{Gx} \geq \mathbf{g}$  be the binding inequalities in  $\mathbf{Ax} \geq \mathbf{b}$  and  $\mathbf{x} \geq 0$  for solution  $\mathbf{x}^*$  that is,  $\mathbf{Gx}^* = \mathbf{g}$ .
- Based on the optimality of  $\mathbf{x}^*$ , there is no direction  $\mathbf{d}$  at  $\mathbf{x}^*$  such that  $\mathbf{cd} < 0$  and  $\mathbf{Gd} \geq 0$  (else we could improve the solution).
- Based on Farka's Lemma, if the system  $\mathbf{cd} < 0, \mathbf{Gd} \geq 0$  does not have a solution, the system  $\mathbf{wG} = \mathbf{c}$ ,  $\mathbf{w} \geq 0$  must have a solution.
- $\mathbf{G}$  is composed of rows from  $\mathbf{A}$  where  $\mathbf{a}_{i*}\mathbf{x}^* = b_i$  and vectors  $\mathbf{e}_i$  for any  $x_i^* = 0$ .
- We can divide the  $\mathbf{w}$  into two sets:
  - $\{w_i, i : \mathbf{a}_{i*}\mathbf{x}^* = b_i\}$  - those corresponding to the binding functional constraints in the primal.
  - $\{w_i^s, j : x_i^* = 0\}$  - those corresponding to the binding non-negativity constraints in the primal.
- Thus  $\mathbf{G}$  has the columns  $\mathbf{a}_{i*}^T$  for  $w_i$  and  $\mathbf{e}_i^T$  for  $w_i^s$ .

- Since  $\mathbf{w}\mathbf{G} = \mathbf{c}$ ,  $\mathbf{w} \geq 0$  must have a solution, this solution is feasible for  $\mathbf{w}\mathbf{A} \leq \mathbf{c}, \mathbf{w} \geq 0$  where  $w_i^s$  are added slacks. Thus,  $\mathbf{G}$  is missing some columns from  $\mathbf{A}$  (and thus some  $w$  variables) and some slack variables if  $\mathbf{w}\mathbf{A} \leq \mathbf{c}, \mathbf{w} \geq 0$  were put into standard form, but those are not needed for feasibility based on the result, and thus can be thought of as set to zero, giving us complementary slackness.

**Example:** Consider a production LP (the primal  $P$ ) where the variables represent the amount of three products to produce, using three resources, represented by the functional constraints. In standard form  $P$  and  $D$  have  $x_4^s, x_5^s, x_6^s$  and  $w_4^s, w_5^s, w_6^s$  as slack variables, respectively.

Decision variables:

$x_i$  : number of units of product  $i$  to produce,  $\forall i = \{1, 2, 3\}$ .

$$(P) : \begin{aligned} & \max z_P = 18x_1 + 16x_2 + 10x_3 \\ & \text{s.t. } 2x_1 + 2x_2 + 1x_3 + x_4^s = 21 \quad (w_1) \\ & \quad 3x_1 + 2x_2 + 2x_3 + x_5^s = 23 \quad (w_2) \\ & \quad 1x_1 + 2x_2 + 1x_3 + x_6^s = 17 \quad (w_3) \\ & \quad x_1, x_2, x_3, x_4^s, x_5^s, x_6^s \geq 0. \end{aligned}$$

$$(D) : \begin{aligned} & \min z_D = 21w_1 + 23w_2 + 17w_3 \\ & \text{s.t. } 2w_1 + 3w_2 + 1w_3 \geq 18 \quad (x_1) \\ & \quad 2w_1 + 2w_2 + 2w_3 \geq 16 \quad (x_2) \\ & \quad 1w_1 + 2w_2 + 1w_3 \geq 10 \quad (x_3) \\ & \quad 1w_1 \geq 0 \\ & \quad 1w_2 \geq 0 \\ & \quad 1w_3 \geq 0 \\ & \quad w_1, w_2, w_3 \text{ urs.} \end{aligned}$$

Decision variables:

$w_i$  : unit selling price for resource  $i$ ,  $\forall i = \{1, 2, 3\}$ .

$$(D) : \begin{aligned} & \min z_D = 21w_1 + 23w_2 + 17w_3 : \\ & \quad 2w_1 + 3w_2 + 1w_3 - w_4^s = 18 \quad (x_1) \\ & \quad 2w_1 + 2w_2 + 2w_3 - w_5^s = 16 \quad (x_2) \\ & \quad 1w_1 + 2w_2 + 1w_3 - w_6^s = 10 \quad (x_3) \\ & \quad w_1, w_2, w_3, w_4^s, w_5^s, w_6^s \geq 0. \end{aligned}$$

The initial basic feasible tableau for the primal, i.e., having the slack variables form the basis, follows:

$P : \max$	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs
$z_P$	1	-18	-16	-10	0	0	0	0
$x_4^s$	0	2	2	1	1	0	0	21
$x_5^s$	0	3	2	2	0	1	0	23
$x_6^s$	0	1	2	1	0	0	1	17

$$x_1, x_2, x_3 = 0, x_4^s = 21, x_5^s = 23, x_6^s = 17 \quad z_P = 0$$

The following dual tableau **conforms with the primal tableau through complementary slackness**.

$D : \min$	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs
	$z_D$	-21	-23	-17	0	0	0	0
	$w_4^s$	0	-2	-3	-1	1	0	-18
	$w_5^s$	0	-2	-2	-2	0	1	-16
	$w_6^s$	0	-1	-2	-1	0	0	-10

$$w_1, w_2, w_3 = 0, w_4^s = -18, w_5^s = -16, w_6^s = -10 \quad z_D = 0$$

**Complementary slackness:**  $w_1 x_4^s = 0, w_2 x_5^s = 0, w_3 x_6^s = 0, x_1 w_4^s = 0, x_2 w_5^s = 0, x_3 w_6^s = 0.$

- If a primal variable is basic, then its corresponding dual variable must be nonbasic, and vice versa.
- The primal is suboptimal, and the dual tableau has a basic infeasible solution.
- Row 0 of the primal tableau has dual variable values in the corresponding primal variable columns.

The primal basis is not optimal, so enter  $x_1$  into the basis, and remove  $x_5^s$ , which yields:

P: Max	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs	
	$z_P$	1	0	-4	2	0	6	0	138
	$x_4^s$	0	0	2/3	-1/3	1	-2/3	0	17/3
	$x_1$	0	1	2/3	2/3	0	1/3	0	23/3
	$x_6^s$	0	0	4/3	1/3	0	-1/3	1	28/3

D: Min	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs	
	$z_D$	1	-17/3	0	-28/3	-23/3	0	0	138
	$w_2$	0	2/3	1	1/3	-1/3	0	0	6
	$w_5^s$	0	-2/3	0	-4/3	-2/3	1	0	-4
	$w_6^s$	0	1/3	0	-1/3	-2/3	0	1	2

The primal tableau does not represent an optimal basic solution, and the dual tableau does not represent a feasible basic solution.

Using Dantzig's rule, we enter  $x_2$  into the basis, and using the ratio test we find that  $x_6^s$  leaves the basis. This change in basis yields the following tableau:

P: Max	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs	
	$z_P$	1	0	0	3	0	5	3	166
	$x_4^s$	0	0	0	-1/2	1	-1/2	-1/2	1
	$x_1$	0	1	0	1/2	0	1/2	-1/2	3
	$x_2$	0	0	1	1/4	0	-1/4	3/4	7

D: Min	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs	
	$z_D$	1	-1	0	0	-3	-7	0	166
	$w_2$	0	1/2	1	0	-1/2	1/4	0	5
	$w_3$	0	1/2	0	1	1/2	-3/4	0	3
	$w_6^s$	0	1/2	0	0	-1/2	-1/4	1	3

Decision variables:

$x_i$  : number of units of product  $i$  to produce,  $\forall i = \{1, 2, 3\}$ .

$$(P) : \max z_P = 18x_1 + 16x_2 + 10x_3 : \\ 2x_1 + 2x_2 + 1x_3 + x_4^s = 21 \quad (w_1) \\ 3x_1 + 2x_2 + 2x_3 + x_5^s = 23 \quad (w_2) \\ 1x_1 + 2x_2 + 1x_3 + x_6^s = 17 \quad (w_3) \\ x_1, x_2, x_3, x_4^s, x_5^s, x_6^s \geq 0.$$

The LP  $\max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\}$  has an optimal solution  $\mathbf{x}^*$  (the dual is  $\min\{\mathbf{wb} : \mathbf{wA} \geq \mathbf{c}, \mathbf{w} \geq 0\}$ ).

- Since  $\mathbf{x}^*$  is optimal there is no direction  $\mathbf{d}$  such that  $\mathbf{c}(\mathbf{x}^* + \lambda\mathbf{d}) > \mathbf{c}\mathbf{x}^*$ ,  $\mathbf{A}(\mathbf{x}^* + \lambda\mathbf{d}) \leq \mathbf{b}$ , and  $\mathbf{x}^* + \lambda\mathbf{d} \geq 0$  for  $\lambda > 0$ .
- Let  $\mathbf{Gx} \leq \mathbf{g}$  be the binding inequalities in  $\mathbf{Ax} \leq \mathbf{b}$  and  $\mathbf{x} \geq 0$  for solution  $\mathbf{x}^*$ , that is,  $\mathbf{Gx}^* = \mathbf{g}$ .

For our example,

$$\mathbf{G}|\mathbf{g} = \left[ \begin{array}{ccc|c} 3 & 2 & 2 & 23 \\ 1 & 2 & 1 & 17 \\ 0 & 0 & -1 & 0 \end{array} \right]$$

- Based on the optimality of  $\mathbf{x}^*$ , there is no direction  $\mathbf{d}$  at  $\mathbf{x}^*$  such that  $\mathbf{cd} > 0$  and  $\mathbf{Gd} \leq 0$  (this includes  $\mathbf{d} \leq 0$ ) (else we could improve the solution).
- From Farka's Lemma, if the system  $\mathbf{cd} > 0$ ,  $\mathbf{Gd} \leq 0$  does not have a solution, the system  $\mathbf{wG} = \mathbf{c}$ ,  $\mathbf{w} \geq 0$  must have a solution.

$$3w_2 + 1w_3 = 18 \quad (x_1) \\ 2w_2 + 2w_3 = 16 \quad (x_2) \\ 2w_2 + 1w_3 - w_6^s = 10 \quad (x_3) \\ w_2, w_3, w_6^s \geq 0.$$

D: Min	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs
$z_D$	1	-1	0	0	-3	-7	0	166
$w_2$	0	1/2	1	0	-1/2	1/4	0	5
$w_3$	0	1/2	0	1	1/2	-3/4	0	3
$w_6^s$	0	1/2	0	0	-1/2	-1/4	1	3

**Challenge 1:** Solve the following LP (as represented in the tableau), using the given tableau as a starting point. Provide the details of the algorithm to do so, and make it valid for both maximization and minimization problems.

$D : \min$	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs
	$z_D$	1	-21	-23	-17	0	0	0
	$w_4^s$	0	-2	-3	-1	1	0	-18
	$w_5^s$	0	-2	-2	-2	0	1	-16
	$w_6^s$	0	-1	-2	-1	0	0	-10

**Challenge 2:** Given the following optimal tableau to our production LP, we can buy 12 units of resource 2 for \$4 a unit. Should we, please provide the analysis needed to make this decision.

$P : \max$	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs	
	$z_P$	1	0	0	3	0	5	3	166
	$x_4^s$	0	0	0	-1/2	1	-1/2	-1/2	1
	$x_1$	0	1	0	1/2	0	1/2	-1/2	3
	$x_2$	0	0	1	1/4	0	-1/4	3/4	7

## 10.2.4. Solution Algorithms

---

We start with some preliminaries, and then discuss the simplex algorithm, assuming an initial basic feasible solution, including tableau formulas. Next two extensions to the algorithm, for finding an initial basic feasible solution, are discussed. We then explore a useful algorithm for solving certain LPs that have “too many” columns.

Consider an LP  $\max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq 0\}$  in standard form, where:

- $\mathbf{A}$  is an  $m \times n$  matrix of rank  $m$  and  $n \geq m$ ;  $\mathbf{A}$  consists of  $n$  column vectors,  $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \dots, \mathbf{a}_n$ .
- $\mathbf{c}$  and  $\mathbf{x}$  are  $n$ -vectors.
- $\mathbf{b}$  is an  $m$ -vector with non-negative elements.

We can partition the problem as  $\mathbf{x} = [\mathbf{x}_B, \mathbf{x}_N]$ ,  $\mathbf{A} = [\mathbf{B}, \mathbf{N}]$ ,  $\mathbf{c} = [\mathbf{c}_B, \mathbf{c}_N]$ , where:

- $\mathbf{x}_B$  is the vector of basic variables
- $\mathbf{B}$  is the *basis matrix*, a nonsingular (i.e., it consists of  $m$  linearly independent columns of  $\mathbf{A}$ )  $m \times m$  matrix.
- $\mathbf{c}_B$  is the vector of cost coefficients for the basic variables.
- $\mathbf{x}_N$  is the vector of nonbasic variables
- $\mathbf{N}$  is the *nonbasic matrix*, a  $m \times n - m$  matrix.
- $\mathbf{c}_N$  is the vector of cost coefficients for the basic variables.

The LP can then be written as:

$$\max \{ \mathbf{c}_B \mathbf{x}_B + \mathbf{c}_N \mathbf{x}_N : \mathbf{B} \mathbf{x}_B + \mathbf{N} \mathbf{x}_N = \mathbf{b}, \mathbf{x}_B, \mathbf{x}_N \geq 0 \}.$$

For the feasible region, we can write the system of equations as follows:

$$\mathbf{B} \mathbf{x}_B + \mathbf{N} \mathbf{x}_N = \mathbf{b}.$$

$$\mathbf{B} \mathbf{x}_B = \mathbf{b} - \mathbf{N} \mathbf{x}_N.$$

Premultiplying by  $\mathbf{B}^{-1}$  yields:

$$\mathbf{x}_B = \mathbf{B}^{-1} \mathbf{b} - \mathbf{B}^{-1} \mathbf{N} \mathbf{x}_N.$$

By setting  $\mathbf{x}_N = 0$  and solving we (potentially) find a *basic feasible solution* to the system, which corresponds to an extreme point of the feasible region. Remember that the nonbasic variables  $\mathbf{x}_N = 0$  represent the defining hyperplanes for a solution.

For any set of  $m$  variables, the result can be:

1. a basic feasible solution,  $\mathbf{x}_B \geq 0$ .
2. a basic infeasible solution, some  $x \in \mathbf{x}_B \leq 0$ .
3. a set of linearly dependent columns that does not span the  $m$ -space.

For this system there are possibly  $n$  choose  $m$   $\binom{n}{m}$  basic solutions, that is, the number of basic feasible solutions is bounded by  $n! / m!(n-m)!$  from above.

#### 10.2.4.1. The Simplex Algorithm

---

It is common practice to put an LP into a tableau. To do so, we first modify the objective function by bringing all the variables to the left-hand side, yielding the following tableau of LP data:

	$z$	$x_i$	$rhs$
Row 0 ( $z$ )	1	$-c_i$	0
Rows 1-m	0	$\mathbf{a}_i$	$\mathbf{b}$

We are interested in tableaus that represent basic solutions, which have a special form; the columns of coefficients for the basis to form an identity matrix, which we can obtain using elementary row operations.

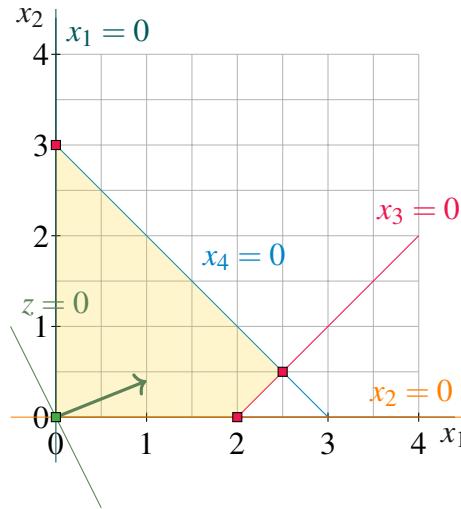
Consider the following LP:

$$\begin{aligned} \text{Max } z &= 2x_1 + x_2 \\ \text{s.t. } x_1 - x_2 + x_3 &= 2 \\ x_1 + x_2 + x_4 &= 3 \\ x_1, x_2, x_3, x_4 &\geq 0, \end{aligned}$$

where  $m=2$ ,  $n=4$ ,  $\mathbf{A} = \begin{bmatrix} 1 & -1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$ ,  
 $\mathbf{a}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ ,  $\mathbf{a}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ ,  $\mathbf{a}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ,  $\mathbf{a}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ,  
 $\mathbf{c} = [2 \ 1 \ 0 \ 0]$ ,  
 $\mathbf{x} = [x_1 \ x_2 \ x_3 \ x_4]^T$ , (T for transpose,  $\mathbf{x}$  is a column vector),  
and  $\mathbf{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ .

The tableau and graph for this LP follow:

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
( $z$ )	1	-2	-1	0	0	0
( $x_3$ )	0	1	-1	1	0	2
( $x_4$ )	0	1	1	0	1	3



Luckily, this tableau already represents a basis, which has basic variables  $\mathbf{x}_B = [x_3 \ x_4]^T$  (we can consider  $z$  as a basic variable of sorts to complete the identity matrix). Thus for this tableau we have  $\mathbf{x}_N = [x_1 \ x_2]^T$ ,  $\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ ,  $\mathbf{N} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$ . This basic solution represents an extreme point if we set the nonbasic variables to zero, as we see in the graph of the LP in the nonbasic variable space.

Here the basic variable values are  $x_3 = 2$  and  $x_4 = 3$ , and  $z = 0$ .

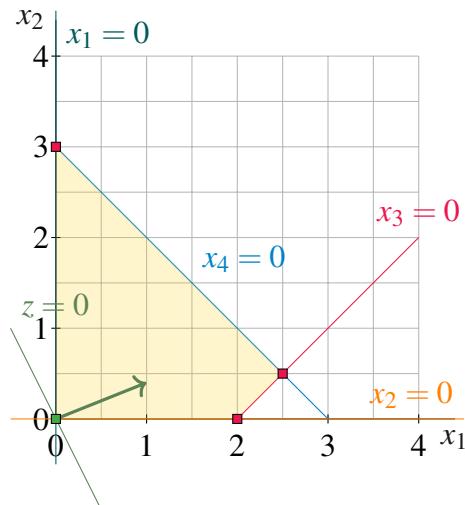
The simplex algorithm (mostly), tableau version:

1. Put the LP into a standard form tableau, find a set of basic variables that form a feasible basis, and modify the tableau to represent the basis using elementary row operations, we want the coefficients of the basic variables to form an identity matrix.
2. Check optimality, for a maximization (minimization), if the Row 0 coefficients for the nonbasic variables are all nonnegative (nonpositive) then the basis is optimal.
3. If the basis is not optimal, then find an adjacent basis that improves the solution. This will involve swapping one of the basic variables with a nonbasic variable to form a new basis.
4. Select a nonbasic entering variable using Dantzig's rule, specifically, for a maximization (minimization) problem pick the nonbasic variable with the smallest negative (largest positive) reduced cost.

5. Select a variable to leave the basis. Conceptually, as we increase the entering variable's value from zero, the values of the basis variables should change, the basic variable that goes to zero first is the leaving variable. To find the leaving variable, use the ratio test. For each row  $1-m$  having a positive coefficient in the entering variable column, divide the *rhs* by the entering variable's (positive) coefficient. The basic variable corresponding to row with the smallest ratio is the leaving variable.
6. Put the tableau into the proper form for the new basis using elementary row operations and go to Step 2.

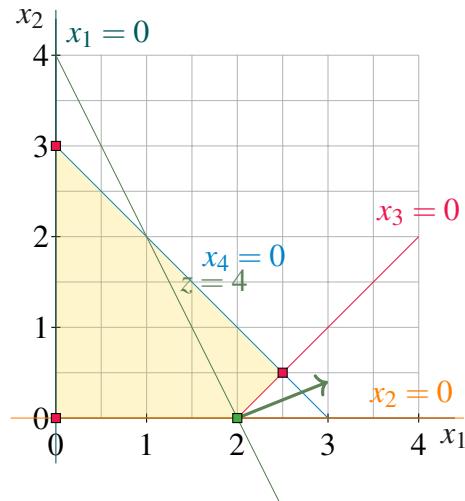
Consider the following example:

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	<i>rhs</i>
( $z$ )	1	-2	-1	0	0	0
( $x_3$ )	0	1	-1	1	0	2
( $x_4$ )	0	1	1	0	1	3

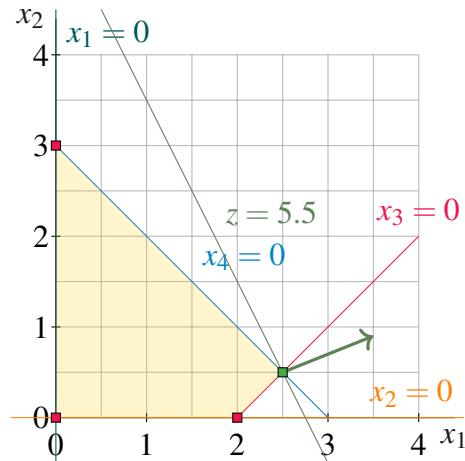


From the graph and tableau we can see that we are at the extreme point  $(0,0)$  and if we increase  $x_1$  by one unit, the objective function ( $z$ -value) increases by 2, thus improving the solution. We can increase  $x_1$  by 2 and still remain in the feasible region, moving to extreme point  $(2,0)$ . Likewise, if we increase  $x_2$  by one unit the objective function increases by 1, and we can increase  $x_2$  by three and still remain in the feasible region, moving to extreme point  $(0,3)$ . As  $x_2$  increases the basic variable  $x_3$  gets larger, while the basic variable  $x_4$  gets smaller, so  $x_2$  enters the basis and  $x_4$  leaves.

max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
$z$	1	0	-3	2	0	4
$x_1$	0	1	-1	1	0	2
$x_4$	0	0	2	-1	1	1



max	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$rhs$
$z$	1	0	0	1/2	3/2	11/2
$x_1$	0	1	0	1/2	1/2	5/2
$x_2$	0	0	1	-1/2	1/2	1/2



Let's consider this algorithm, and what we know, and see if there are any missing parts, or other information we would find valuable.

- Unique optimal solution
- Multiple optimal solutions
- Unbounded optimal objective value
- Empty feasible region (an infeasible LP)

### Tableau Formulas:

We can modify the tableau for a particular basis  $\mathbf{B}$  using the following formulas:

	$z$	$x_i$	$rhs$
( $z$ )	1	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - \mathbf{c}_i$	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{b}$
( $x_B$ )	0	$\mathbf{B}^{-1} \mathbf{a}_i$	$\mathbf{B}^{-1} \mathbf{b}$

If we partition the variables, the formulas simplify as follows:

	$z$	$x_B$	$x_N$	$rhs$
$z$	1	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{B} - \mathbf{c}_B = 0$	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{N} - \mathbf{c}_N$	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{b}$
$x_B$	0	$\mathbf{B}^{-1} \mathbf{B} = \mathbf{I}$	$\mathbf{B}^{-1} \mathbf{N}$	$\mathbf{B}^{-1} \mathbf{b}$

The formulas for the coefficients for rows 1- $m$ , that is  $\mathbf{B}^{-1} \mathbf{a}_i$  (or  $\mathbf{B}^{-1} \mathbf{A}$  for all the columns on the left-hand side) is fairly straight forward; Multiplying by  $\mathbf{B}^{-1}$  is essentially the same as doing the elementary row operations required to get an identity matrix in the basic variable columns.

Now consider the formula  $\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - \mathbf{c}_i$  for the row 0 coefficients. Where did this come from?

Consider an expanded basis matrix  $\hat{\mathbf{B}}$ , which includes the  $z$ -variable column and row, as follows:  $\begin{bmatrix} 1 & -\mathbf{c}_B \\ 0 & \mathbf{B} \end{bmatrix}$ , which yields  $\hat{\mathbf{B}}^{-1}$  of  $\begin{bmatrix} 1 & \mathbf{c}_B \mathbf{B}^{-1} \\ 0 & \mathbf{B}^{-1} \end{bmatrix}$ , and the column for  $x_i$  is  $[-c_i, \mathbf{a}_i]^T$ . Multiplying these yields  $\begin{bmatrix} 1 & \mathbf{c}_B \mathbf{B}^{-1} \\ 0 & \mathbf{B}^{-1} \end{bmatrix} \begin{bmatrix} -c_i \\ \mathbf{a}_i \end{bmatrix}$ , which results in dot product of  $[1, \mathbf{c}_B \mathbf{B}^{-1}] [-c_i, \mathbf{a}_i] = \mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - c_i$  for the first element of the resulting column vector.

For example, consider the following LP:

$$\begin{aligned} \text{Max } z &= 2x_1 + 3x_2 \\ \text{s.t. } 1x_1 + 1x_2 &\geq 2 \\ 4x_1 + 6x_2 &\leq 9 \\ x_1, x_2 &\geq 0. \end{aligned}$$

For this problem, if we have  $x_1$  ad  $x_2$  as the basic variables, then

$$\hat{\mathbf{B}} = \begin{bmatrix} 1 & -2 & -3 \\ 0 & 1 & 1 \\ 0 & 4 & 6 \end{bmatrix} \text{ and } \hat{\mathbf{B}}^{-1} = \begin{bmatrix} 1 & 0 & 1/2 \\ 0 & 3 & -1/2 \\ 0 & -2 & 1/2 \end{bmatrix},$$

and the column for  $x_1$  is  $[-2, 1, 4]^T$ .

When we multiply  $\hat{\mathbf{B}}^{-1}$  and  $[-2, 1, 4]^T$  we get

$$\left[ \begin{array}{ccccc} 1 & 0 & 1/2 & -2 & 0 \\ 0 & 3 & -1/2 & 1 & 1 \\ 0 & -2 & 1/2 & 4 & 0 \end{array} \right]$$

The simplex algorithm again:

1. Put the LP into a standard form tableau, find a feasible basis  $\mathbf{B}$  and modify the tableau using  $\mathbf{B}^{-1}$  and the tableau formulas.
2. Check optimality, if  $\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - \mathbf{c}_i \geq (\leq) 0$  for  $i : x_i \in \mathbf{x}_N$  for a maximization (minimization) problem, then the current basic solution is optimal. Stop.
3. Select an entering nonbasic variable using Dantzig's rule, specifically, entering variable  $x_i$ , where  
 $i = \min(\max)_i \{ \mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - \mathbf{c}_i < (>) 0 : (i : x_i \in \mathbf{x}_N) \}$  for a maximization (minimization) problem.
4. Select a variable to leave the basis using the ratio test. For entering variable  $x_i$  the leaving variable is the basic variable corresponding to row  $j$ , where :  
 $\min_j \{ [\mathbf{B}^{-1} \mathbf{b}]_j / [\mathbf{B}^{-1} \mathbf{a}_i]_j, (j : j = 1, \dots, m, [\mathbf{B}^{-1} \mathbf{a}_i]_j > 0) \}.$
5. Put the tableau into the proper form for the new basis and go to Step 2.

**Finding an Initial BFS** When a basic feasible solution is not apparent, we can produce one using *artificial variables*. This *artificial* basis is undesirable from the perspective of the original problem, we do not want the artificial variables in our solution, so we penalize them in the objective function, and allow the simplex algorithm to drive them to zero (if possible) and out of the basis. There are two such methods, the **Big M method** and the **Two-phase method**, which we illustrate below:

Solve the following LP using the Big M Method and the simplex algorithm:

$$\begin{aligned} \max \quad & z = 9x_1 + 6x_2 \\ \text{s.t.} \quad & 3x_1 + 3x_2 \leq 9 \\ & 2x_1 - 2x_2 \geq 3 \\ & 2x_1 + 2x_2 \geq 4 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Here is the LP is transformed into standard form by using slack variables  $x_3$ ,  $x_4$ , and  $x_5$ , with the required artificial variables  $x_6$  and  $x_7$ , which allow us to easily find an initial basic feasible solution (to the artificial

problem).

$$\begin{aligned}
 \max \quad & z_a = 9x_1 + 6x_2 - Mx_6 - Mx_7 \\
 \text{s.t.} \quad & 3x_1 + 3x_2 + x_3 = 9 \\
 & 2x_1 - 2x_2 - x_4 + x_6 = 3 \\
 & 2x_1 + 2x_2 - x_5 + x_7 = 4 \\
 & x_i \geq 0, \quad i = 1, \dots, 7.
 \end{aligned}$$

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	-9	-6	0	0	0	M	M	0	
0	3	3	1	0	0	0	0	9	
0	2	-2	0	-1	0	1	0	3	
0	2	2	0	0	-1	0	1	4	

This tableau is not in the correct form, it does not represent a basis, the columns for the artificial variables need to be adjusted.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	-9 - 4M	-6	0	M	M	0	0	-7M	
0	3	3	1	0	0	0	0	9	3
0	2	-2	0	-1	0	1	0	3	3/2
0	2	2	0	0	-1	0	1	4	2

The current solution is not optimal, so  $x_1$  enters the basis, and by the ratio test,  $x_6$  (an artificial variable) leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	-15 - 4M	0	-9/2 - M	M	9/2 + 2M	0	27/2 - M	
0	0	6	1	3/2	0	-3/2	0	3/2	3/4
0	1	-1	0	-1/2	0	1/2	0	3/2	-
0	0	4	0	1	-1	-1	1	1	1/4

The current solution is not optimal, so  $x_2$  enters the basis, and by the ratio test,  $x_7$  (an artificial variable) leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	0	0	-3/4	-15/4	-	-	17 1/4	
0	0	0	1	0	3/2	0	-3/2	3	-
0	1	0	0	-1/4	-1/4	1/2	1/4	7/4	-
0	0	1	0	1/4	-1/4	-1/4	1/4	1/4	1

The current solution is not optimal, so  $x_4$  enters the basis, and by the ratio test,  $x_2$  leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	3	0	0	-9/2	-	-	18	
0	0	0	1	0	3/2	0	-3/2	3	-
0	1	1	0	0	-1/2	0	1/2	2	-
0	0	4	0	1	-1	-1	1	1	1

The current solution is not optimal, so  $x_5$  enters the basis, and by the ratio test,  $x_3$  leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	3	3	0	0	-	-	27	
0	0	0	2/3	0	1	0	-1	2	
0	1	1	1/3	0	0	0	0	3	
0	0	4	2/3	1	0	-1	0	3	

The current solution is optimal!

Solve the following LP using the Two-phase Method and Simplex Algorithm.

$$\begin{aligned}
 & \max z = 2x_1 + 3x_2 \\
 & \text{s.t. } 3x_1 + 3x_2 \geq 6 \\
 & \quad 2x_1 - 2x_2 \leq 2 \\
 & \quad -3x_1 + 3x_2 \leq 6 \\
 & \quad x_1, x_2 \geq 0.
 \end{aligned}$$

Here is first phase LP (in standard form), where  $x_3$ ,  $x_4$ , and  $x_5$  are slack variables, and  $x_6$  is an artificial variable.

$$\begin{aligned}
 & \min z_a = x_6 \\
 & \text{s.t. } 3x_1 + 3x_2 - x_3 + x_6 = 6 \\
 & \quad 2x_1 - 2x_2 + x_4 = 2 \\
 & \quad -3x_1 + 3x_2 + x_5 = 6 \\
 & \quad x_i \geq 0, \quad i = 1, \dots, 6.
 \end{aligned}$$

Next, we put the LP into a tableau, which, still is not in the right form for our basic variables ( $x_6$ ,  $x_4$ , and  $x_5$ ).

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	0	0	0	0	0	-1	0	
0	3	3	-1	0	0	1	6	
0	2	-2	0	1	0	0	2	
0	-3	3	0	0	1	0	6	

To remedy this, we use row operation to modify the row 0 coefficients, yielding the following:

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	3	3	-1	0	0	0	6	
0	3	3	-1	0	0	1	6	2
0	2	-2	0	1	0	0	2	-
0	-3	3	0	0	1	0	6	2

The current solution is not optimal, either  $x_1$  or  $x_2$  can enter the basis, let's choose  $x_2$ . Then by the ratio test, either  $x_6$  (an artificial variable) or  $x_5$  (a slack variable) can leave the basis. Let's choose  $x_6$ .

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	0	0	0	0	0	-1	0	
0	1	1	-1/3	0	0	1/3	2	
0	4	0	-2/3	1	0	2/3	6	
0	-6	0	1	0	1	-1	0	

The current solution is optimal, so we end the first phase with a basic feasible solution to the original problem, with  $x_2$ ,  $x_4$ , and  $x_5$  as the basic variables. Now we provide a new row zero that corresponds to the original problem.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	1	0	-1	0	0	0	6	
0	1	1	-1/3	0	0	1/3	2	
0	4	0	-2/3	1	0	2/3	6	
0	-6	0	1	0	1	-1	0	

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	-5	0	0	0	1	-1	6	
0	-1	1	0	0	1/3	0	2	
0	0	0	0	1	2/3	0	6	
0	-6	0	1	0	1	-1	0	

From this tableau we can see that the LP is unbounded and an extreme point is [0, 2, 0, 6, 0] and an extreme direction is [1, 1, 6, 0, 0].

### Degeneracy and the Simplex Algorithm

Degeneracy must be considered in the simplex algorithm, as it causes some trouble. For instance, it might mislead us into thinking there are multiple optimal solutions, or provide faulty insight. Further, the algorithm as described can *cycle*, that is, remain on a degenerate extreme point repeatedly cycling through a subset of bases that represent that point, never leaving.

min	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$rhs$
	1	0	0	0	3/4	-20	1/2	-6	0
	0	1	0	0	1/4	-8	-1	9	0
	0	0	1	0	1/2	-12	-1/2	3	0
	0	0	0	1	0	0	1	0	1

Solve the following LP using the Simplex Algorithm:

$$\begin{aligned} \max \quad & z = 40x_1 + 30x_2 \\ \text{s.t.} \quad & 6x_1 + 4x_2 \leq 40 \\ & 4x_1 + 2x_2 \leq 20 \\ & x_1, x_2 \geq 0. \end{aligned}$$

By adding slack variables, we have the following tableau. Luckily, this tableau represents a basis, where  $BV=\{s_1, s_2\}$ , but by inspecting the row 0 (objective function row) coefficients, we can see that this is not optimal. By Dantzig's Rule, we enter  $x_1$  into the basis, and by the ratio test we see that  $s_2$  leaves the basis. By performing elementary row operations, we obtain the following tableau for the new basis  $BV=\{s_1, x_1\}$ .

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	-40	-30	0	0	0
0	6	4	1	0	40
0	4	2	0	1	20

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	0	-10	0	10	200
0	0	1	1	-3/2	10
0	1	1/2	0	1/4	5

This tableau is not optimal, entering  $x_2$  into the basis can improve the objective function value. The basic variables  $s_1$  and  $x_1$  tie in the ration test. If we have  $x_1$  leave the basis, we get the following tableau ( $BV=\{s_1, x_2\}$ ).

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	20	0	0	15	300
0	-2	0	1	-2	0
0	2	1	0	1/2	10

This is an optimal tableau, with an objective function value of 300, If instead of  $x_1$  leaving the basis, suppose  $s_1$  left, this would lead to the following tableau ( $BV=\{x_2, x_1\}$ ).

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	0	0	10	-5	300
0	0	1	1	-3/2	10
0	1	0	-1/2	1	0

This tableau does not look optimal, yet the objective function value is the same as the optimal solution's. This occurs because the optimal extreme point is a degenerate.

#### 10.2.4.2. Dual Simplex Algorithm

---

The dual simplex algorithm is essentially performing the simplex algorithm, on the dual problem, on the primal tableau. Remember, the Simplex algorithm, in relation to the KKT conditions, maintains primal feasibility, complementary slackness, and strives for dual feasibility. The dual simplex algorithm maintains dual feasibility, complementary slackness, and strives for primal feasibility.

1. Pick the row with the smallest  $\bar{b}_i$ , where  $\bar{b}_i < 0$  ( $\bar{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$ ), this corresponds to the leaving variable.
2. Pick a column with the minimum  $\{|z_j - c_j/y_{ij}| : y_{ij} < 0\}$ , this corresponds to the entering variable.
3. Pivot, and repeat until primal feasibility is achieved.

### 10.2.4.3. Primal-Dual Algorithm

---

The primal dual algorithm is another method for solving LPs. This algorithm starts with a feasible dual solution (not necessarily basic) and searches for a primal feasible solution will maintaining complementary slackness between the primal and dual solutions. Consider the following primal dual pair:

$$(P) : \min\{\mathbf{c}\mathbf{x} : \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq 0\}$$

$$(D) : \max\{\mathbf{w}\mathbf{b} : \mathbf{wA} \leq \mathbf{c}, \mathbf{w} \text{ urs}\}.$$

To solve  $P$  using the primal-dual algorithm, first find a feasible solution to  $D$ , it does not necessarily have to be basic, and form the following restricted primal problem:

$$(P_R) : \min\{z_R = \mathbf{1x}^a : \mathbf{a}_j x_j + \mathbf{x}^a = \mathbf{b}, x_j \geq 0, j \in Q, \mathbf{x}^a \geq 0\},$$

where  $Q = \{j : w_a j = c_j\}$ , that is,  $Q$  is the set of indexes for binding dual constraints, and  $x_j, j \in Q$  are the primal variables that can be non-zero given the current dual solution and complementary slackness. The vector  $\mathbf{x}^a$  is a vector of artificial variables; this problem looks very similar to the phase 1 problem in the two-phase method. The objective is the same, to find a basic feasible primal solution, but here we use a *restricted* or limited number of primal variables, those with indexes in set  $Q$ .

Solve  $P_R$  (using simplex) and if  $z_R = 0$ , stop, the solution is optimal for  $P$  because we have satisfied the KKT conditions, else let  $(v)^*$  be the corresponding optimal dual solution  $D_R$ .

$$D_R : \max\{\mathbf{vb} : \mathbf{va}_j \leq 0, i \in Q, \mathbf{v} \leq 1, \mathbf{v} \text{ urs}\}.$$

Note that for each  $j \in Q$ ,  $\mathbf{v}^* \mathbf{a}_j \leq 0$ . For  $i \notin Q$  calculate  $\mathbf{v}^* \mathbf{a}_i$ , if  $\mathbf{v}^* \mathbf{a}_i > 0$  then  $x_i$  can be added to the restricted primal to improve  $z_R$ . To get  $x_i$  into  $Q$  we must modify the original dual solution  $\mathbf{w}$ , first we calculate  $\theta$ .

$$\theta = \min_{i \notin Q} \{ |(\mathbf{wa}_i - c_i)| / \mathbf{v}^* \mathbf{a}_i : \mathbf{v}^* \mathbf{a}_i > 0 \} > 0$$

We take the absolute value of  $\mathbf{wa}_i - c_i$  because if the primal problem is a minimization, this term will always be non-positive, for a primal maximization this will always be non-negative (thus the absolute value is not needed).

We then replace  $\mathbf{w}$  by  $\mathbf{w} + \theta \mathbf{v}^*$ . We use this  $\theta$ -step like a ratio test, changing the current dual solution such that we can enter a new primal variable into the restricted primal, one that will improve the solution (and move us closer to feasibility), while maintaining dual feasibility and complementary slackness.

If we think about this algorithm on a tableau, we get the following formulas, where the  $z_P$  row corresponds to our dual solution, and defines the set  $Q$ , and simplex is performed on on the restricted problem (using row  $z_P^R$  for the objective function row, and using column defined by  $Q$ ).

	$z$	$\mathbf{x}$	$\mathbf{x}^s$	$\mathbf{x}^a$	rhs
$z_P$	1	$\mathbf{w}\mathbf{a}_i - c_i$	$\mathbf{w}\mathbf{a}_i$	0	0
$z_R$	1	$\mathbf{v}\mathbf{a}_i$	$\mathbf{v}\mathbf{a}_i$	$\mathbf{v}\mathbf{a}_i - 1$	0
	0	$\mathbf{B}^{-1}\mathbf{a}_i$	$\mathbf{B}^{-1}\mathbf{a}_i$	$\mathbf{B}^{-1}\mathbf{a}_i$	$\mathbf{B}^{-1}\mathbf{b}$

**Example 10.2: Primal-dual algorithm**

Consider again the following LP and solve using the primal-dual algorithm, using a starting feasible dual solution of  $\mathbf{w} = [10, 0, 0]$ .

**Solution.**

$$\begin{aligned}
 \max \quad & 18x_1 + 16x_2 + 10x_3 \\
 \text{s.t.} \quad & 2x_1 + 2x_2 + 1x_3 + x_4^s = 21 \quad (w_1) \\
 & 3x_1 + 2x_2 + 2x_3 + x_5^s = 23 \quad (w_2) \\
 & 1x_1 + 2x_2 + 1x_3 + x_6 = 17 \quad (w_3) \\
 & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0.
 \end{aligned}$$

$$\begin{aligned}
 D1 : \min \quad & 21w_1 + 23w_2 + 17w_3 \\
 \text{s.t.} \quad & 2w_1 + 3w_2 + 1w_3 \geq 18 \quad (x_1) \\
 & 2w_1 + 2w_2 + 2w_3 \geq 16 \quad (x_2) \\
 & 1w_1 + 2w_2 + 1w_3 \geq 10 \quad (x_3) \\
 & w_1, w_2, w_3 \geq 0.
 \end{aligned}$$

Use the modified tableau method, and write  $Q$  and the restricted problem for each step. Be able to explain the complete process.

For  $\mathbf{w} = [10, 0, 0]$  we have  $Q = \{3, 5, 6\}$

	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	rhs
$z$	2	4	0	10	0	0	0	210
$z_R$	0	0	0	0	0	0	-1	0
	2	2	1	1	0	0	1	21
	3	2	2	0	1	0	0	23
	1	2	1	0	0	1	0	17

We need to make a minor adjustment to the artificial variable column.

	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	rhs
$z_P$	1	2	4	0	10	0	0	0	210
$z_R$	1	2	2	1	1	0	0	0	21
	0	2	2	1	1	0	0	1	21
	0	3	2	2	0	1	0	0	23
	0	1	2	1	0	0	1	0	17

$x_3$  enters the restricted basis and  $x_5^s$  leaves the restricted basis, resulting in the following tableau:

	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	rhs
$z$	2	4	0	10	0	0	0	210
$z_R$	1/2	1	0	1	-1/2	0	0	19/2
	1/2	1	0	1	-1/2	0	1	19/2
	3/2	1	1	0	1/2	0	0	23/2
	-1/2	1	0	0	-1/2	1	0	11/2

To find  $\Theta$  we take the minimum of  $4/1$  and  $2/(1/2)$ , which are equal, and thus  $\Theta = 4$ , using this we adjust the  $z$ -row, yielding the following tableau. Notice that this operation ensures that the  $z$ -row remains non-negative. Now  $Q = \{1, 2, 3, 6\}$ .

	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	rhs
$z$	0	0	0	6	2	0	0	172
$z_R$	1/2	1	0	1	-1/2	0	0	19/2
	1/2	1	0	1	-1/2	0	1	19/2
	3/2	1	1	0	1/2	0	0	23/2
	-1/2	1	0	0	-1/2	1	0	11/2

$x_2$  enters the basis (of the restricted problem) and  $x_6$  leaves.

	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	rhs
$z$	0	0	0	6	2	0	0	172
$z_R$	1	0	0	1	0	-1	0	4
	1	0	0	1	0	-1	1	4
	2	0	1	0	1	-1	0	6
	-1/2	1	0	0	-1/2	1	0	11/2

This is not an optimal solution, so now  $x_1$  enters the restricted basis and  $x_3$  leaves.

	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	rhs
$z$	0	0	0	6	2	0	0	172
$z_R$	0	0	-1/2	1	-1/2	-1/2	0	1
	0	0	-1/2	1	-1/2	-1/2	1	1
	1	0	1/2	0	1/2	-1/2	0	3
	0	1	1/4	0	-1/4	3/4	0	7

Here  $\Theta = 6$ , which yields the following, having Now  $Q = \{1, 2, 4\}$ ; we can see that  $x_4$  enters and  $a_1$  leaves, which will change the  $z_R$ -row, this makes the restricted primal optimal, but does not change any other rows, and thus this is the optimal solution.

	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	rhs
$z$	0	0	3	0	5	3	0	166
$z_R$	0	0	-1/2	1	-1/2	-1/2	0	1
	0	0	-1/2	1	-1/2	-1/2	1	1
	1	0	1/2	0	1/2	-1/2	0	3
	0	1	1/4	0	-1/4	3/4	0	7



### 10.2.5. Sensitivity Analysis

Consider an arbitrary LP, which we will call the primal ( $P$ ):

$$(P) : \max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\},$$

where  $\mathbf{A}$  is an  $m \times n$  matrix, and  $\mathbf{x}$  is a  $n$  element column vector. Every primal LP has a related LP, which we call the dual, the dual of ( $P$ ) is:

$$(D) : \min\{\mathbf{w}\mathbf{b} : \mathbf{w}\mathbf{A} \geq \mathbf{c}, \mathbf{w} \geq 0\}.$$

If an LP has a different form from  $P$ , we can convert it to the above form to find the dual, or use the rules in the following table:

$\max \mathbf{c}\mathbf{x} :$

$$\begin{aligned} \mathbf{a}_{1*}\mathbf{x} &\leq b_1 \quad (w_1 \geq 0) \\ \mathbf{a}_{2*}\mathbf{x} &= b_2 \quad (w_2 \text{ urs}) \\ \mathbf{a}_{3*}\mathbf{x} &\geq b_3 \quad (w_3 \leq 0) \\ &\vdots \\ x_1 &\geq 0, x_2 \text{ urs}, x_3 \leq 0, \dots \end{aligned}$$

$\min \mathbf{w}\mathbf{b} :$

$$\begin{aligned} \mathbf{w}\mathbf{a}_{*1} &\geq c_1 \quad (x_1 \geq 0) \\ \mathbf{w}\mathbf{a}_{*2} &= c_2 \quad (x_2 \text{ urs}) \\ \mathbf{w}\mathbf{a}_{*3} &\leq c_3 \quad (x_3 \leq 0) \\ &\vdots \\ w_1 &\geq 0, w_2 \text{ urs}, w_3 \leq 0, \dots \end{aligned}$$

Define  $\mathbf{w} = \mathbf{c}_B \mathbf{B}^{-1}$  as the vector of *shadow prices*, where  $w_i$  represents the change in the objective function value caused by a unit change to the associated  $b_i$  parameter (i.e., increasing the amount of resource  $i$  by one unit, see dual objective function).

Some observations:

- The dual of  $D$  is  $P$ .
- Each primal constraint has an associated dual variable ( $w_i$ ) and each dual constraint has an associated primal variable ( $x_i$ ).
- When the primal is a maximization, the dual is a minimization, and vice versa.

Consider the following primal tableau (where  $z_P$  is the primal objective function value) for  $(P) : \max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ .

	$z_P$	$x_i$	rhs
$z_P$	1	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - c_i$	$\mathbf{c}_B \mathbf{B}^{-1} \mathbf{b}$
$BV$	0	$\mathbf{B}^{-1} \mathbf{a}_i$	$\mathbf{B}^{-1} \mathbf{b}$

Observe that if a basis for  $P$  is optimal, then the row zero coefficients for the variables are greater than, or equal to, zero, that is,  $c_B B^{-1} a_i - c_i \geq 0$  for each  $x_i$  (if the variable is a slack, this simplifies to  $c_B B^{-1} \geq 0$ ).

Substituting  $w = c_B B^{-1}$  we get  $wA \geq c, w \geq 0$  which corresponds to dual feasibility.

$$(D) : \min\{wb : wA \geq c, w \geq 0\}.$$

### Weak Duality Property

If  $x$  and  $w$  are feasible solutions to  $P$  and  $D$ , respectively, then  $cx \leq wAx \leq wb$ .

$$(P) : \max\{cx : Ax \leq b, x \geq 0\}.$$

$$(D) : \min\{wb : wA \geq c, w \geq 0\}.$$

This implies that the objective function value for a feasible solution to  $P$  is a lower bound on the objective function value for the optimal solution to  $D$ , and the objective function value for a feasible solution to  $D$  is an upper bound on the objective function value for the optimal solution to  $P$ .

Thus if the objective function values are equal, i.e.,  $cx = wb$ , then the solutions  $x$  and  $w$  are optimal.

### Theorem 10.3: Fundamental Theorem of Duality

or problems  $P$  and  $D$  (i.e., any primal dual set) exactly one of the following is true:

1. Both have optimal solutions  $x$  and  $w$  where  $cx = wb$ .
2. One problem is unbounded (i.e., the objective function value can become arbitrarily large for a maximization, or arbitrarily small for a minimization), and the other is infeasible.
3. Both are infeasible.

### Theorem 10.4: Farkás Lemma

Consider the following two systems:

1.  $Ax \geq 0, cx < 0$ .
2.  $wA = c, w \geq 0$ .

Exactly one of these systems has a solution.

#### Suppose system 1 has $x$ as a solution:

- If  $w$  were a solution to system 2, then post-multiplying each side of  $wA = c$  by  $x$  would yield  $wAx = cx$ .
- Since  $Ax \geq 0$  and  $w \geq 0$ , this implies that  $cx \geq 0$ , which violates  $cx < 0$ .
- Thus we show that if system 1 has a solution, system 2 cannot have one.

#### Suppose system 1 has no solution:

- Consider the following LP:  $\min\{cx : Ax \geq 0\}$ .
- The optimal solution is  $cx = 0$  and  $x = 0$ .

- The LP in standard form (substitute  $\mathbf{x} = \mathbf{x}' - \mathbf{x}''$ ,  $\mathbf{x}' \geq 0$  and  $\mathbf{x}'' \geq 0$  and add  $\mathbf{x}^s \geq 0$ ) follows:

$$\min\{\mathbf{c}\mathbf{x}' - \mathbf{c}\mathbf{x}'': \mathbf{A}\mathbf{x}' - \mathbf{A}\mathbf{x}'' - \mathbf{x}^s = 0, \mathbf{x}', \mathbf{x}'', \mathbf{x}^s \geq 0\}$$

- $\mathbf{x}' = 0, \mathbf{x}'' = 0, \mathbf{x}^s = 0$  is an optimal extreme point solution.
- Using  $\mathbf{x}^s$  as an initial feasible basis, solve with the simplex algorithm (with cycling prevention) to find a basis where  $\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - c_i \leq 0$  for all variables. Define  $\mathbf{w} = \mathbf{c}_B \mathbf{B}^{-1}$ .
- This yields  $\mathbf{w}\mathbf{A} - \mathbf{c} \leq 0, -\mathbf{w}\mathbf{A} + \mathbf{c} \leq 0, -\mathbf{w} \leq 0\}$ , from the columns for variables  $\mathbf{x}', \mathbf{x}'', \mathbf{x}^s$ , respectively. Thus,  $\mathbf{w} \geq 0$  and  $\mathbf{w}\mathbf{A} = \mathbf{c}$ , and system 2 has a solution.

### Karush-Kuhn-Tucker (KKT) Conditions

$$(P) : \max\{\mathbf{c}\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}.$$

$$(D) : \min\{\mathbf{w}\mathbf{b} : \mathbf{w}\mathbf{A} \geq \mathbf{c}, \mathbf{w} \geq 0\}.$$

For problems  $P$  and  $D$ , with solutions  $\mathbf{x}$  and  $\mathbf{w}$ , respectively, we have the following conditions, which for LPs are necessary and sufficient conditions for optimality:

- $\mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0$  (primal feasibility).
- $\mathbf{w}\mathbf{A} \geq \mathbf{c}, \mathbf{w} \geq 0$  (dual feasibility).
- $\mathbf{w}(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0$  and  $\mathbf{x}(\mathbf{c} - \mathbf{w}\mathbf{A}) = 0$  (complementary slackness).

Note we can rewrite the third condition as  $\mathbf{w}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{w}\mathbf{x}^s = 0$  and  $\mathbf{x}(\mathbf{c} - \mathbf{w}\mathbf{A}) = \mathbf{x}\mathbf{w}^s = 0$ , where  $\mathbf{x}^s$  and  $\mathbf{w}^s$  are the slack variables for the primal and dual problems, respectively.

### Why do the KKT conditions hold?

Suppose that the LP  $\min\{\mathbf{c}\mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq 0\}$  has an optimal solution  $\mathbf{x}^*$  (the dual is  $\max\{\mathbf{w}\mathbf{b} : \mathbf{w}\mathbf{A} \leq \mathbf{c}, \mathbf{w} \geq 0\}$ ).

- Since  $\mathbf{x}^*$  is optimal there is no direction  $\mathbf{d}$  such that  $\mathbf{c}(\mathbf{x}^* + \lambda\mathbf{d}) < \mathbf{c}\mathbf{x}^*$ ,  $\mathbf{A}(\mathbf{x}^* + \lambda\mathbf{d}) \geq \mathbf{b}$ , and  $\mathbf{x}^* + \lambda\mathbf{d} \geq 0$  for  $\lambda > 0$ .
- Let  $\mathbf{G}\mathbf{x} \geq \mathbf{g}$  be the binding inequalities in  $\mathbf{A}\mathbf{x} \geq \mathbf{b}$  and  $\mathbf{x} \geq 0$  for solution  $\mathbf{x}^*$  that is,  $\mathbf{G}\mathbf{x}^* = \mathbf{g}$ .
- Based on the optimality of  $\mathbf{x}^*$ , there is no direction  $\mathbf{d}$  at  $\mathbf{x}^*$  such that  $\mathbf{c}\mathbf{d} < 0$  and  $\mathbf{G}\mathbf{d} \geq 0$  (else we could improve the solution).
- Based on Farka's Lemma, if the system  $\mathbf{c}\mathbf{d} < 0, \mathbf{G}\mathbf{d} \geq 0$  does not have a solution, the system  $\mathbf{w}\mathbf{G} = \mathbf{c}$ ,  $\mathbf{w} \geq 0$  must have a solution.
- $\mathbf{G}$  is composed of rows from  $\mathbf{A}$  where  $\mathbf{a}_{i*}\mathbf{x}^* = b_i$  and vectors  $\mathbf{e}_i$  for any  $x_i^* = 0$ .
- We can divide the  $\mathbf{w}$  into two sets:
  - $\{w_i, i : \mathbf{a}_{i*}\mathbf{x}^* = b_i\}$  - those corresponding to the binding functional constraints in the primal.
  - $\{w_i^s, j : x_i^* = 0\}$  - those corresponding to the binding non-negativity constraints in the primal.
- Thus  $\mathbf{G}$  has the columns  $\mathbf{a}_{i*}^T$  for  $w_i$  and  $e_i^T$  for  $w_i^s$ .
- Since  $\mathbf{w}\mathbf{G} = \mathbf{c}$ ,  $\mathbf{w} \geq 0$  must have a solution, this solution is feasible for  $\mathbf{w}\mathbf{A} \leq \mathbf{c}, \mathbf{w} \geq 0$  where  $w_i^s$  are added slacks. Thus,  $\mathbf{G}$  is missing some columns from  $\mathbf{A}$  (and thus some  $w$  variables) and some

slack variables if  $\mathbf{wA} \leq \mathbf{c}, \mathbf{w} \geq 0$  were put into standard form, but those are not needed for feasibility based on the result, and thus can be thought of as set to zero, giving us complementary slackness.

**Example:****Example 10.5: Production LP**

Consider a production LP (the primal  $P$ ) where the variables represent the amount of three products to produce, using three resources, represented by the functional constraints. In standard form  $P$  and  $D$  have  $x_4^s, x_5^s, x_6^s$  and  $w_4^s, w_5^s, w_6^s$  as slack variables, respectively.

**Solution.** Decision variables:

$x_i$  : number of units of product  $i$  to produce,  $\forall i = \{1, 2, 3\}$ .

$$(P) : \begin{aligned} & \max z_P = 18x_1 + 16x_2 + 10x_3 \\ & \text{s.t. } 2x_1 + 2x_2 + 1x_3 + x_4^s = 21 \quad (w_1) \\ & \quad 3x_1 + 2x_2 + 2x_3 + x_5^s = 23 \quad (w_2) \\ & \quad 1x_1 + 2x_2 + 1x_3 + x_6^s = 17 \quad (w_3) \\ & \quad x_1, x_2, x_3, x_4^s, x_5^s, x_6^s \geq 0. \end{aligned}$$

$$(D) : \begin{aligned} & \min z_D = 21w_1 + 23w_2 + 17w_3 \\ & \text{s.t. } 2w_1 + 3w_2 + 1w_3 \geq 18 \quad (x_1) \\ & \quad 2w_1 + 2w_2 + 2w_3 \geq 16 \quad (x_2) \\ & \quad 1w_1 + 2w_2 + 1w_3 \geq 10 \quad (x_3) \\ & \quad 1w_1 \geq 0 \\ & \quad 1w_2 \geq 0 \\ & \quad 1w_3 \geq 0 \\ & \quad w_1, w_2, w_3 \text{ urs.} \end{aligned}$$

**Decision variables:**

$w_i$  : unit selling price for resource  $i$ ,  $\forall i = \{1, 2, 3\}$ .

$$(D) : \begin{aligned} & \min z_D = 21w_1 + 23w_2 + 17w_3 : \\ & \quad 2w_1 + 3w_2 + 1w_3 - w_4^s = 18 \quad (x_1) \\ & \quad 2w_1 + 2w_2 + 2w_3 - w_5^s = 16 \quad (x_2) \\ & \quad 1w_1 + 2w_2 + 1w_3 - w_6^s = 10 \quad (x_3) \\ & \quad w_1, w_2, w_3, w_4^s, w_5^s, w_6^s \geq 0. \end{aligned}$$

The initial basic feasible tableau for the primal, i.e., having the slack variables form the basis, follows:

$P : \max$	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs
$z_P$	1	-18	-16	-10	0	0	0	0
$x_4^s$	0	2	2	1	1	0	0	21
$x_5^s$	0	3	2	2	0	1	0	23
$x_6^s$	0	1	2	1	0	0	1	17

$$x_1, x_2, x_3 = 0, x_4^s = 21, x_5^s = 23, x_6^s = 17 z_P = 0$$

The following dual tableau **conforms with the primal tableau through complementary slackness**.

$D : \min$	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs
	$z_D$	1	-21	-23	-17	0	0	0
	$w_4^s$	0	-2	-3	-1	1	0	-18
	$w_5^s$	0	-2	-2	-2	0	1	-16
	$w_6^s$	0	-1	-2	-1	0	0	-10

$$w_1, w_2, w_3 = 0, w_4^s = -18, w_5^s = -16, w_6^s = -10 z_D = 0$$

**Complementary slackness:**  $w_1 x_4^s = 0, w_2 x_5^s = 0, w_3 x_6^s = 0, x_1 w_4^s = 0, x_2 w_5^s = 0, x_3 w_6^s = 0$ .

- If a primal variable is basic, then its corresponding dual variable must be nonbasic, and vice versa.
- The primal is suboptimal, and the dual tableau has a basic infeasible solution.
- Row 0 of the primal tableau has dual variable values in the corresponding primal variable columns.

The primal basis is not optimal, so enter  $x_1$  into the basis, and remove  $x_5^s$ , which yields:

P: Max	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs	
	$z_P$	1	0	-4	2	0	6	0	138
	$x_4^s$	0	0	2/3	-1/3	1	-2/3	0	17/3
	$x_1$	0	1	2/3	2/3	0	1/3	0	23/3
	$x_6^s$	0	0	4/3	1/3	0	-1/3	1	28/3

D: Min	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs	
	$z_D$	1	-17/3	0	-28/3	-23/3	0	0	138
	$w_2$	0	2/3	1	1/3	-1/3	0	0	6
	$w_5^s$	0	-2/3	0	-4/3	-2/3	1	0	-4
	$w_6^s$	0	1/3	0	-1/3	-2/3	0	1	2

The primal tableau does not represent an optimal basic solution, and the dual tableau does not represent a feasible basic solution.

Using Dantzig's rule, we enter  $x_2$  into the basis, and using the ratio test we find that  $x_6^s$  leaves the basis. This change in basis yields the following tableau:

P: Max	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs	
	$z_P$	1	0	0	3	0	5	3	166
	$x_4^s$	0	0	0	-1/2	1	-1/2	-1/2	1
	$x_1$	0	1	0	1/2	0	1/2	-1/2	3
	$x_2$	0	0	1	1/4	0	-1/4	3/4	7

D: Min	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs	
	$z_D$	1	-1	0	0	-3	-7	0	166
	$w_2$	0	1/2	1	0	-1/2	1/4	0	5
	$w_3$	0	1/2	0	1	1/2	-3/4	0	3
	$w_6^s$	0	1/2	0	0	-1/2	-1/4	1	3

Decision variables:

$x_i$  : number of units of product  $i$  to produce,  $\forall i = \{1, 2, 3\}$ .

$$(P) : \max z_P = 18x_1 + 16x_2 + 10x_3 :$$

$$2x_1 + 2x_2 + 1x_3 + x_4^s = 21 \quad (w_1)$$

$$3x_1 + 2x_2 + 2x_3 + x_5^s = 23 \quad (w_2)$$

$$1x_1 + 2x_2 + 1x_3 + x_6^s = 17 \quad (w_3)$$

$$x_1, x_2, x_3, x_4^s, x_5^s, x_6^s \geq 0.$$



The LP  $\max\{\mathbf{c}\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\}$  has an optimal solution  $\mathbf{x}^*$  (the dual is  $\min\{\mathbf{wb} : \mathbf{wA} \geq \mathbf{c}, \mathbf{w} \geq 0\}$ ).

- Since  $\mathbf{x}^*$  is optimal there is no direction  $\mathbf{d}$  such that  $\mathbf{c}(\mathbf{x}^* + \lambda\mathbf{d}) > \mathbf{cx}^*$ ,  $\mathbf{A}(\mathbf{x}^* + \lambda\mathbf{d}) \leq \mathbf{b}$ , and  $\mathbf{x}^* + \lambda\mathbf{d} \geq 0$  for  $\lambda > 0$ .
- Let  $\mathbf{Gx} \leq \mathbf{g}$  be the binding inequalities in  $\mathbf{Ax} \leq \mathbf{b}$  and  $\mathbf{x} \geq 0$  for solution  $\mathbf{x}^*$ , that is,  $\mathbf{Gx}^* = \mathbf{g}$ .

For our example,

$$\mathbf{G|g} = \left[ \begin{array}{ccc|c} 3 & 2 & 2 & 23 \\ 1 & 2 & 1 & 17 \\ 0 & 0 & -1 & 0 \end{array} \right]$$

- Based on the optimality of  $\mathbf{x}^*$ , there is no direction  $\mathbf{d}$  at  $\mathbf{x}^*$  such that  $\mathbf{cd} > 0$  and  $\mathbf{Gd} \leq 0$  (this includes  $\mathbf{d} \leq 0$ ) (else we could improve the solution).
- From Farka's Lemma, if the system  $\mathbf{cd} > 0$ ,  $\mathbf{Gd} \leq 0$  does not have a solution, the system  $\mathbf{wG} = \mathbf{c}$ ,  $\mathbf{w} \geq 0$  must have a solution.

$$3w_2 + 1w_3 = 18 \quad (x_1)$$

$$2w_2 + 2w_3 = 16 \quad (x_2)$$

$$2w_2 + 1w_3 - w_6^s = 10 \quad (x_3)$$

$$w_2, w_3, w_6^s \geq 0.$$

D: Min	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs
$z_D$	1	-1	0	0	-3	-7	0	166
$w_2$	0	1/2	1	0	-1/2	1/4	0	5
$w_3$	0	1/2	0	1	1/2	-3/4	0	3
$w_6^s$	0	1/2	0	0	-1/2	-1/4	1	3

**Challenge 1:** Solve the following LP (as represented in the tableau), using the given tableau as a starting point. Provide the details of the algorithm to do so, and make it valid for both maximization and minimization problems.

$D : \min$	$z_D$	$w_1$	$w_2$	$w_3$	$w_4^s$	$w_5^s$	$w_6^s$	rhs
	$z_D$	-21	-23	-17	0	0	0	0
	$w_4^s$	0	-2	-3	-1	1	0	-18
	$w_5^s$	0	-2	-2	-2	0	1	-16
	$w_6^s$	0	-1	-2	-1	0	0	-10

**Challenge 2:** Given the following optimal tableau to our production LP, we can buy 12 units of resource 2 for \$4 a unit. Should we, please provide the analysis needed to make this decision.

$P : \max$	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs	
	$z_P$	1	0	0	3	0	5	3	166
	$x_4^s$	0	0	0	-1/2	1	-1/2	-1/2	1
	$x_1$	0	1	0	1/2	0	1/2	-1/2	3
	$x_2$	0	0	1	1/4	0	-1/4	3/4	7

**Buying more of a resource:** In the example problem, we currently have 23 units of the second resource,  $b_2 = 23$ . We want to know the range of  $b_2$ -values for which the current basis remains feasible. The formula  $\mathbf{B}^{-1}\mathbf{b}$  shows us the impact of changing  $b_2$  (which only changes the *rhs* column of the tableau). So, we set  $\mathbf{B}^{-1}\mathbf{b} \geq 0$  replacing 23 with unknown  $b_2$ , and solve.

$$\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & 1/2 & -1/2 \\ 0 & -1/4 & 3/4 \end{bmatrix} \begin{bmatrix} 21 \\ b_2 \\ 17 \end{bmatrix} \geq 0 \Rightarrow 17 \leq b_2 \leq 25.$$

$P : \max$	$z_P$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs	
	$z_P$	1	0	0	3	0	5	3	166
	$x_4^s$	0	0	0	-1/2	1	-1/2	-1/2	1
	$x_1$	0	1	0	1/2	0	1/2	-1/2	3
	$x_2$	0	0	1	1/4	0	-1/4	3/4	7

If  $17 \leq b_2 \leq 25$ , then the current basis remains feasible (and optimal). The *shadow price* for this resource is 5, thus 10 is the break-even price for the two additional units of resource 2. If 4 units of resource 2 were for sale, what is the break-even price?

If we add these additional resources, and recalculate the tableau, we get the following:

$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs
1	0	0	3	0	5	3	186
0	0	0	-1/2	1	-1/2	-1/2	-1
0	1	0	1/2	0	1/2	-1/2	5
0	0	1	1/4	0	-1/4	3/4	6

This tableau looks optimal (see row zero), but the basis is infeasible. We can find a new basis that is feasible and still looks optimal using the *Dual Simplex Method*.

Here we find the current basic variable with the smallest negative *rhs* coefficient, in this case there is only one negative coefficient, and that is for  $x_4^s$ . This is the leaving variable.

To find the entering variable, use the ratio test and pivot. Note that in this case either  $x_3$  or  $x_6^s$  can enter (they tie in the ratio test), and either route leads to an optimal solution (there are multiple optimal solutions here). If we enter  $x_3$  into the basis we get the following tableau:

$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	rhs
1	0	0	0	6	2	0	180
0	0	0	1	-2	1	1	2
0	1	0	0	1	0	-1	4
0	0	1	0	1/2	-1/2	1/2	11/2

Thus, the break-even price for the 4 units of resource 2 is 14.

**Adding a new constraint:** Given an optimal basis (i.e., tableau), what does adding a new constraint do? Consider a new resource having the following constraint:

$$3/2x_1 + 3/2x_2 + 3/2x_3 \leq 14.$$

We can enter this into the current optimal tableau:

$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^s$	rhs
1	0	0	3	0	5	3	0	166
0	0	0	-1/2	1	-1/2	-1/2	0	1
0	1	0	1/2	0	1/2	-1/2	0	3
0	0	1	1/4	0	-1/4	3/4	0	7
0	3/2	3/2	3/2	0	0	0	1	14

This tableau no longer looks like one having a basic solution, so using elementary row operations, we get the following:

$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^s$	rhs
1	0	0	3	0	5	3	0	166
0	0	0	-1/2	1	-1/2	-1/2	0	1
0	1	0	1/2	0	1/2	-1/2	0	3
0	0	1	1/4	0	-1/4	3/4	0	7
0	0	0	3/8	0	-3/8	-3/8	1	-1

This tableau is no longer feasible, so using dual simplex we obtain the following ( $x_7^s$  leaves the basis and

$x_6^s$  enters).

$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^s$	rhs
1	0	0	6	0	2	0	8	158
0	0	0	-1	1	0	0	-4/3	7/3
0	1	0	0	0	1	0	-4/3	13/3
0	0	1	1	0	-1	0	2	5
0	0	0	-1	0	1	1	-8/3	8/3

What if the constraint was

$$3/2x_1 + 3/2x_2 + 3/2x_3 = 14.$$

What if the constraint was

$$3/2x_1 + 3/2x_2 + 3/2x_3 = 18.$$

Using  $x_7$  as an artificial variable, we get the following tableau (an additional row labeled  $z_a$  is added for the phase 1 problem).

	$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	RHS
$z$	1	0	0	3	0	5	3	0	166
$z_a$	0	0	0	0	0	0	0	-1	0
$x_4$	0	0	0	-1/2	1	-1/2	-1/2	0	1
$x_1$	0	1	0	1/2	0	1/2	-1/2	0	3
$x_2$	0	0	1	1/4	0	-1/4	3/4	0	7
$x_7$	0	0	0	3/8	0	-3/8	-3/8	1	3

Adjusting this tableau to represent the initial, artificial, basis yields:

	$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7^a$	RHS
$z$	1	0	0	3	0	5	3	0	166
$z_a$	0	0	0	3/8	0	-3/8	-3/8	0	3
$x_4$	0	0	0	-1/2	1	-1/2	-1/2	0	1
$x_1$	0	1	0	1/2	0	1/2	-1/2	0	3
$x_2$	0	0	1	1/4	0	-1/4	3/4	0	7
$x_7$	0	0	0	3/8	0	-3/8	-3/8	1	3

Since the phase 1 problem is a minimization, we enter  $x_3$  into the basis, and remove  $x_1$ , this yields an optimal solution to the phase 1 problem having the artificial variable  $x_7$  in the basis, thus with the new constraint, the LP is infeasible.

**Adding a new variable:** Consider the following tableau, to which a new column has been added (corresponding to  $x_7$ ), given this column, please find the original data for this variable, i.e., the values of  $c_7$  and  $\mathbf{a}_7$ .

$z$	$x_1$	$x_2$	$x_3$	$x_4^s$	$x_5^s$	$x_6^s$	$x_7$	RHS
1	0	0	3	0	5	3	-1	166
0	0	0	-1/2	1	-1/2	-1/2	0	1
0	1	0	1/2	0	1/2	-1/2	0	3
0	0	1	1/4	0	-1/4	3/4	1/2	7

## 10.2.6. Theory Applications

---

### Cutting Stock - Column Generation

Consider the Cutting Stock Problem, which we use to illustrate column generation:

Given a stock board of length  $q$  and demand  $d_i$  for boards of length  $l_i$  (where  $l_i \leq q$ ), you must cut the stock boards to satisfy this demand, while minimizing waste, i.e., the number of stock boards required to satisfy the demand.

Problem parameters:

$P$  set of cutting pattern indexes,  $\{1, 2, \dots, n\}$ .

$L$  set of board length indexes,  $\{1, 2, \dots, m\}$ .

$d_i$  demand for boards of length  $l_i$ ,  $i = 1, \dots, m$ .

$a_{ij}$  number of boards of length  $l_i$ , obtained when one stock board is cut using pattern  $j$ ,  $i \in L$ ,  $j \in P$ .

Decision variable:

$x_j$  number of stock boards to cut using pattern  $j \in P$ .

$$\begin{aligned} \text{Min } z &= \sum_{j \in P} x_j \\ \text{s.t. } \sum_{j \in P} a_{ij} x_j &\geq d_i, \forall i \in L \\ x_j &\geq 0, \forall j \in P. \end{aligned}$$

It can be difficult to enumerate all possible cutting patterns  $\mathbf{a}_j$  (this set can be quite large).

Instead, solve a restricted problem (as follows) where  $P_R$  is a subset of  $P$  that provides a feasible solution:

$$\begin{aligned} \text{Min } z &= \sum_{j \in P_R} x_j \\ \text{s.t. } \sum_{j \in P_R} a_{ij} x_j &\geq d_i, \forall i \in L \\ x_j &\geq 0, \forall j \in P_R. \end{aligned}$$

The optimal solution to the restricted problem is a feasible solution to the full problem. We want to find a new cutting pattern that will allow us to improve the restricted problem solution. Recall that the optimality condition for a minimization is  $\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - \mathbf{c}_i \leq 0$ , thus to improve the restricted problem we want a column defined by  $\mathbf{a}_i$  such that  $\mathbf{c}_B \mathbf{B}^{-1} \mathbf{a}_i - \mathbf{c}_i > 0$ . For this problem  $c_i = 1, i \in P$ .

To find this vector  $\mathbf{a}_i$  (a column in the simplex tableau and a cutting pattern), we use the optimal solution

to the restricted primal, which defines  $\mathbf{c}_B \mathbf{B}^{-1}$ , we can solve the following integer program:

$$\begin{aligned} \text{Max } & \sum_{i \in L} \mathbf{c}_B \mathbf{B}^{-1} a_i \\ \text{s.t. } & \sum_{i \in L} l_i a_i \leq q, \\ & a_i \in \mathbb{Z}^{\geq 0}, \forall i \in L, \end{aligned}$$

where the  $a_i$ 's are the decision variables. This produces a new cutting pattern, which is then added to the restricted problem. This process continues until the sub-problem provides an optimal solution of zero.

### **Revenue Management - Shadow Prices**

Consider an airline with a hub-and-spoke route structure. We define a flight as one take-off and landing of an aircraft at a particular time, flights are usually given flight numbers. Ticket prices are based on the itinerary, where an itinerary is a specific flight or set of (connecting) flights and a booking class (reflected to rules for the ticket, e.g., refundability). The following figure illustrates seven possible combinations that can be used to build itineraries using connections through airport B, the hub (A-B, B-C, B-D, B-E, A-B-C, A-B-D, A-B-E).

The airline forecasts demand for all important itinerary. Here are the problem parameters.

Problem Parameters:

$F$  set of all flights (one take-off and landing of an aircraft, at a specific time).

$c_f$  capacity of flight  $f$ ,  $\forall f \in F$ .

$I$  set of all itineraries (set of flights that customer uses) and booking class.

$I_f$  set of all itineraries on flight  $f$ ,  $\forall f \in F$ .

$d_i$  demand for itinerary  $i$ ,  $\forall i \in I$ .

$f_i$  fare for itinerary  $i$ ,  $\forall i \in I$ .

Decision Variables:

$x_i$  # of passengers accepted for itinerary  $i$ ,  $\forall i \in I$ .

The following linear program maximizes the revenue:

$$\begin{aligned} \text{Max } & \sum_{i \in I} f_i x_i \\ \text{s.t.: } & x_i \leq d_i, \quad \forall i \in I \\ & \sum_{i \in I_f} x_i \leq c_f, \quad \forall f \in F \\ & x_i \geq 0, \quad \forall i \in I \end{aligned}$$

## 10.3 Other material for Integer Linear Programming

---

Recall the problem on lemonade and lemon juice from Chapter 9.1:

**Problem.** Say you are a vendor of lemonade and lemon juice. Each unit of lemonade requires 1 lemon and 2 litres of water. Each unit of lemon juice requires 3 lemons and 1 litre of water. Each unit of lemonade gives a profit of \$3. Each unit of lemon juice gives a profit of \$2. You have 6 lemons and 4 litres of water available. How many units of lemonade and lemon juice should you make to maximize profit?

Letting  $x$  denote the number of units of lemonade to be made and letting  $y$  denote the number of units of lemon juice to be made, the problem could be formulated as the following linear programming problem:

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & x + 3y \leq 6 \\ & 2x + y \leq 4 \\ & x \geq 0 \\ & y \geq 0. \end{aligned}$$

The problem has a unique optimal solution at  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.6 \end{bmatrix}$  for a profit of 6.8. But this solution requires us to make fractional units of lemonade and lemon juice. What if we require the number of units to be integers? In other words, we want to solve

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & x + 3y \leq 6 \\ & 2x + y \leq 4 \\ & x \geq 0 \\ & y \geq 0 \\ & x, y \in \mathbb{Z}. \end{aligned}$$

This problem is no longer a linear programming problem. But rather, it is an integer linear programming problem.

A **mixed-integer linear programming problem** is a problem of minimizing or maximizing a linear function subject to finitely many linear constraints such that the number of variables are finite and at least one of which is required to take on integer values.

If all the variables are required to take on integer values, the problem is called a **pure integer linear programming problem** or simply an **integer linear programming problem**. Normally, we assume the problem data to be rational numbers to rule out some pathological cases.

Mixed-integer linear programming problems are in general difficult to solve yet they are too important to ignore because they have a wide range of applications (e.g. transportation planning, crew scheduling, circuit design, resource management etc.) Many solution methods for these problems have been devised and some of them first solve the **linear programming relaxation** of the original problem, which is the problem obtained from the original problem by dropping all the integer requirements on the variables.

**Example 10.6**

Let (MP) denote the following mixed-integer linear programming problem:

$$\begin{array}{lllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 & + & x_2 & + & x_3 \geq 1 \\ & -x_1 & - & x_2 & + & 2x_3 \geq 0 \\ & -x_1 & + & 5x_2 & - & x_3 = 3 \\ & x_1, & x_2, & x_3 & \geq 0 \\ & & & x_3 & \in \mathbb{Z}. \end{array}$$

The linear programming relaxation of (MP) is:

$$\begin{array}{lllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 & + & x_2 & + & x_3 \geq 1 \\ & -x_1 & - & x_2 & + & 2x_3 \geq 0 \\ & -x_1 & + & 5x_2 & - & x_3 = 3 \\ & x_1, & x_2, & x_3 & \geq 0. \end{array}$$

Let (P1) denote the linear programming relaxation of (MP). Observe that the optimal value of (P1) is a lower bound for the optimal value of (MP) since the feasible region of (P1) contains all the feasible solutions to (MP), thus making it possible to find a feasible solution to (P1) with objective function value better than the optimal value of (MP). Hence, if an optimal solution to the linear programming relaxation happens to be a feasible solution to the original problem, then it is also an optimal solution to the original problem. Otherwise, there is an integer variable having a nonintegral value  $v$ . What we then do is to create two new subproblems as follows: one requiring the variable to be at most the greatest integer less than  $v$ , the other requiring the variable to be at least the smallest integer greater than  $v$ . This is the basic idea behind the **branch-and-bound method**. We now illustrate these ideas on (MP).

Solving the linear programming relaxation (P1), we find that  $\mathbf{x}' = \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{1}{3} \end{bmatrix}$  is an optimal solution to (P1). Note

that  $\mathbf{x}'$  is not a feasible solution to (MP) because  $x'_3$  is not an integer. We now create two subproblems (P2) and (P3) such that (P2) is obtained from (P1) by adding the constraint  $x_3 \leq \lfloor x'_3 \rfloor$  and (P3) is obtained from (P1) by adding the constraint  $x_3 \geq \lceil x'_3 \rceil$ . (For a number  $a$ ,  $\lfloor a \rfloor$  denotes the greatest integer at most  $a$  and  $\lceil a \rceil$  denotes the smallest integer at least  $a$ .) Hence, (P2) is the problem

$$\begin{array}{lllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 & + & x_2 & + & x_3 \geq 1 \\ & -x_1 & - & x_2 & + & 2x_3 \geq 0 \\ & -x_1 & + & 5x_2 & - & x_3 = 3 \\ & & & x_3 & \leq 0 \\ & x_1, & x_2, & x_3 & \geq 0, \end{array}$$

and (P3) is the problem

$$\begin{array}{lllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 + x_2 + x_3 & \geq & 1 \\ & -x_1 - x_2 + 2x_3 & \geq & 0 \\ & -x_1 + 5x_2 - x_3 & = & 3 \\ & & x_3 & \geq & 1 \\ & x_1, x_2, x_3 & \geq & 0. \end{array}$$

Note that any feasible solution to (MP) must be a feasible solution to either (P2) or (P3). Using the help of a solver, one sees that (P2) is infeasible. The problem (P3) has an optimal solution at  $\mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{4}{5} \\ 1 \end{bmatrix}$ , which

is also feasible to (MP). Hence,  $\mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{4}{5} \\ 1 \end{bmatrix}$  is an optimal solution to (MP).

We now give a description of the method for a general mixed-integer linear programming problem (MIP). Suppose that (MIP) is a minimization problem and has  $n$  variables  $x_1, \dots, x_n$ . Let  $\mathcal{I} \subseteq \{1, \dots, n\}$  denote the set of indices  $i$  such that  $x_i$  is required to be an integer in (MIP).

### Branch-and-bound method

**Input:** The problem (MIP).

**Steps:**

1. Set  $\text{bestbound} := \infty$ ,  $\mathbf{x}_{\text{best}}^* := \text{N/A}$ ,  $\text{activeproblems} := \{(LP)\}$  where  $(LP)$  denotes the linear programming relaxation of (MIP).
2. If there is no problem in  $\text{activeproblems}$ , then stop; if  $\mathbf{x}_{\text{best}}^* \neq \text{N/A}$ , then  $\mathbf{x}_{\text{best}}^*$  is an optimal solution; otherwise, (MIP) has no optimal solution.
3. Select a problem  $P$  from  $\text{activeproblems}$  and remove it from  $\text{activeproblems}$ .
4. Solve  $P$ .
  - If  $P$  is unbounded, then stop and conclude that (MIP) does not have an optimal solution.
  - If  $P$  is infeasible, go to step 2.
  - If  $P$  has an optimal solution  $\mathbf{x}^*$ , then let  $z$  denote the objective function value of  $\mathbf{x}^*$ .
5. If  $z \geq \text{bestbound}$ , go to step 2.
6. If  $x_i^*$  is not an integer for some  $i \in \mathcal{I}$ , then create two subproblems  $P_1$  and  $P_2$  such that  $P_1$  is the problem obtained from  $P$  by adding the constraint  $x_i \leq \lfloor x_i^* \rfloor$  and  $P_2$  is the problem obtained from  $P$  by adding the constraint  $x_i \geq \lceil x_i^* \rceil$ . Add the problems  $P_1$  and  $P_2$  to  $\text{activeproblems}$  and go to step 2.
7. Set  $\mathbf{x}_{\text{best}}^* = \mathbf{x}^*$ ,  $\text{bestbound} = z$  and go to step 2.

### Remarks.

- Throughout the algorithm,  $\text{activeproblems}$  is a set of subproblems remained to be solved. Note that for each problem  $P$  in  $\text{activeproblems}$ ,  $P$  is a linear programming problem and that any feasible solution to  $P$  satisfying the integrality requirements is a feasible solution to (MIP).
- $\mathbf{x}_{\text{best}}^*$  is the feasible solution to (MIP) that has the best objective function value found so far and

**bestbound** is its objective function value. It is often called an **incumbent**.

- In practice, how a problem from **activeproblems** is selected in step 3 has an impact on the overall performance. However, there is no general rule for selection that guarantees good performance all the time.
- In step 5, the problem  $P$  is discarded since it cannot contain any feasible solution to (MIP) having a better objective function value than  $x_{\text{best}}^*$ .
- If step 7 is reached, then  $x^*$  is a feasible solution to (MIP) having objective function value better than **bestbound**. So it becomes the current best solution.
- It is possible for the algorithm to never terminate. Below is an example for which the algorithm will never stop:

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & x_1 + 2x_2 - 2x_3 = 1 \\ & x_1, x_2, x_3 \geq 0 \\ & x_1, x_2, x_3 \in \mathbb{Z}. \end{aligned}$$

However, it is easy to see that  $\mathbf{x}^* = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$  is an optimal solution because there is no feasible solution with  $x_1 = 0$ .

One way to keep track of the progress of the computations is to set up a progress chart with the following headings:

Iter	solved	status	branching	activeproblems	$\mathbf{x}_{\text{best}}^*$	bestbound
------	--------	--------	-----------	----------------	------------------------------	-----------

In a given iteration, the entry in the **solved** column denotes the subproblem that has been solved with the result in the **status** column. The **branching** column indicates the subproblems created from the solved subproblem with an optimal solution not feasible to (MIP). The entries in the remaining columns contain the latest information in the given iteration. For the example (MP) above, the chart could look like the following:

Iter	solved	status	branching	activeproblems	$\mathbf{x}_{\text{best}}^*$	bestbound
1	(P1)	optimal $\mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{1}{3} \end{bmatrix}$	(P2): $x_3 \leq 0$ , (P3): $x_3 \geq 1$	(P2), (P3)	N/A	$\infty$
2	(P2)	infeasible	—	(P3)	N/A $\begin{bmatrix} 0 \\ \frac{4}{5} \\ 1 \end{bmatrix}$	$\infty$
3	(P3)	optimal $\mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{4}{5} \\ 1 \end{bmatrix}$	—	—	$\begin{bmatrix} 0 \\ \frac{4}{5} \\ 1 \end{bmatrix}$	1

## Exercises

---

1. Suppose that (MP) in Example 20.8 above has  $x_2$  required to be an integer as well. Continue with the computations and determine an optimal solution to the modified problem.
2. With the help of a solver, determine the optimal value of

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & x + 3y \leq 6 \\ & 2x + y \leq 4 \\ & x, y \geq 0 \\ & x, y \in \mathbb{Z}. \end{aligned}$$

3. Let  $\mathbf{A} \in \mathbb{Q}^{m \times n}$  and  $\mathbf{b} \in \mathbb{Q}^m$ . Let  $S$  denote the system

$$\begin{aligned} \mathbf{Ax} &\geq \mathbf{b} \\ \mathbf{x} &\in \mathbb{Z}^n \end{aligned}$$

- a. Suppose that  $\mathbf{d} \in \mathbb{Q}^m$  satisfies  $\mathbf{d} \geq 0$  and  $\mathbf{d}^T \mathbf{A} \in \mathbb{Z}^n$ . Prove that every  $\mathbf{x}$  satisfying  $S$  also satisfies  $\mathbf{d}^T \mathbf{Ax} \geq \lceil \mathbf{d}^T \mathbf{b} \rceil$ . (This inequality is known as a **Chvátal-Gomory cutting plane**.)
- b. Suppose that  $\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 5 & 3 \\ 7 & 6 \end{bmatrix}$  and  $\mathbf{b} = \begin{bmatrix} 2 \\ 1 \\ 8 \end{bmatrix}$ . Show that every  $\mathbf{x}$  satisfying  $S$  also satisfies  $x_1 + x_2 \geq 2$ .

## Solutions

---

1. An optimal solution to the modified problem is given by  $x^* = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ .
2. An optimal solution is  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ . Thus, the optimal value is 6.
3. a. Since  $\mathbf{d} \geq 0$  and  $\mathbf{Ax} \geq \mathbf{b}$ , we have  $\mathbf{d}^T \mathbf{Ax} \geq \mathbf{d}^T \mathbf{b}$ . If  $\mathbf{d}^T \mathbf{b}$  is an integer, the result follows immediately. Otherwise, note that  $\mathbf{d}^T \mathbf{A} \in \mathbb{Z}^n$  and  $\mathbf{x} \in \mathbb{Z}^n$  imply that  $\mathbf{d}^T \mathbf{Ax}$  is an integer. Thus,  $\mathbf{d}^T \mathbf{Ax}$  must be greater than or equal to the least integer greater than  $\mathbf{d}^T \mathbf{b}$ .
- b. Take  $\mathbf{d} = \begin{bmatrix} \frac{1}{9} \\ 0 \\ \frac{1}{9} \end{bmatrix}$  and apply the result in the previous part.

# 11. Simplex Method

## Definition 11.1: Standard Form

A linear program is in standard form if it is written as

$$\begin{aligned} & \max c^\top x \\ & \text{s.t. } Ax = b \\ & \quad x \geq 0. \end{aligned}$$

## Definition 11.2: Extreme Point

A point  $x$  in a convex set  $C$  is called an extreme point if it cannot be written as a strict convex combination of other points in  $C$ .

## Theorem 11.3: Optimal Extreme Point - Bounded Case

Consider a linear optimization problem in standard form. Suppose that the feasible region is bounded and non-empty.

Then there exists an optimal solution at an extreme point of the feasible region.

**Proof.** [Proof Sketch]



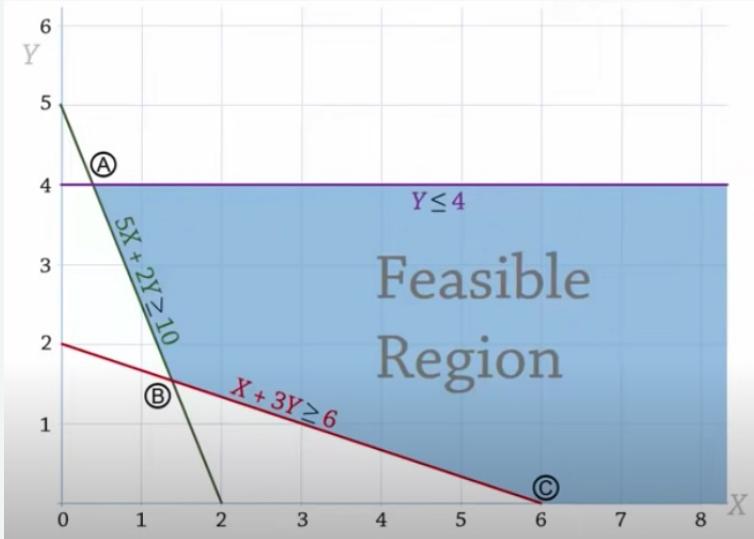
## Definition 11.4: Basic solution

A basic solution to  $Ax = b$  is obtained by setting  $n - m$  variables equal to 0 and solving for the values of the remaining  $m$  variables. This assumes that the setting  $n - m$  variables equal to 0 yields unique values for the remaining  $m$  variables or, equivalently, the columns of the remaining  $m$  variables are linearly independent.

**Example 11.5**

Consider the problem

$$\begin{aligned} \max \quad & Z = -5X - 7Y \\ \text{s.t.} \quad & X + 3Y \geq 6 \\ & 5X + 2Y \geq 10 \\ & Y \leq 4 \\ & X, Y \geq 0 \end{aligned}$$



We begin by converting this problem to standard form.

$$\begin{aligned} \max \quad & Z = -5X - 7Y + 0s_1 + 0s_2 + 0s_3 \\ \text{s.t.} \quad & X + 3Y - s_1 = 6 \\ & 5X + 2Y - s_2 = 10 \\ & Y + s_3 = 4 \\ & X, Y, s_1, s_2, s_3 \geq 0 \end{aligned}$$

Thus, we can write this problem in matrix form with

$$\max \begin{bmatrix} -5 \\ -7 \\ 0 \\ 0 \\ 0 \end{bmatrix}^\top \begin{bmatrix} X \\ Y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad (11.1)$$

$$\begin{bmatrix} 1 & 3 & -1 & 0 & 0 \\ 5 & 2 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 10 \\ 4 \end{bmatrix} \quad (11.2)$$

$$(X, Y, s_1, s_2, s_3) \geq 0 \quad (11.3)$$

### Definition 11.6: Basic feasible solution

Any basic solution in which all the variables are non-negative is a basic feasible solution.

### Theorem 11.7: BFS iff extreme

A point in the feasible region of an LP is an extreme point if and only if it is a basic feasible solution to the LP.

To prove this theorem, we

### Theorem 11.8: Representation

Consider an LP in standard form, having bfs  $b_1, \dots, b_k$ . Any point  $x$  in the LP's feasible region may be written in the form

$$x = d + \sum_{i=1}^k \sigma_i b_i$$

where  $d$  is 0 or a direction of unboundedness and  $\sum_{i=1}^k \sigma_i = 1$  and  $\sigma_i \geq 0$ .

### Theorem 11.9: Optimal bfs

If an LP has an optimal solution, then it has an optimal bfs.

**Proof.** Let  $x$  be an optimal solution. Then

$$x = d + \sum_{i=1}^k \sigma_i b_i$$

where  $d$  is 0 or a direction of unboundedness.

- If  $c^\top d > 0$ , the  $x' = \lambda d + \sum_{i=1}^k \sigma_i b_i$  has bigger objective value for  $|\lambda| > 1$ , which is a contradiction since  $x$  was optimal.
- If  $c^\top d < 0$ , the  $x'' = \sum_{i=1}^k \sigma_i b_i$  has a bigger objective value, which is a contradiction since  $x$  was optimal.

Thus, we conclude that  $c^\top d = 0$ .

Since

$$c^\top x \geq c^\top b_i$$

for all  $i = 1, \dots, k$ , we can conclude that

$$c^\top x = c^\top b_i$$

for all  $i$  such that  $\sigma_i > 0$ . Hence, there exists an optimal basic feasible solution. ♠

## 11.1 Simplex Method

---

## 11.2 Finding Feasible Basis

---

**Finding an Initial BFS** When a basic feasible solution is not apparent, we can produce one using *artificial variables*. This *artificial* basis is undesirable from the perspective of the original problem, we do not want the artificial variables in our solution, so we penalize them in the objective function, and allow the simplex algorithm to drive them to zero (if possible) and out of the basis. There are two such methods, the **Big M method** and the **Two-phase method**, which we illustrate below:

Solve the following LP using the Big M Method and the simplex algorithm:

$$\begin{aligned} \max \quad & z = 9x_1 + 6x_2 \\ \text{s.t.} \quad & 3x_1 + 3x_2 \leq 9 \\ & 2x_1 - 2x_2 \geq 3 \\ & 2x_1 + 2x_2 \geq 4 \\ & x_1, x_2 \geq 0. \end{aligned}$$

Here is the LP is transformed into standard form by using slack variables  $x_3$ ,  $x_4$ , and  $x_5$ , with the required artificial variables  $x_6$  and  $x_7$ , which allow us to easily find an initial basic feasible solution (to the artificial

problem).

$$\begin{aligned}
 \max \quad & z_a = 9x_1 + 6x_2 - Mx_6 - Mx_7 \\
 \text{s.t.} \quad & 3x_1 + 3x_2 + x_3 = 9 \\
 & 2x_1 - 2x_2 - x_4 + x_6 = 3 \\
 & 2x_1 + 2x_2 - x_5 + x_7 = 4 \\
 & x_i \geq 0, \quad i = 1, \dots, 7.
 \end{aligned}$$

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	-9	-6	0	0	0	M	M	0	
0	3	3	1	0	0	0	0	9	
0	2	-2	0	-1	0	1	0	3	
0	2	2	0	0	-1	0	1	4	

This tableau is not in the correct form, it does not represent a basis, the columns for the artificial variables need to be adjusted.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	-9 - 4M	-6	0	M	M	0	0	-7M	
0	3	3	1	0	0	0	0	9	3
0	2	-2	0	-1	0	1	0	3	3/2
0	2	2	0	0	-1	0	1	4	2

The current solution is not optimal, so  $x_1$  enters the basis, and by the ratio test,  $x_6$  (an artificial variable) leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	-15 - 4M	0	-9/2 - M	M	9/2 + 2M	0	27/2 - M	
0	0	6	1	3/2	0	-3/2	0	3/2	3/4
0	1	-1	0	-1/2	0	1/2	0	3/2	-
0	0	4	0	1	-1	-1	1	1	1/4

The current solution is not optimal, so  $x_2$  enters the basis, and by the ratio test,  $x_7$  (an artificial variable) leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	0	0	-3/4	-15/4	-	-	17 1/4	
0	0	0	1	0	3/2	0	-3/2	3	-
0	1	0	0	-1/4	-1/4	1/2	1/4	7/4	-
0	0	1	0	1/4	-1/4	-1/4	1/4	1/4	1

The current solution is not optimal, so  $x_4$  enters the basis, and by the ratio test,  $x_2$  leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	3	0	0	-9/2	-	-	18	
0	0	0	1	0	3/2	0	-3/2	3	-
0	1	1	0	0	-1/2	0	1/2	2	-
0	0	4	0	1	-1	-1	1	1	1

The current solution is not optimal, so  $x_5$  enters the basis, and by the ratio test,  $x_3$  leaves the basis.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	RHS	ratio
1	0	3	3	0	0	-	-	27	
0	0	0	2/3	0	1	0	-1	2	
0	1	1	1/3	0	0	0	0	3	
0	0	4	2/3	1	0	-1	0	3	

The current solution is optimal!

Solve the following LP using the Two-phase Method and Simplex Algorithm.

$$\begin{aligned}
 & \max z = 2x_1 + 3x_2 \\
 & \text{s.t. } 3x_1 + 3x_2 \geq 6 \\
 & \quad 2x_1 - 2x_2 \leq 2 \\
 & \quad -3x_1 + 3x_2 \leq 6 \\
 & \quad x_1, x_2 \geq 0.
 \end{aligned}$$

Here is first phase LP (in standard form), where  $x_3$ ,  $x_4$ , and  $x_5$  are slack variables, and  $x_6$  is an artificial variable.

$$\begin{aligned}
 & \min z_a = x_6 \\
 & \text{s.t. } 3x_1 + 3x_2 - x_3 + x_6 = 6 \\
 & \quad 2x_1 - 2x_2 + x_4 = 2 \\
 & \quad -3x_1 + 3x_2 + x_5 = 6 \\
 & \quad x_i \geq 0, \quad i = 1, \dots, 6.
 \end{aligned}$$

Next, we put the LP into a tableau, which, still is not in the right form for our basic variables ( $x_6$ ,  $x_4$ , and  $x_5$ ).

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	0	0	0	0	0	-1	0	
0	3	3	-1	0	0	1	6	
0	2	-2	0	1	0	0	2	
0	-3	3	0	0	1	0	6	

To remedy this, we use row operation to modify the row 0 coefficients, yielding the following:

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	3	3	-1	0	0	0	6	
0	3	3	-1	0	0	1	6	2
0	2	-2	0	1	0	0	2	-
0	-3	3	0	0	1	0	6	2

The current solution is not optimal, either  $x_1$  or  $x_2$  can enter the basis, let's choose  $x_2$ . Then by the ratio test, either  $x_6$  (an artificial variable) or  $x_5$  (a slack variable) can leave the basis. Let's choose  $x_6$ .

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	0	0	0	0	0	-1	0	
0	1	1	-1/3	0	0	1/3	2	
0	4	0	-2/3	1	0	2/3	6	
0	-6	0	1	0	1	-1	0	

The current solution is optimal, so we end the first phase with a basic feasible solution to the original problem, with  $x_2$ ,  $x_4$ , and  $x_5$  as the basic variables. Now we provide a new row zero that corresponds to the original problem.

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	1	0	-1	0	0	0	6	
0	1	1	-1/3	0	0	1/3	2	
0	4	0	-2/3	1	0	2/3	6	
0	-6	0	1	0	1	-1	0	

$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	RHS	ratio
1	-5	0	0	0	1	-1	6	
0	-1	1	0	0	1/3	0	2	
0	0	0	0	1	2/3	0	6	
0	-6	0	1	0	1	-1	0	

From this tableau we can see that the LP is unbounded and an extreme point is [0, 2, 0, 6, 0] and an extreme direction is [1, 1, 6, 0, 0].

### Degeneracy and the Simplex Algorithm

Degeneracy must be considered in the simplex algorithm, as it causes some trouble. For instance, it might mislead us into thinking there are multiple optimal solutions, or provide faulty insight. Further, the algorithm as described can *cycle*, that is, remain on a degenerate extreme point repeatedly cycling through a subset of bases that represent that point, never leaving.

min	$z$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$rhs$
	1	0	0	0	3/4	-20	1/2	-6	0
	0	1	0	0	1/4	-8	-1	9	0
	0	0	1	0	1/2	-12	-1/2	3	0
	0	0	0	1	0	0	1	0	1

Solve the following LP using the Simplex Algorithm:

$$\begin{aligned} \max \quad & z = 40x_1 + 30x_2 \\ \text{s.t.} \quad & 6x_1 + 4x_2 \leq 40 \\ & 4x_1 + 2x_2 \leq 20 \\ & x_1, x_2 \geq 0. \end{aligned}$$

By adding slack variables, we have the following tableau. Luckily, this tableau represents a basis, where  $BV=\{s_1, s_2\}$ , but by inspecting the row 0 (objective function row) coefficients, we can see that this is not optimal. By Dantzig's Rule, we enter  $x_1$  into the basis, and by the ratio test we see that  $s_2$  leaves the basis. By performing elementary row operations, we obtain the following tableau for the new basis  $BV=\{s_1, x_1\}$ .

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	-40	-30	0	0	0
0	6	4	1	0	40
0	4	2	0	1	20

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	0	-10	0	10	200
0	0	1	1	-3/2	10
0	1	1/2	0	1/4	5

This tableau is not optimal, entering  $x_2$  into the basis can improve the objective function value. The basic variables  $s_1$  and  $x_1$  tie in the ration test. If we have  $x_1$  leave the basis, we get the following tableau ( $BV=\{s_1, x_2\}$ ).

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	20	0	0	15	300
0	-2	0	1	-2	0
0	2	1	0	1/2	10

This is an optimal tableau, with an objective function value of 300, If instead of  $x_1$  leaving the basis, suppose  $s_1$  left, this would lead to the following tableau ( $BV=\{x_2, x_1\}$ ).

$z$	$x_1$	$x_2$	$s_1$	$s_2$	RHS
1	0	0	10	-5	300
0	0	1	1	-3/2	10
0	1	0	-1/2	1	0

This tableau does not look optimal, yet the objective function value is the same as the optimal solution's. This occurs because the optimal extreme point is a degenerate.

## 12. Duality

---

Before I prove the stronger duality theorem, let me first provide some intuition about where this duality thing comes from in the first place.<sup>6</sup> Consider the following linear programming problem:

$$\begin{aligned} & \text{maximize } 4x_1 + x_2 + 3x_3 \\ \text{subject to } & x_1 + 4x_2 \leq 2 \\ & 3x_1 - x_2 + x_3 \leq 4 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Let  $\sigma^*$  denote the optimum objective value for this LP. The feasible solution  $x = (1, 0, 0)$  gives us a lower bound  $\sigma^* \geq 4$ . A different feasible solution  $x = (0, 0, 3)$  gives us a better lower bound  $\sigma^* \geq 9$ . We could play this game all day, finding different feasible solutions and getting ever larger lower bounds. How do we know when we're done? Is there a way to prove an upper bound on  $\sigma^*$ ?

In fact, there is. Let's multiply each of the constraints in our LP by a new non-negative scalar value  $y_i$ :

$$\begin{aligned} & \text{maximize } 4x_1 + x_2 + 3x_3 \\ \text{subject to } & y_1(x_1 + 4x_2) \leq 2y_1 \\ & y_2(3x_1 - x_2 + x_3) \leq 4y_2 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Because each  $y_i$  is non-negative, we do not reverse any of the inequalities. Any feasible solution  $(x_1, x_2, x_3)$  must satisfy both of these inequalities, so it must also satisfy their sum:

$$(y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 \leq 2y_1 + 4y_2.$$

Now suppose that each  $y_i$  is larger than the  $i$  th coefficient of the objective function:

$$y_1 + 3y_2 \geq 4, \quad 4y_1 - y_2 \geq 1, \quad y_2 \geq 3.$$

This assumption lets us derive an upper bound on the objective value of any feasible solution:

$$4x_1 + x_2 + 3x_3 \leq (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 \leq 2y_1 + 4y_2.$$

In particular, by plugging in the optimal solution  $(x_1^*, x_2^*, x_3^*)$  for the original LP, we obtain the following upper bound on  $\sigma^*$ :

$$\sigma^* = 4x_1^* + x_2^* + 3x_3^* \leq 2y_1 + 4y_2.$$

Now it's natural to ask how tight we can make this upper bound. How small can we make the expression  $2y_1 + 4y_2$  without violating any of the inequalities we used to prove the upper bound? This is just another

linear programming problem.

$$\begin{array}{ll} \text{minimize} & 2y_1 + 4y_2 \\ \text{subject to} & y_1 + 3y_2 \geq 4 \\ & 4y_1 - y_2 \geq 1 \\ & y_2 \geq 3 \\ & y_1, y_2 \geq 0 \end{array}$$

"This example is taken from Robert Vanderbei's excellent textbook Linear Programming: Foundations and Extensions [Springer, 2001], but the idea appears earlier in Jens Clausen's 1997 paper 'Teaching Duality in Linear Programming: The Multiplier Approach'.

<https://www.cs.purdue.edu/homes/egrigore/580FT15/26-lp-jefferickson.pdf>

In which we introduce the theory of duality in linear programming.

## 12.1 The Dual of Linear Program

---

Suppose that we have the following linear program in maximization standard form:

$$\begin{array}{ll} \text{maxim} & x_1 + 2x_2 + x_3 + x_4 \\ \text{subject to} & x_1 + 2x_2 + x_3 \leq 2 \\ & x_2 + x_4 \leq 1 \\ & x_1 + 2x_3 \leq 1 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_3 \geq 0 \end{array}$$

and that an LP-solver has found for us the solution  $x_1 := 1, x_2 := \frac{1}{2}, x_3 := 0, x_4 := \frac{1}{2}$  of cost 2.5. How can we convince ourselves, or another user, that the solution is indeed optimal, without having to trace the steps of the computation of the algorithm?

Observe that if we have two valid inequalities

$$a \leq b \text{ and } c \leq d$$

then we can deduce that the inequality

$$a + c \leq b + d$$

(derived by "summing the left hand sides and the right hand sides" of our original inequalities) is also true. In fact, we can also scale the inequalities by a positive multiplicative factor before adding them up, so for every non-negative values  $y_1, y_2 \geq 0$  we also have

$$y_1 a + y_2 c \leq y_1 b + y_2 d$$

Going back to our linear program (1), we see that if we scale the first inequality by  $\frac{1}{2}$ , add the second inequality, and then add the third inequality scaled by  $\frac{1}{2}$ , we get that, for every  $(x_1, x_2, x_3, x_4)$  that is feasible for (1),

$$x_1 + 2x_2 + 1.5x_3 + x_4 \leq 2.5$$

And so, for every feasible  $(x_1, x_2, x_3, x_4)$ , its cost is

$$x_1 + 2x_2 + x_3 + x_4 \leq x_1 + 2x_2 + 1.5x_3 + x_4 \leq 2.5$$

meaning that a solution of cost 2.5 is indeed optimal.

In general, how do we find a good choice of scaling factors for the inequalities, and what kind of upper bounds can we prove to the optimum?

Suppose that we have a maximization linear program in standard form.

$$\begin{aligned} & \text{maximize} && c_1x_1 + \dots + c_nx_n \\ & \text{subject to} && \\ & && a_{1,1}x_1 + \dots + a_{1,n}x_n \leq b_1 \\ & && \vdots \\ & && a_{m,1}x_1 + \dots + a_{m,n}x_n \leq b_m \\ & && x_1 \geq 0 \\ & && \vdots \\ & && x_n \geq 0 \end{aligned}$$

For every choice of non-negative scaling factors  $y_1, \dots, y_m$ , we can derive the inequality

$$\begin{aligned} & y_1 \cdot (a_{1,1}x_1 + \dots + a_{1,n}x_n) \\ & \quad + \dots \\ & \quad + y_m \cdot (a_{m,1}x_1 + \dots + a_{m,n}x_n) \\ & \leq y_1b_1 + \dots + y_mb_m \end{aligned}$$

which is true for every feasible solution  $(x_1, \dots, x_n)$  to the linear program (2). We can rewrite the inequality as

$$\begin{aligned} & (a_{1,1}y_1 + \dots + a_{m,1}y_m) \cdot x_1 \\ & \quad + \dots \\ & \quad + (a_{1,n}y_1 + \dots + a_{m,n}y_m) \cdot x_n \\ & \leq y_1b_1 + \dots + y_mb_m \end{aligned}$$

So we get that a certain linear function of the  $x_i$  is always at most a certain value, for every feasible  $(x_1, \dots, x_n)$ . The trick is now to choose the  $y_i$  so that the linear function of the  $x_i$  for which we get an upper bound is, in turn, an upper bound to the cost function of  $(x_1, \dots, x_n)$ . We can achieve this if we choose the  $y_i$  such that

$$c_1 \leq a_{1,1}y_1 + \cdots + a_{m,1}y_m$$

⋮

$$c_n \leq a_{1,n}y_1 + \cdots + a_{m,n}y_m$$

Now we see that for every non-negative  $(y_1, \dots, y_m)$  that satisfies (3), and for every  $(x_1, \dots, x_n)$  that is feasible for (2),

$$\begin{aligned} & c_1x_1 + \cdots + c_nx_n \\ & \leq (a_{1,1}y_1 + \cdots + a_{m,1}y_m) \cdot x_1 \\ & \quad + \cdots \\ & \quad + (a_{1,n}y_1 + \cdots + a_{m,n}y_m) \cdot x_n \\ & \leq y_1b_1 + \cdots + y_mb_m \end{aligned}$$

Clearly, we want to find the non-negative values  $y_1, \dots, y_m$  such that the above upper bound is as strong as possible, that is we want to

$$\begin{aligned} & \text{minimize} && b_1y_1 + \cdots + b_my_m \\ & \text{subject to} && \\ & && a_{1,1}y_1 + \cdots + a_{m,1}y_m \geq c_1 \\ & && \vdots \\ & && a_{n,1}y_1 + \cdots + a_{m,n}y_m \geq c_n \\ & && y_1 \geq 0 \\ & && \vdots \\ & && y_m \geq 0 \end{aligned}$$

So we find out that if we want to find the scaling factors that give us the best possible upper bound to the optimum of a linear program in standard maximization form, we end up with a new linear program, in standard minimization form. Definition 1 If

$$\begin{aligned} & \mathbf{c}^T \mathbf{x} \\ & \text{maximize} \\ & \text{subject to} \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

is a linear program in maximization standard form, then its dual is the minimization linear program

$$\begin{aligned} & \mathbf{b}^T \mathbf{y} \\ & \text{subject to} \\ & A^T \mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq 0 \end{aligned}$$

So if we have a linear program in maximization linear form, which we are going to call the primal linear program, its dual is formed by having one variable for each constraint of the primal (not counting the non-negativity constraints of the primal variables), and having one constraint for each variable of the primal

(plus the nonnegative constraints of the dual variables); we change maximization to minimization, we switch the roles of the coefficients of the objective function and of the right-hand sides of the inequalities, and we take the transpose of the matrix of coefficients of the left-hand side of the inequalities.

The optimum of the dual is now an upper bound to the optimum of the primal. How do we do the same thing but starting from a minimization linear program? We can rewrite

$$\begin{aligned} & \underset{l}{\text{minimize}} \quad \mathbf{c}^T \mathbf{y} \\ & \text{subject to} \\ & \quad A\mathbf{y} \geq \mathbf{b} \\ & \quad \mathbf{y} \geq 0 \end{aligned}$$

in an equivalent way as

$$\begin{aligned} & \text{mubject to} \quad -\mathbf{c}^T \mathbf{y} \\ & \text{maximize} \\ & \quad -A\mathbf{y} \leq -\mathbf{b} \\ & \quad \mathbf{y} \geq 0 \end{aligned}$$

If we compute the dual of the above program we get

$$\begin{aligned} & \text{mubject to} \quad -\mathbf{b}^T \mathbf{z} \\ & \text{minimize} \\ & \quad -A^T \mathbf{z} \geq -\mathbf{c} \\ & \quad \mathbf{z} \geq 0 \end{aligned}$$

that is,

$$\begin{aligned} & \text{maximize} \quad \mathbf{b}^T \mathbf{z} \\ & \text{subject to} \\ & \quad A^T \mathbf{z} \leq \mathbf{c} \\ & \quad \mathbf{z} \geq 0 \end{aligned}$$

So we can form the dual of a linear program in minimization normal form in the same way in which we formed the dual in the maximization case:

- switch the type of optimization,
- introduce as many dual variables as the number of primal constraints (not counting the non-negativity constraints),
- define as many dual constraints (not counting the non-negativity constraints) as the number of primal variables.
- take the transpose of the matrix of coefficients of the left-hand side of the inequality,
- switch the roles of the vector of coefficients in the objective function and the vector of right-hand sides in the inequalities.

Note that:

Fact 2 The dual of the dual of a linear program is the linear program itself.

We have already proved the following:

Fact 3 If the primal (in maximization standard form) and the dual (in minimization standard form) are both feasible, then

$$\text{opt( primal )} \leq \text{opt( dual )}$$

Which we can generalize a little

Theorem 4 (Weak Duality Theorem) If  $LP_1$  is a linear program in maximization standard form,  $LP_2$  is a linear program in minimization standard form, and  $LP_1$  and  $LP_2$  are duals of each other then:

- If  $LP_1$  is unbounded, then  $LP_2$  is infeasible;
- If  $LP_2$  is unbounded, then  $LP_1$  is infeasible;
- If  $LP_1$  and  $LP_2$  are both feasible and bounded, then

$$\text{opt}(LP_1) \leq \text{opt}(LP_2)$$

ProOF: We have proved the third statement already. Now observe that the third statement is also saying that if  $LP_1$  and  $LP_2$  are both feasible, then they have to both be bounded, because every feasible solution to  $LP_2$  gives a finite upper bound to the optimum of  $LP_1$  (which then cannot be  $+\infty$ ) and every feasible solution to  $LP_1$  gives a finite lower bound to the optimum of  $LP_2$  (which then cannot be  $-\infty$ ).

What is surprising is that, for bounded and feasible linear programs, there is always a dual solution that certifies the exact value of the optimum.

Theorem 5 (Strong Duality) If either  $LP_1$  or  $LP_2$  is feasible and bounded, then so is the other, and

$$\text{opt}(LP_1) = \text{opt}(LP_2)$$

To summarize, the following cases can arise:

- If one of  $LP_1$  or  $LP_2$  is feasible and bounded, then so is the other;
- If one of  $LP_1$  or  $LP_2$  is unbounded, then the other is infeasible;
- If one of  $LP_1$  or  $LP_2$  is infeasible, then the other cannot be feasible and bounded, that is, the other is going to be either infeasible or unbounded. Either case can happen.

## 12.2 Linear programming duality

---

Consider the following problem:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A} \mathbf{x} \geq \mathbf{b}. \end{aligned} \tag{12.1}$$

In the remark at the end of Chapter ??, we saw that if (12.1) has an optimal solution, then there exists  $\mathbf{y}^* \in \mathbb{R}^m$  such that  $\mathbf{y}^* \geq 0$ ,  $\mathbf{y}^{*\top} \mathbf{A} = \mathbf{c}^T$ , and  $\mathbf{y}^{*\top} \mathbf{b} = \gamma$  where  $\gamma$  denotes the optimal value of (12.1).

Take any  $\mathbf{y} \in \mathbb{R}^m$  satisfying  $\mathbf{y} \geq 0$  and  $\mathbf{y}^\top \mathbf{A} = \mathbf{c}^\top$ . Then we can infer from  $\mathbf{Ax} \geq \mathbf{b}$  the inequality  $\mathbf{y}^\top \mathbf{Ax} \geq \mathbf{y}^\top \mathbf{b}$ , or more simply,  $\mathbf{c}^\top \mathbf{x} \geq \mathbf{y}^\top \mathbf{b}$ . Thus, for any such  $\mathbf{y}$ ,  $\mathbf{y}^\top \mathbf{b}$  gives a lower bound for the objective function value of any feasible solution to (12.1). Since  $\gamma$  is the optimal value of  $(P)$ , we must have  $\gamma \geq \mathbf{y}^\top \mathbf{b}$ .

As  $\mathbf{y}^* \mathbf{b} = \gamma$ , we see that  $\gamma$  is the optimal value of

$$\begin{aligned} \max \quad & \mathbf{y}^\top \mathbf{b} \\ \text{s.t.} \quad & \mathbf{y}^\top \mathbf{A} = \mathbf{c}^\top \\ & \mathbf{y} \geq 0. \end{aligned} \tag{12.2}$$

Note that (12.2) is a linear programming problem! We call it the **dual problem** of the **primal problem** (12.1). We say that the dual variable  $y_i$  is **associated** with the constraint  $\mathbf{a}^{(i)\top} \mathbf{x} \geq b_i$  where  $\mathbf{a}^{(i)\top}$  denotes the  $i$ th row of  $\mathbf{A}$ .

In other words, we define the dual problem of (12.1) to be the linear programming problem (12.2). In the discussion above, we saw that if the primal problem has an optimal solution, then so does the dual problem and the optimal values of the two problems are equal. Thus, we have the following result:

### Theorem 12.1: strong-duality-special

*Suppose that (12.1) has an optimal solution. Then (12.2) also has an optimal solution and the optimal values of the two problems are equal.*

At first glance, requiring all the constraints to be  $\geq$ -inequalities as in (12.1) before forming the dual problem seems a bit restrictive. We now see how the dual problem of a primal problem in general form can be defined. We first make two observations that motivate the definition.

#### Observation 1

Suppose that our primal problem contains a mixture of all types of linear constraints:

$$\begin{aligned} \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} \geq \mathbf{b} \\ & \mathbf{A}' \mathbf{x} \leq \mathbf{b}' \\ & \mathbf{A}'' \mathbf{x} = \mathbf{b}'' \end{aligned} \tag{12.3}$$

where  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,  $\mathbf{A}' \in \mathbb{R}^{m' \times n}$ ,  $\mathbf{b}' \in \mathbb{R}^{m'}$ ,  $\mathbf{A}'' \in \mathbb{R}^{m'' \times n}$ , and  $\mathbf{b}'' \in \mathbb{R}^{m''}$ .

We can of course convert this into an equivalent problem in the form of (12.1) and form its dual.

However, if we take the point of view that the function of the dual is to infer from the constraints of (12.3) an inequality of the form  $\mathbf{c}^\top \mathbf{x} \geq \gamma$  with  $\gamma$  as large as possible by taking an appropriate linear combination of the constraints, we are effectively looking for  $\mathbf{y} \in \mathbb{R}^m$ ,  $\mathbf{y} \geq 0$ ,  $\mathbf{y}' \in \mathbb{R}^{m'}$ ,  $\mathbf{y}' \leq 0$ , and  $\mathbf{y}'' \in \mathbb{R}^{m''}$ , such that

$$\mathbf{y}^\top \mathbf{A} + \mathbf{y}'^\top \mathbf{A}' + \mathbf{y}''^\top \mathbf{A}'' = \mathbf{c}^\top$$

with  $\mathbf{y}^\top \mathbf{b} + \mathbf{y}'^\top \mathbf{b}' + \mathbf{y}''^\top \mathbf{b}''$  to be maximized.

(The reason why we need  $\mathbf{y}' \leq 0$  is because inferring a  $\geq$ -inequality from  $\mathbf{A}' \mathbf{x} \leq \mathbf{b}'$  requires nonpositive multipliers. There is no restriction on  $\mathbf{y}''$  because the constraints  $\mathbf{A}'' \mathbf{x} = \mathbf{b}''$  are equalities.)

This leads to the dual problem:

$$\begin{aligned} \max \quad & \mathbf{y}^T \mathbf{b} + \mathbf{y}'^T \mathbf{b}' + \mathbf{y}''^T \mathbf{b}'' \\ \text{s.t.} \quad & \mathbf{y}^T \mathbf{A} + \mathbf{y}'^T \mathbf{A}' + \mathbf{y}''^T \mathbf{A}'' = \mathbf{c}^T \\ & \mathbf{y} \geq 0 \\ & \mathbf{y}' \leq 0. \end{aligned} \tag{12.4}$$

In fact, we could have derived this dual by applying the definition of the dual problem to

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \begin{bmatrix} \mathbf{A} \\ -\mathbf{A}' \\ \mathbf{A}'' \\ -\mathbf{A}'' \end{bmatrix} \mathbf{x} \geq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b}' \\ \mathbf{b}'' \\ -\mathbf{b}'' \end{bmatrix}, \end{aligned}$$

which is equivalent to (12.3). The details are left as an exercise.

### Observation 2

Consider the primal problem of the following form:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A} \mathbf{x} \geq \mathbf{b} \\ & x_i \geq 0 \quad i \in P \\ & x_i \leq 0 \quad i \in N \end{aligned} \tag{12.5}$$

where  $P$  and  $N$  are disjoint subsets of  $\{1, \dots, n\}$ . In other words, constraints of the form  $x_i \geq 0$  or  $x_i \leq 0$  are separated out from the rest of the inequalities.

Forming the dual of (12.5) as defined under Observation 1, we obtain the dual problem

$$\begin{aligned} \max \quad & \mathbf{y}^T \mathbf{b} \\ \text{s.t.} \quad & \mathbf{y}^T \mathbf{a}^{(i)} = c_i \quad i \in \{1, \dots, n\} \setminus (P \cup N) \\ & \mathbf{y}^T \mathbf{a}^{(i)} + p_i = c_i \quad i \in P \\ & \mathbf{y}^T \mathbf{a}^{(i)} + q_i = c_i \quad i \in N \\ & p_i \geq 0 \quad i \in P \\ & q_i \leq 0 \quad i \in N \end{aligned} \tag{12.6}$$

where  $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$ . Note that this problem is equivalent to the following without the variables  $p_i$ ,  $i \in P$  and  $q_i$ ,  $i \in N$ :

$$\begin{aligned} \max \quad & \mathbf{y}^T \mathbf{b} \\ \text{s.t.} \quad & \mathbf{y}^T \mathbf{a}^{(i)} = c_i \quad i \in \{1, \dots, n\} \setminus (P \cup N) \\ & \mathbf{y}^T \mathbf{a}^{(i)} \leq c_i \quad i \in P \\ & \mathbf{y}^T \mathbf{a}^{(i)} \geq c_i \quad i \in N, \end{aligned} \tag{12.7}$$

which can be taken as the dual problem of (12.5) instead of (12.6). The advantage here is that it has fewer variables than (12.6).

Hence, the dual problem of

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A} \mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

is simply

$$\begin{aligned} \max \quad & \mathbf{y}^T \mathbf{b} \\ \text{s.t.} \quad & \mathbf{y}^T \mathbf{A} \leq \mathbf{c}^T \\ & \mathbf{y} \geq 0. \end{aligned}$$

As we can see from above, there is no need to associate dual variables to constraints of the form  $x_i \geq 0$  or  $x_i \leq 0$  provided we have the appropriate types of constraints in the dual problem. Combining all the observations lead to the definition of the dual problem for a primal problem in general form as discussed next.

### 12.2.1. The dual problem

---

Let  $\mathbf{A} \in \mathbb{R}^{m \times n}$ ,  $\mathbf{b} \in \mathbb{R}^m$ ,  $\mathbf{c} \in \mathbb{R}^n$ . Let  $\mathbf{a}^{(i)^T}$  denote the  $i$ th row of  $\mathbf{A}$ . Let  $\mathbf{A}_j$  denote the  $j$ th column of  $\mathbf{A}$ .

Let  $(P)$  denote the minimization problem with variables in the tuple  $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$  given as follows:

- The objective function to be minimized is  $\mathbf{c}^T \mathbf{x}$
- The constraints are

$$\mathbf{a}^{(i)^T} \mathbf{x} \sqcup_i b_i$$

where  $\sqcup_i$  is  $\leq$ ,  $\geq$ , or  $=$  for  $i = 1, \dots, m$ .

- For each  $j \in \{1, \dots, n\}$ ,  $x_j$  is constrained to be nonnegative, nonpositive, or free (i.e. not constrained to be nonnegative or nonpositive.)

Then the **dual problem** is defined to be the maximization problem with variables in the tuple  $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$  given as follows:

- The objective function to be maximized is  $\mathbf{y}^T \mathbf{b}$
- For  $j = 1, \dots, n$ , the  $j$ th constraint is

$$\begin{cases} \mathbf{y}^T \mathbf{A}_j \leq c_j & \text{if } x_j \text{ is constrained to be nonnegative} \\ \mathbf{y}^T \mathbf{A}_j \geq c_j & \text{if } x_j \text{ is constrained to be nonpositive} \\ \mathbf{y}^T \mathbf{A}_j = c_j & \text{if } x_j \text{ is free.} \end{cases}$$

- For each  $i \in \{1, \dots, m\}$ ,  $y_i$  is constrained to be nonnegative if  $\sqcup_i$  is  $\geq$ ;  $y_i$  is constrained to be nonpositive if  $\sqcup_i$  is  $\leq$ ;  $y_i$  is free if  $\sqcup_i$  is  $=$ .

The following table can help remember the above.

Primal (min)	Dual (max)
$\geq$ constraint	$\geq 0$ variable
$\leq$ constraint	$\leq 0$ variable
$=$ constraint	free variable
$\geq 0$ variable	$\leq$ constraint
$\leq 0$ variable	$\geq$ constraint
free variable	$=$ constraint

Below is an example of a primal-dual pair of problems based on the above definition:

Consider the primal problem:

$$\begin{array}{lllllll} \min & x_1 & - & 2x_2 & + & 3x_3 & \\ \text{s.t.} & -x_1 & & & + & 4x_3 & = 5 \\ & 2x_1 & + & 3x_2 & - & 5x_3 & \geq 6 \\ & & & & 7x_2 & & \leq 8 \\ & x_1 & & & & & \geq 0 \\ & & x_2 & & & & \text{free} \\ & & & & x_3 & & \leq 0. \end{array}$$

Here,  $\mathbf{A} = \begin{bmatrix} -1 & 0 & 4 \\ 2 & 3 & -5 \\ 0 & 7 & 0 \end{bmatrix}$ ,  $\mathbf{b} = \begin{bmatrix} 5 \\ 6 \\ 8 \end{bmatrix}$ , and  $\mathbf{c} = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$ .

The primal problem has three constraints. So the dual problem has three variables. As the first constraint in the primal is an equation, the corresponding variable in the dual is free. As the second constraint in the primal is a  $\geq$ -inequality, the corresponding variable in the dual is nonnegative. As the third constraint in the primal is a  $\leq$ -inequality, the corresponding variable in the dual is nonpositive. Now, the primal problem has three variables. So the dual problem has three constraints. As the first variable in the primal is nonnegative, the corresponding constraint in the dual is a  $\leq$ -inequality. As the second variable in the primal is free, the corresponding constraint in the dual is an equation. As the third variable in the primal

is nonpositive, the corresponding constraint in the dual is a  $\geq$ -inequality. Hence, the dual problem is:

$$\begin{array}{lllll} \max & 5y_1 & + & 6y_2 & + & 8y_3 \\ \text{s.t.} & -y_1 & + & 2y_2 & & \leq 1 \\ & & & 3y_2 & + & 7y_3 = -2 \\ & 4y_1 & - & 5y_2 & & \geq 3 \\ & y_1 & & & & \text{free} \\ & & & y_2 & & \geq 0 \\ & & & & y_3 & \leq 0. \end{array}$$

**Remarks.** Note that in some books, the primal problem is always a maximization problem. In that case, what is our primal problem is their dual problem and what is our dual problem is their primal problem.

One can now prove a more general version of Theorem 12.2 as stated below. The details are left as an exercise.

### Theorem 12.2: Duality Theorem for Linear Programming

Let  $(P)$  and  $(D)$  denote a primal-dual pair of linear programming problems. If either  $(P)$  or  $(D)$  has an optimal solution, then so does the other. Furthermore, the optimal values of the two problems are equal.

Theorem 12.2.1 is also known informally as **strong duality**.

## Exercises

---

1. Write down the dual problem of

$$\begin{array}{llll} \min & 4x_1 & - & 2x_2 \\ \text{s.t.} & x_1 & + & 2x_2 \geq 3 \\ & 3x_1 & - & 4x_2 = 0 \\ & & & x_2 \geq 0. \end{array}$$

2. Write down the dual problem of the following:

$$\begin{array}{llll} \min & 3x_2 & + & x_3 \\ \text{s.t.} & x_1 & + & x_2 + 2x_3 = 1 \\ & & & x_1 - 3x_3 \leq 0 \\ & x_1, & x_2, & x_3 \geq 0. \end{array}$$

3. Write down the dual problem of the following:

$$\begin{array}{llll} \min & x_1 & - & 9x_3 \\ \text{s.t.} & x_1 - 3x_2 + 2x_3 = 1 \\ & x_1 & & \leq 0 \\ & & x_2 & \text{free} \\ & & & x_3 \geq 0. \end{array}$$

4. Determine all values  $c_1, c_2$  such that the linear programming problem

$$\begin{aligned} \min \quad & c_1x_1 + c_2x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \geq 2 \\ & x_1 + 3x_2 \geq 1. \end{aligned}$$

has an optimal solution. Justify your answer

## Solutions

---

1. The dual is

$$\begin{aligned} \max \quad & 3y_1 \\ \text{s.t.} \quad & y_1 + 3y_2 = 4 \\ & 2y_1 - 4y_2 \leq -2 \\ & y_1 \geq 0. \end{aligned}$$

2. The dual is

$$\begin{aligned} \max \quad & y_1 \\ \text{s.t.} \quad & y_1 + y_2 \leq 0 \\ & y_1 \leq 3 \\ & 2y_1 - 3y_2 \leq 1 \\ & y_1 \quad \text{free} \\ & y_2 \leq 0. \end{aligned}$$

3. The dual is

$$\begin{aligned} \max \quad & y_1 \\ \text{s.t.} \quad & y_1 \geq 1 \\ & -3y_1 = 0 \\ & 2y_1 \leq -9 \\ & y_1 \quad \text{free}. \end{aligned}$$

4. Let (P) denote the given linear programming problem.

Note that  $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  is a feasible solution to (P). Therefore, by Theorem ??, it suffices to find all values  $c_1, c_2$  such that

(P) is not unbounded. This amounts to finding all values  $c_1, c_2$  such that the dual problem of (P) has a feasible solution.

The dual problem of (P) is

$$\begin{aligned} \max \quad & 2y_1 + y_2 \\ & 2y_1 + y_2 = c_1 \\ & y_1 + 3y_2 = c_2 \\ & y_1, y_2 \geq 0. \end{aligned}$$

The two equality constraints gives  $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{3}{5}c_1 - \frac{1}{5}c_2 \\ -\frac{1}{5}c_1 + \frac{2}{5}c_2 \end{bmatrix}$ . Thus, the dual problem is feasible if and

only if  $c_1$  and  $c_2$  are real numbers satisfying

$$\begin{aligned}\frac{3}{5}c_1 - \frac{1}{5}c_2 &\geq 0 \\ -\frac{1}{5}c_1 + \frac{2}{5}c_2 &\geq 0,\end{aligned}$$

or more simply,

$$\frac{1}{3}c_2 \leq c_1 \leq 2c_2.$$



## 13. Sensitivity Analysis

---



# 14. Multi-Objective Optimization

---

## Outcomes

- Define multi objective optimization problems
- Discuss the solutions in terms of the Pareto Frontier
- Explore approaches for finding the Pareto Frontier
- Use software to solve for or approximate the Pareto Frontier

## Resources

*Python Multi Objective Optimization (Pymoo)*

## 14.1 Multi Objective Optimization and The Pareto Frontier

---

On Dealing with Ties and Multiple Objectives in Linear Programming

Consider a high end furniture manufacturer which builds dining tables and chairs out of expensive bocote and rosewood.

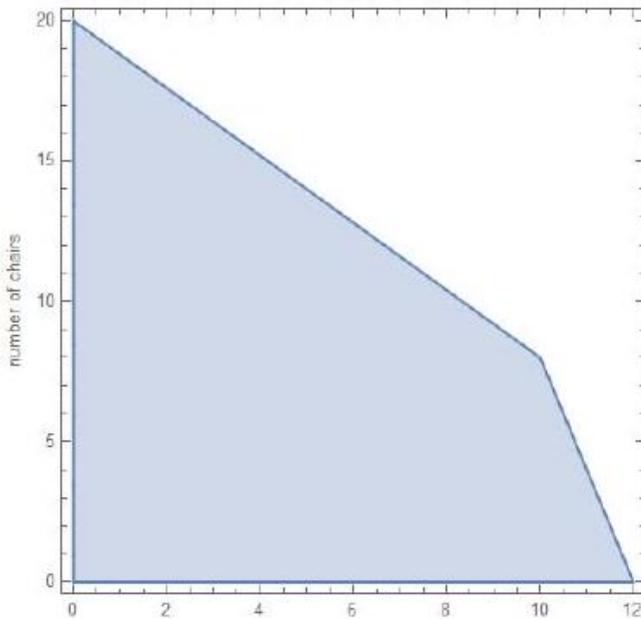


The manufacturer has an ongoing deal with a foreign sawmill which supplies them with 960 and 200 board-feet (bdft) of bocote and rosewood respectively each month.

A single table requires 80 bdft of bocote and 20 bdft of rosewood.

Each chair requires only 20 bdft of bocote but 10 bdft of rosewood.

$$\begin{aligned} P = \{(x,y) \in \mathbb{R}^2 : \\ 80x + 20y \leq 960 \\ 12x + 10y \leq 200 \\ x, y \geq 0\} \end{aligned}$$



Suppose that each table will sell for \$7000 while a chair goes for \$1500. To increase profit we want to maximize

$$F(x, y) = 8000x + 2000y$$

over  $P$ . Having taken a linear programming class, the manager knows his way around these problems and begins the simplex method:

$$\text{Maximize } 8000x + 2000y$$

$$\text{s.t. } 80x + 20y \leq 960$$

$$12x + 10y \leq 200$$

$$x, y \geq 0$$

-4	-1	0	0	0
4	1	1	0	48
6	5	0	1	100

$$\text{Maximize } 4x + y$$

$$\text{s.t. } 4x + y + s_1 = 48$$

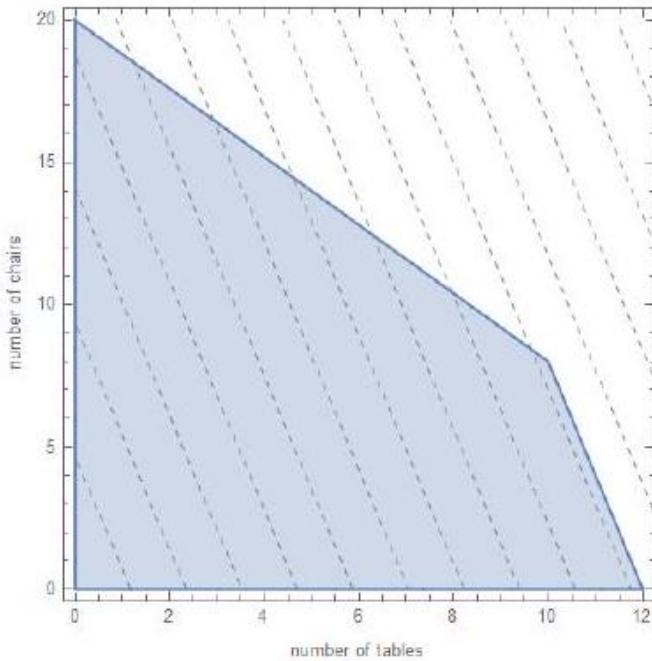
$$\begin{array}{ll} f & 6x + 5y + s_2 = 100 \\ 2000 & x, y \geq 0 \end{array}$$

Having found an optimal solution, the manager is quick to set up production. The best thing to do is produce 12 tables a month and no chairs!

But there are actually multiple optima!

How could we have noticed this from the tableau? From the original formulation?

Is the manager's solution really the best?



Having fired the prior manager for producing no chairs, a new and more competent manager is hired. This one knows that *Dalbergia stevensonii* (the tree which produces their preferred rosewood) is a threatened species and decides that she doesn't want to waste any more rosewood than is necessary.

After some investigation, she finds that table production wastes nearly 10 bdft of rosewood per table while chairs are dramatically more efficient wasting only 2 bdft per chair. She comes up with a new, secondary objective function that she would like to minimize:

$$w(x, y) = 10x + 2y.$$

Having noticed that there are multiple profit-maximizers, she formulates a new problem to break the tie:

$$\begin{aligned} & \text{Minimize } 10x + 2y \\ \text{s.t. } & 80x + 20y = 960 \\ & x \in [10, 12] \\ & y \in [0, 8]. \end{aligned}$$

This is easy in this case because the set of profit-optimal solutions is simple.

Because this is an LP, the optimal solution will be at an extreme point; there are only two here, so the problem reduces to

$$\arg \min \{10x + 2y : (x, y) \in \{(12, 0), (10, 8)\}\}$$

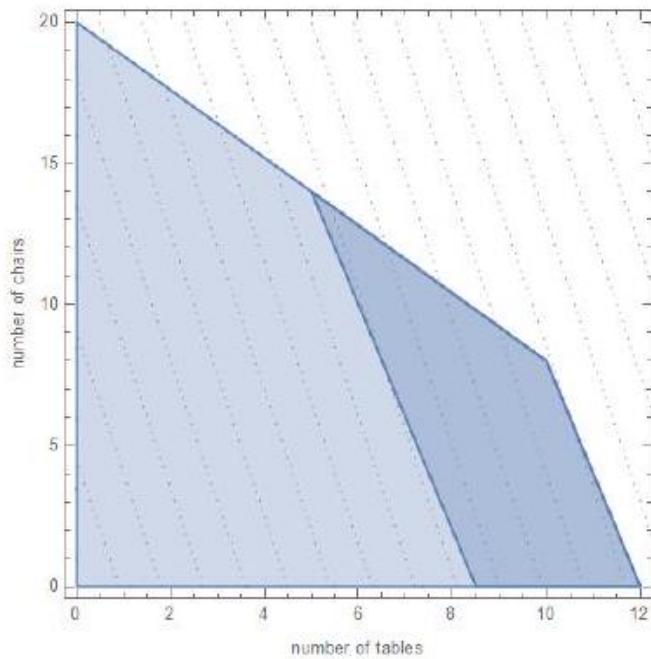
Therefore, swapping out some tables for chairs reduces waste and without affecting revenue!

What the manager just did is called the Ordered Criteria or Lexicographic method for Multi-Objective Optimization. After a few months, the manager convinces the owners that reducing waste is worth a small

loss in profit. The owners concede to a 30% loss in revenue and our manager gets to work on a new model:

$$\begin{aligned}
 & \text{Minimize } 10x + 2y \\
 \text{s.t.} \quad & 8000x + 2000y \geq (\alpha)96000 \\
 & 80x + 20y \leq 960 \\
 & 12x + 10y \leq 200 \\
 & x, y \geq 0
 \end{aligned}$$

where  $\alpha = 0.7$ . This new constraint limits us to solutions which offered at least 70% of maximum possible revenue.



The strategy is called the Benchmark or Rollover method because we choose a benchmark for one of our objectives (revenue in this case), roll that benchmark into the constraints, and optimize for the second objective (waste).

Notice that if we set  $\alpha$  to 1, the rollover problem is equivalent to the lexicographic problem. Either approach requires a known optimal value to the first objective function.

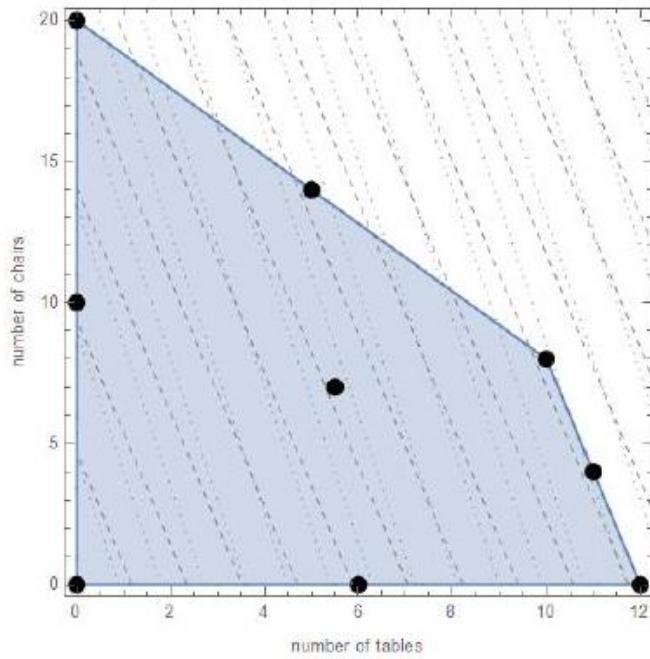
Interestingly, our rollover solution is NOT an extreme point to the ORIGINAL feasible region. Given a set  $P$  and some number of functions  $f_i : P \rightarrow \mathbb{R}$  that we seek to maximize, we call a point  $\mathbf{x} \in P$  Pareto Optimal or Efficient if there does not exist another point  $\bar{\mathbf{x}} \in P$  such that

- $f_i(\bar{\mathbf{x}}) > f_i(\mathbf{x})$  for some  $i$  and  
 $\rightarrow f_j(\bar{\mathbf{x}}) \geq f_j(\mathbf{x})$  for all  $j \neq i$ .

That is, we cannot make any objective better without making some other objective worse.

The Pareto Frontier is the set of all Pareto optimal points for some problem. Which of these points is Pareto optimal?

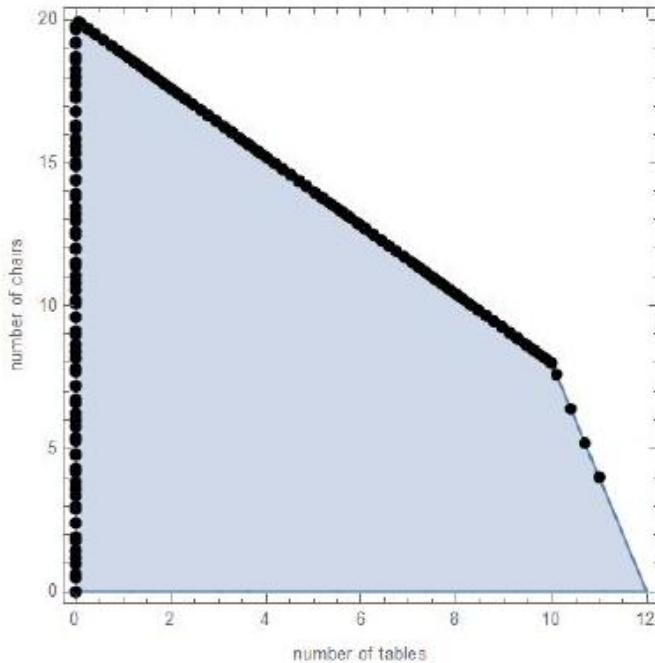
What is the frontier of this problem?



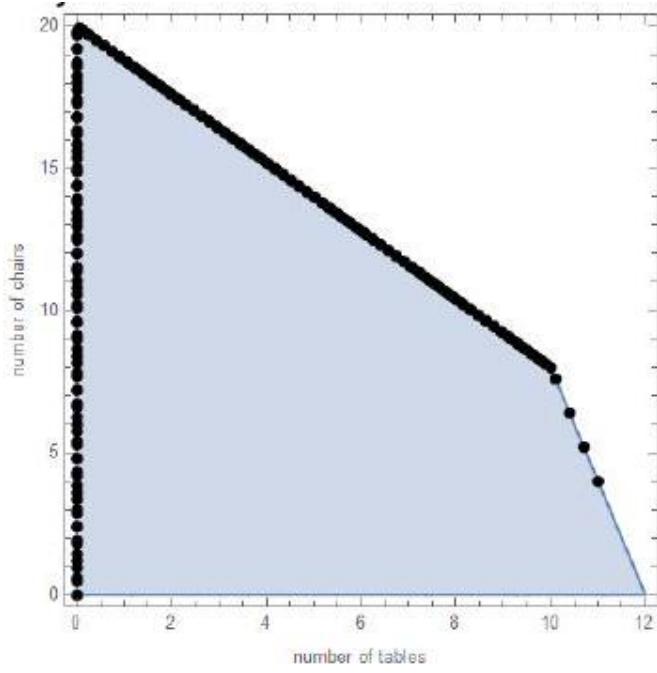
The rollover method is generalized in Goal Programming

By varying  $\alpha$ , it is possible to generate many distinct efficient solutions.

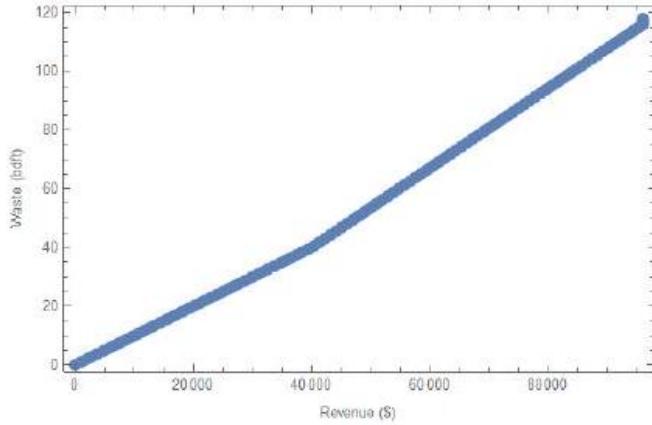
However, this method can generate inefficient solutions if the underlying model is poorly constructed.



It is more common to see a Pareto frontier plotted with respect to its objectives.



number of table



One of the owners of our manufactory decides to explore possible planning himself; he implements the multi-objective method that he remembers, Scalarization by picking some arbitrary constant  $\lambda \in [0, 1]$  and combining his two objectives like so:

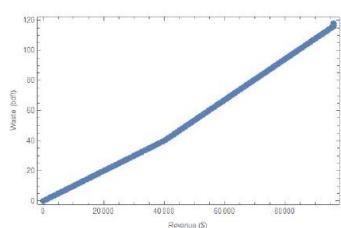
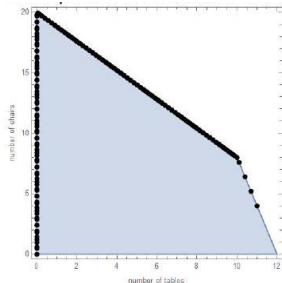
$$\begin{aligned} \text{Minimize} \quad & \lambda(8000x + 2000y) + (1 - \lambda)(10x + 2y) \\ \text{s.t.} \quad & 80x + 20y \leq 960 \\ & 12x + 10y \leq 200 \\ & x, y \geq 0 \end{aligned}$$

What is the benefit of this method?

Where does it fall short?

## 14.2 What points will the Scalarization method find if we vary $\lambda$ ?

---

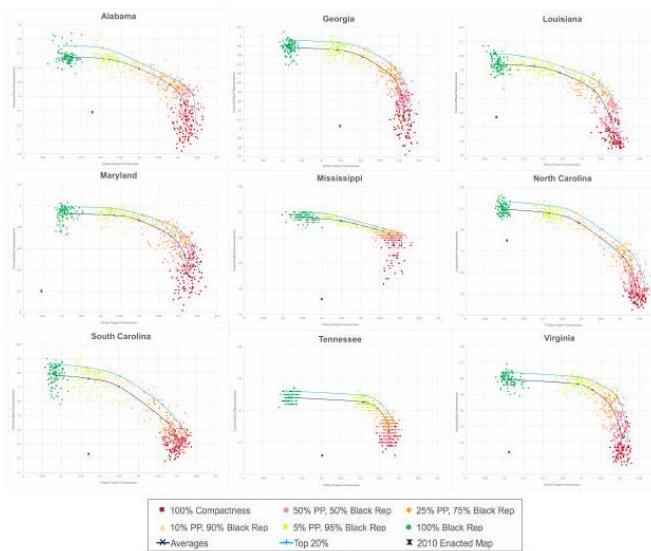


These are all nice ideas, but the problem presented above is neither difficult nor practical.

What are some areas that a Pareto frontier would be actually useful?

## 14.3 Political Redistricting [3]

---

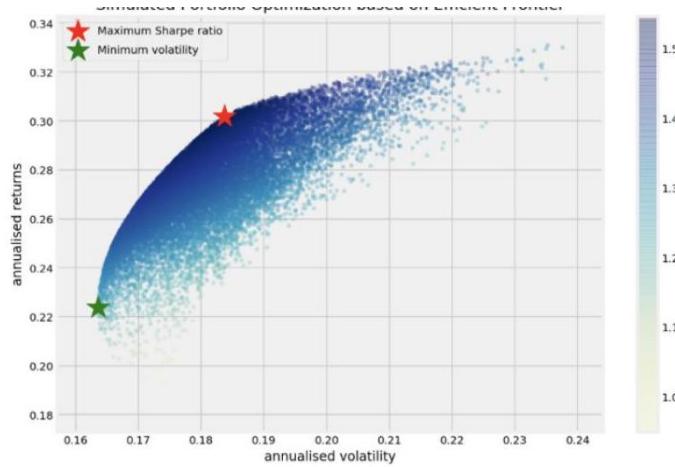


## 14.4 Portfolio Optimization [5]

---

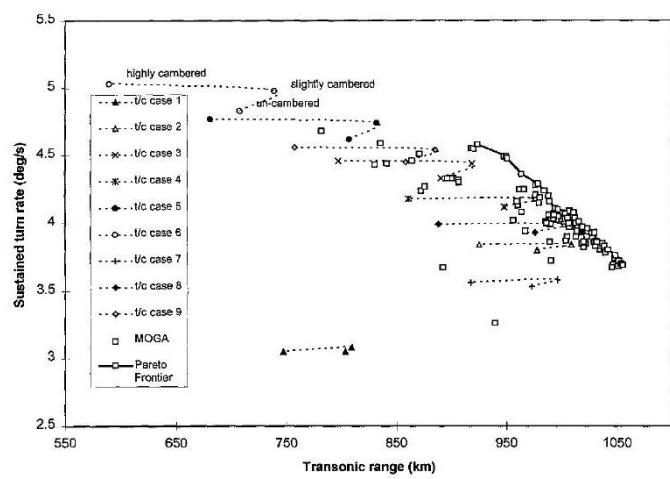
## 14.5 Simulated Portfolio Optimization based on Efficient Frontier

---



## 14.6 Aircraft Design [1]

---



## 14.7 Vehicle Dynamics [4]

---

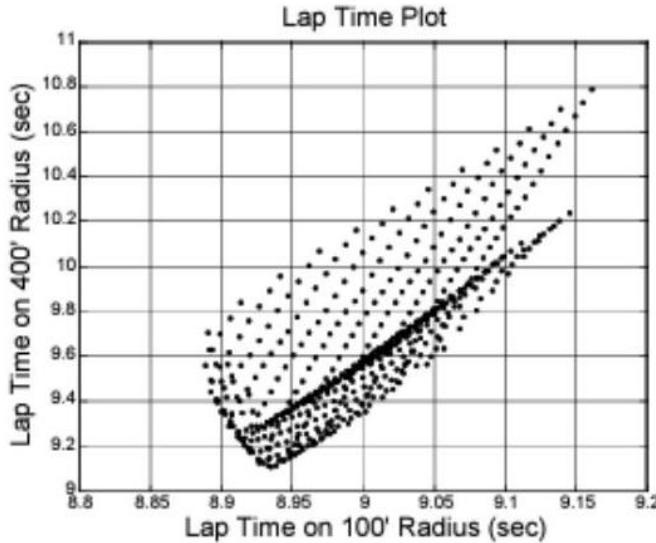
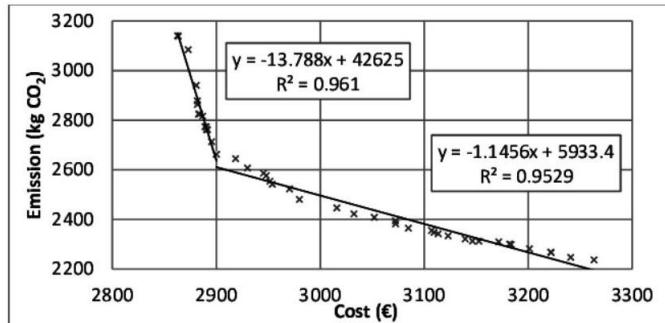


Figure 7: Grid Search Results in the Performance Space

## 14.8 Sustainable Construction [2]

---



## 14.9 References

---

S. Fenwick and John C. Harris. the application of pareto frontier methods in the multidisciplinary wing design of a generic modern military delta aircraft: Semantic scholar, Jan 1999.

URL: <https://WWW.semanticscholar.org/paper/>

The-application-of-Pareto-frontier-methods-in-the-a-Fenwick-Harris/ fced 00a59 d200c2c74 ed 655 a 457344 bcleea 6 ff 5.

T García-Segura, V Yepes, and J Alcalá.

Sustainable design using multiobjective optimization of high-strength concrete i-beams. In The 2014 International Conference on High Performance and Optimum Design of Structures and Materials HPSM/OPTI, volume 137, pages 347 – 358, 2014.

URL: [https://www.researchgate.net/publication/271439836\\_Sustainable\\_design\\_using\\_multiobjective\\_optimization\\_of\\_high-strength\\_concrete\\_l-beams](https://www.researchgate.net/publication/271439836_Sustainable_design_using_multiobjective_optimization_of_high-strength_concrete_l-beams).

Nicholas Goedert, Robert Hildebrand, Laurel Travis, Matthew Pierson, and Jamie Fravel. Black representation and district compactness in southern congressional districts. not yet published, ask Dr. Hildebrand for it.

Edward M Kasprzak and Kemper E Lewis.

Pareto analysis in multiobjective optimization using the collinearity theorem and scaling method. Structural and Multidisciplinary Optimization.

Ricky Kim.

Efficient frontier portfolio optimisation in python, Jun 2021.

URL: <https://towardsdatascience.com/efficient-frontier-portfolio-optimisation-in-python-e7844051e7f>.

# **Part III**

## **Discrete Algorithms**



# 15. Graph Algorithms

---

## Resources

*Youtube! Video of many graph algorithms by Google engineer (6+ hours)*

## 15.1 Graph Theory and Network Flows

---

In the modern world, planning efficient routes is essential for business and industry, with applications as varied as product distribution, laying new fiber optic lines for broadband internet, and suggesting new friends within social network websites like Facebook.

This field of mathematics started nearly 300 years ago as a look into a mathematical puzzle (we'll look at it in a bit). The field has exploded in importance in the last century, both because of the growing complexity of business in a global economy and because of the computational power that computers have provided us.

## 15.2 Graphs

---

### 15.2.1. Drawing Graphs

---

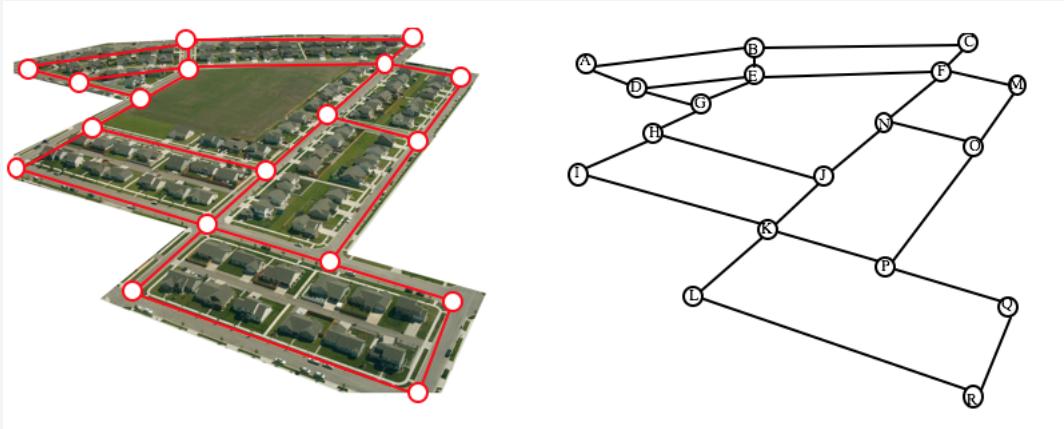
#### Example 15.1

*Here is a portion of a housing development from Missoula, Montana<sup>a</sup>. As part of her job, the development's lawn inspector has to walk down every street in the development making sure homeowners' landscaping conforms to the community requirements.*



Naturally, she wants to minimize the amount of walking she has to do. Is it possible for her to walk down every street in this development without having to do any backtracking? While you might be able to answer that question just by looking at the picture for a while, it would be ideal to be able to answer the question for any picture regardless of its complexity.

To do that, we first need to simplify the picture into a form that is easier to work with. We can do that by drawing a simple line for each street. Where streets intersect, we will place a dot.



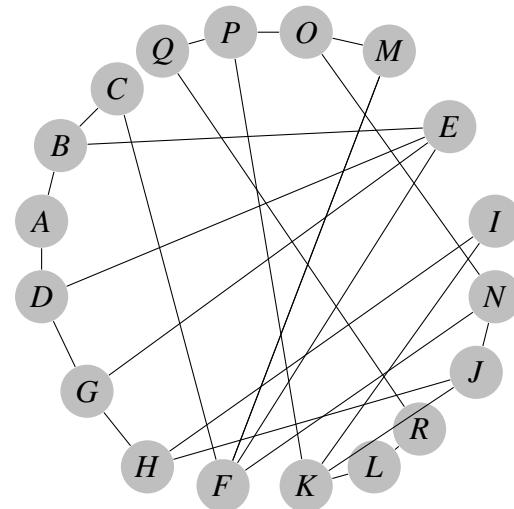
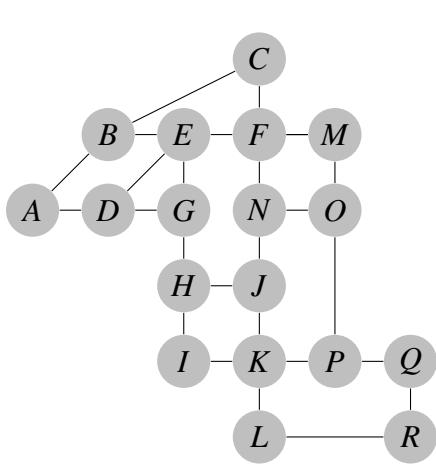
<sup>a</sup>Same Beebe. <http://www.flickr.com/photos/sbeebe/2850476641/>

This type of simplified picture is called a **graph**.

### Definition 15.2: Graphs, Vertices, and Edges

A graph consists of a set of dots, called vertices, and a set of edges connecting pairs of vertices.

While we drew our original graph to correspond with the picture we had, there is nothing particularly important about the layout when we analyze a graph. Both of the graphs below are equivalent to the one drawn above since they show the same edge connections between the same vertices as the original graph.



You probably already noticed that we are using the term graph differently than you may have used the term in the past to describe the graph of a mathematical function.

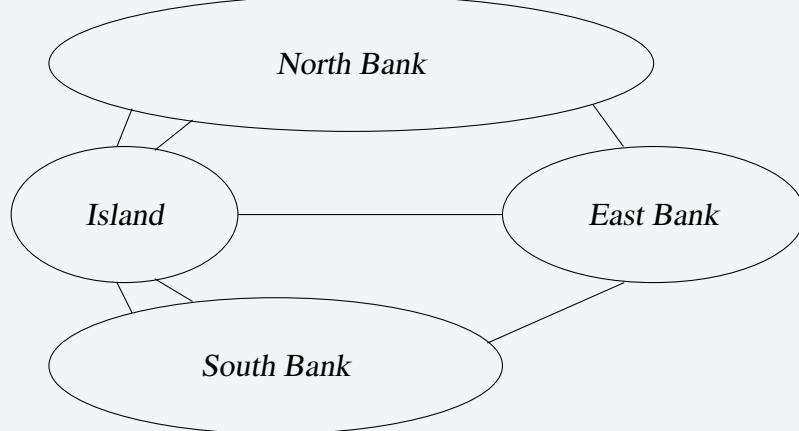
### Example 15.3

*Back in the 18th century in the Prussian city of Königsberg, a river ran through the city and seven bridges crossed the forks of the river. The river and the bridges are highlighted in the picture to the right*

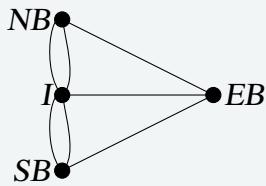
#### Picture

*As a weekend amusement, townsfolk would see if they could find a route that would take them across every bridge once and return them to where they started.*

*Leonard Euler (pronounced OY-lur), one of the most prolific mathematicians ever, looked at this problem in 1735, laying the foundation for graph theory as a field in mathematics. To analyze this problem, Euler introduced edges representing the bridges:*



*Since the size of each land mass it is not relevant to the question of bridge crossings, each can be shrunk down to a vertex representing the location:*



Notice that in this graph there are two edges connecting the north bank and island, corresponding to the two bridges in the original drawing. Depending upon the interpretation of edges and vertices appropriate to a scenario, it is entirely possible and reasonable to have more than one edge connecting two vertices.

While we haven't answered the actual question yet of whether or not there is a route which crosses every bridge once and returns to the starting location, the graph provides the foundation for exploring this question.

## 15.3 Definitions

While we loosely defined some terminology earlier, we now will try to be more specific.

### Definition 15.4: Vertex

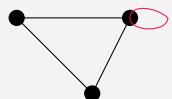
A vertex is a dot in the graph that could represent an intersection of streets, a land mass, or a general location, like "work?" or "school". Vertices are often connected by edges. Note that vertices only occur when a dot is explicitly placed, not whenever two edges cross. Imagine a freeway overpass – the freeway and side street cross, but it is not possible to change from the side street to the freeway at that point, so there is no intersection and no vertex would be placed.

### Definition 15.5: Edges

Edges connect pairs of vertices. An edge can represent a physical connection between locations, like a street, or simply that a route connecting the two locations exists, like an airline flight.

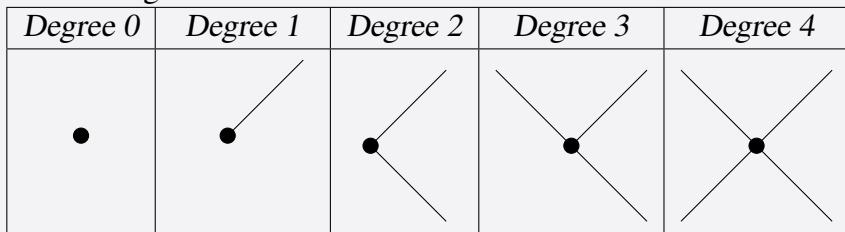
### Definition 15.6: Loop

A loop is a special type of edge that connects a vertex to itself. Loops are not used much in street network graphs.

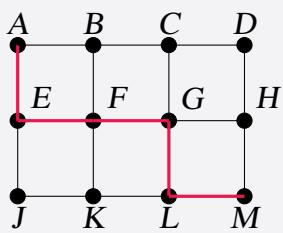


**Definition 15.7: Degree of a vertex**

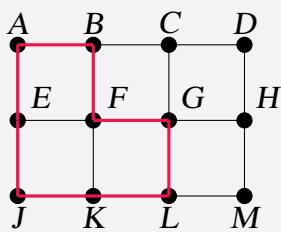
The degree of a vertex is the number of edges meeting at that vertex. It is possible for a vertex to have a degree of zero or larger.

**Definition 15.8: Path**

A path is a sequence of vertices using the edges. Usually we are interested in a path between two vertices. For example, a path from vertex A to vertex M is shown below. It is one of many possible paths in this graph.

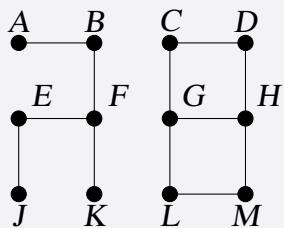
**Definition 15.9: Circuit (a.k.a. cycle)**

A circuit (a.k.a. cycle) is a path that begins and ends at the same vertex. A circuit (a.k.a. cycle) starting and ending at vertex A is shown below.



**Definition 15.10: Connected**

A graph is connected if there is a path from any vertex to any other vertex. Every graph drawn so far has been connected. The graph below is **disconnected**; there is no way to get from the vertices on the left to the vertices on the right.

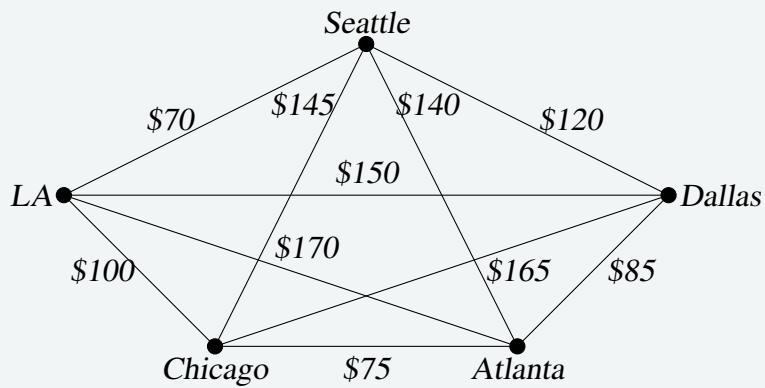
**Definition 15.11: Weights**

Depending upon the problem being solved, sometimes weights are assigned to the edges. The weights could represent the distance between two locations, the travel time, or the travel cost. It is important to note that the distance between vertices in a graph does not necessarily correspond to the weight of an edge.

**Exercise 15.12**

The graph below shows 5 cities. The weights on the edges represent the airfare for a one-way flight between the cities.

- How many vertices and edges does the graph have?
- Is the graph connected?
- What is the degree of the vertex representing LA?
- If you fly from Seattle to Dallas to Atlanta, is that a path or a circuit?
- If you fly from LA to Chicago to Dallas to LA, is that a path or a circuit?



## 15.4 Shortest Path

### Outcomes

- *What is the problem statement?*
- *How to use Dijkstra's algorithm*
- *Software solutions*

### Resources

- *YouTube Video of Dijkstra's Algorithm*
- *Python Example using Networkx and also showing Dijkstra's algorithm*

When you visit a website like Google Maps or use your Smartphone to ask for directions from home to your Aunt's house in Pasadena, you are usually looking for a shortest path between the two locations. These computer applications use representations of the street maps as graphs, with estimated driving times as edge weights.

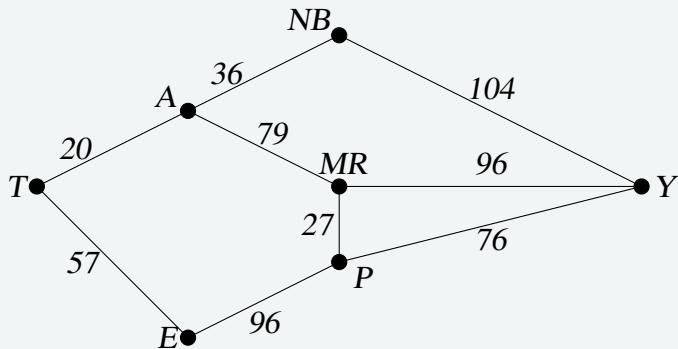
While often it is possible to find a shortest path on a small graph by guess-and-check, our goal in this chapter is to develop methods to solve complex problems in a systematic way by following **algorithms**. An algorithm is a step-by-step procedure for solving a problem. Dijkstra's (pronounced dike-strə) algorithm will find the shortest path between two vertices.

### Dijkstra's Algorithm

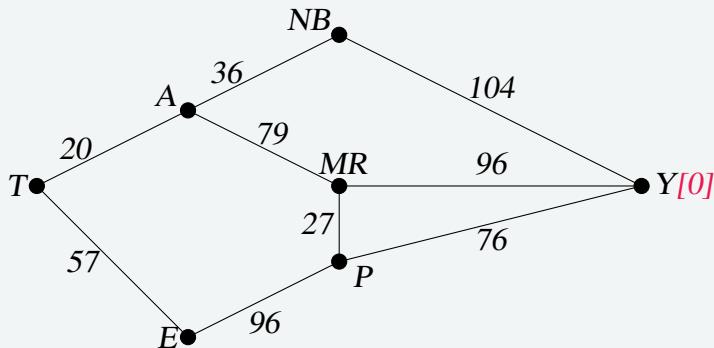
1. Mark the ending vertex with a distance of zero. Designate this vertex as current.
2. Find all vertices leading to the current vertex. Calculate their distances to the end. Since we already know the distance the current vertex is from the end, this will just require adding the most recent edge. Don't record this distance if it is longer than a previously recorded distance.
3. Mark the current vertex as visited. We will never look at this vertex again.
4. Mark the vertex with the smallest distance as current, and repeat from step 2.

**Example 15.13**

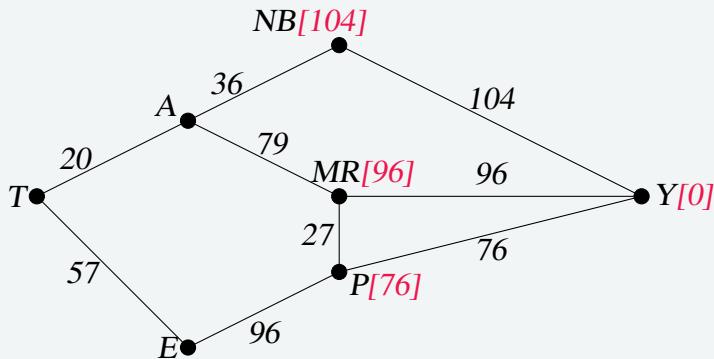
Suppose you need to travel from Yakima, WA (vertex Y) to Tacoma, WA (vertex T). Looking at a map, it looks like driving through Auburn (A) then Mount Rainier (MR) might be shortest, but it's not totally clear since that road is probably slower than taking the major highway through North Bend (NB). A graph with travel times in minutes is shown below. An alternate route through Eatonville (E) and Packwood (P) is also shown.



Step 1: Mark the ending vertex with a distance of zero. The distances will be recorded in [brackets] after the vertex name.



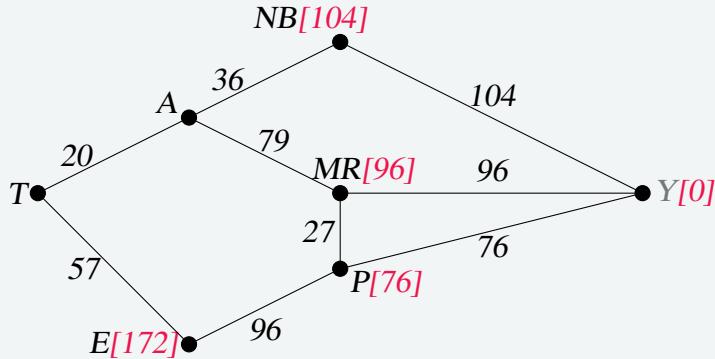
Step 2: For each vertex leading to Y, we calculate the distance to the end. For example, NB is a distance of 104 from the end, and MR is 96 from the end. Remember that distances in this case refer to the travel time in minutes.



Step 3 & 4: We mark Y as visited, and mark the vertex with the smallest recorded distance as current. At this point, P will be designated current. Back to step 2.

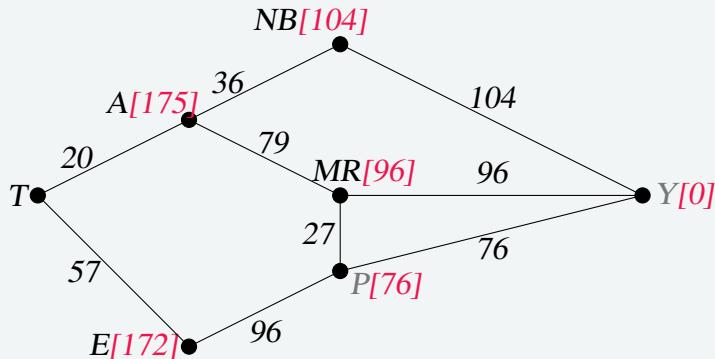
**Step 2 (#2):** For each vertex leading to P (and not leading to a visited vertex) we find the distance from the end. Since E is 96 minutes from P, and we've already calculated P is 76 minutes from Y, we can compute that E is  $96 + 76 = 172$  minutes from Y.

If we make the same computation for MR, we'd calculate  $76 + 27 = 103$ . Since this is larger than the previously recorded distance from Y to MR, we will not replace it.



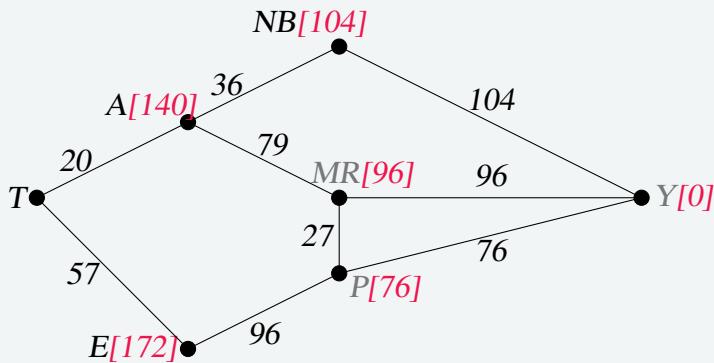
**Step 3 & 4 (#2):** We mark P as visited, and designate the vertex with the smallest recorded distance as current: MR. Back to step 2.

**Step 2 (#3):** For each vertex leading to MR (and not leading to a visited vertex) we find the distance to the end. The only vertex to be considered is A, since we've already visited Y and P. Adding MR's distance 96 to the length from A to MR gives the distance  $96 + 79 = 175$  minutes from A to Y.



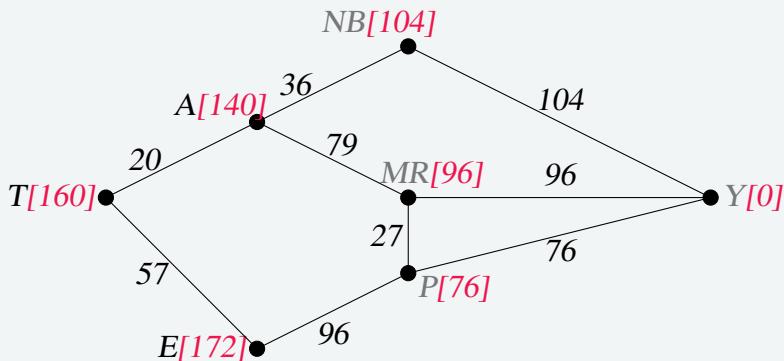
**Step 3 & 4 (#3):** We mark MR as visited, and designate the vertex with smallest recorded distance as current: NB. Back to step 2.

**Step 2 (#4):** For each vertex leading to NB, we find the distance to the end. We know the shortest distance from NB to Y is 104 and the distance from A to NB is 36, so the distance from A to Y through NB is  $104 + 36 = 140$ . Since this distance is shorter than the previously calculated distance from Y to A through MR, we replace it.



Step 3 & 4 (#4): We mark NB as visited, and designate A as current, since it now has the shortest distance.

Step 2 (#5): T is the only non-visited vertex leading to A, so we calculate the distance from T to Y through A:  $20 + 140 = 160$  minutes.



Step 3 & 4 (#5): We mark A as visited, and designate E as current.

Step 2 (#6): The only non-visited vertex leading to E is T. Calculating the distance from T to Y through E, we compute  $172 + 57 = 229$  minutes. Since this is longer than the existing marked time, we do not replace it.

Step 3 (#6): We mark E as visited. Since all vertices have been visited, we are done.

From this, we know that the shortest path from Yakima to Tacoma will take 160 minutes. Tracking which sequence of edges yielded 160 minutes, we see the shortest path is Y-NB-A-T.

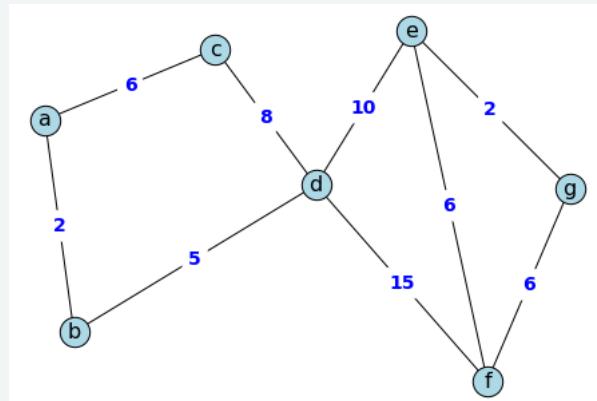
Dijkstra's algorithm is an **optimal algorithm**, meaning that it always produces the actual shortest path, not just a path that is pretty short, provided one exists. This algorithm is also **efficient**, meaning that it can be implemented in a reasonable amount of time. Dijkstra's algorithm takes around  $V^2$  calculations, where V is the number of vertices in a graph<sup>1</sup>. A graph with 100 vertices would take around 10,000 calculations. While that would be a lot to do by hand, it is not a lot for computer to handle. It is because of this efficiency that your car's GPS unit can compute driving directions in only a few seconds.

<sup>1</sup>It can be made to run faster through various optimizations to the implementation.

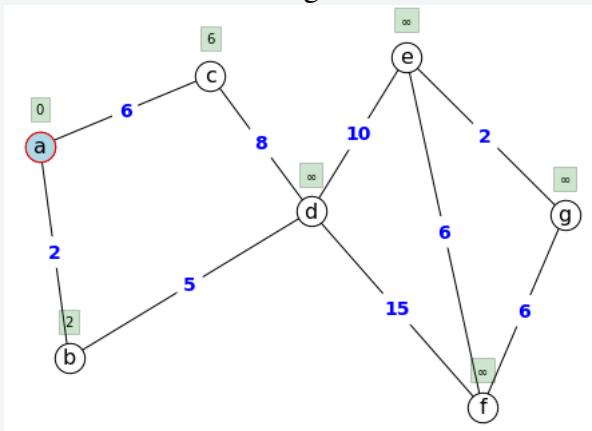
In contrast, an **inefficient** algorithm might try to list all possible paths then compute the length of each path. Trying to list all possible paths could easily take  $10^{25}$  calculations to compute the shortest path with only 25 vertices; that's a 1 with 25 zeros after it! To put that in perspective, the fastest computer in the world would still spend over 1000 years analyzing all those paths.

### Example 15.14: Dijkstra's algorithm example

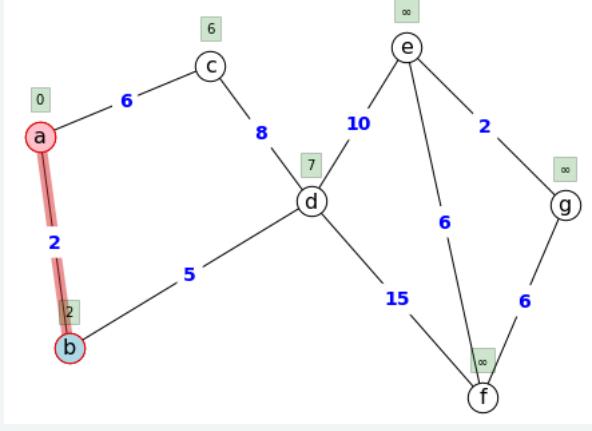
We would like to find a shortest path in the graph from node a to node g. See *Code for python code to solve this problem and create these graphics.*



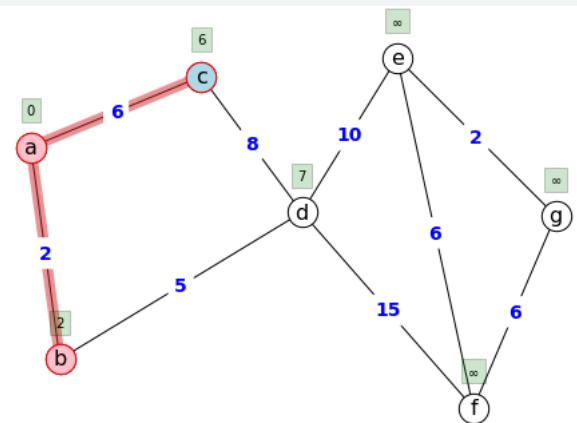
We will initialize our algorithm at node 'a'.



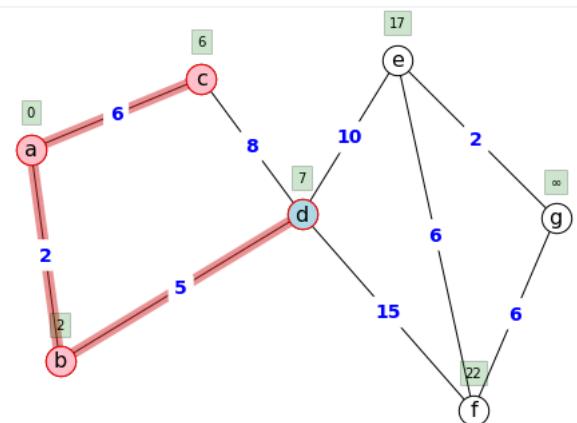
current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞



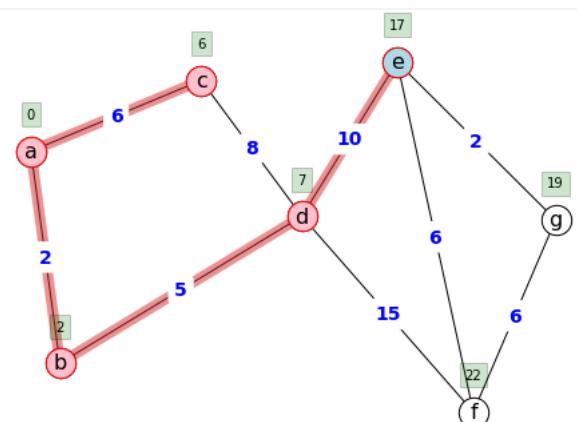
current	a	b	c	d	e	f	g
b	0	2	6	7	∞	∞	∞



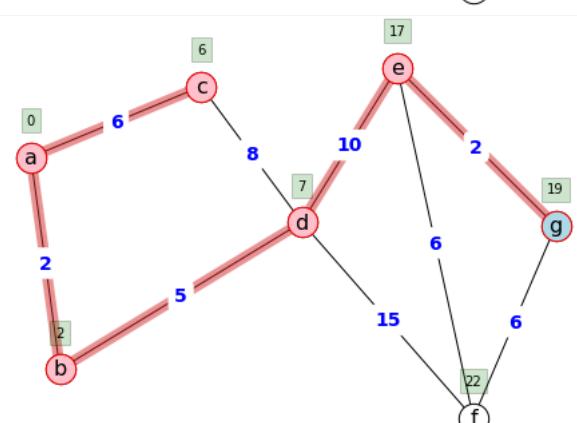
current	a	b	c	d	e	f	g
c	0	2	6	7	$\infty$	$\infty$	$\infty$



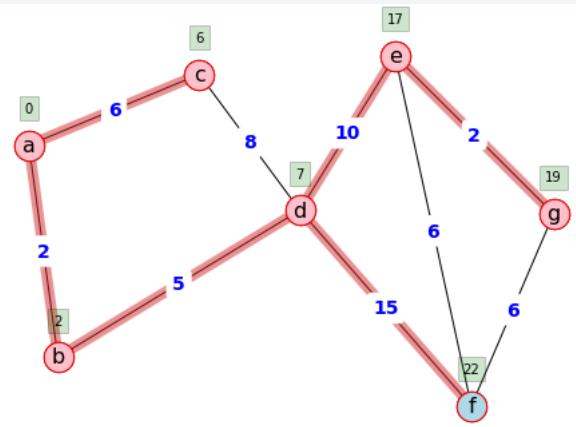
current	a	b	c	d	e	f	g
d	0	2	6	7	17	22	$\infty$



current	a	b	c	d	e	f	g
e	0	2	6	7	17	22	19

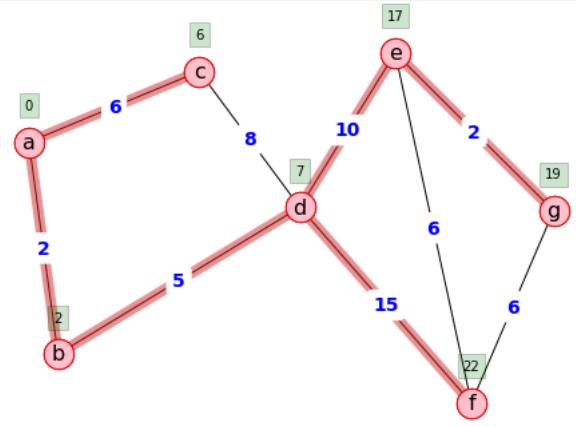


current	a	b	c	d	e	f	g
g	0	2	6	7	17	22	19



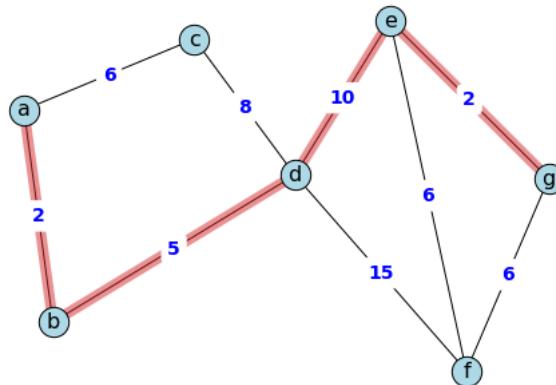
current	a	b	c	d	e	f	g
f	0	2	6	7	17	22	19

We can now summarize our calculations that followed Dijkstra's algorithm.



current	a	b	c	d	e	f	g
a	0	2	6	$\infty$	$\infty$	$\infty$	$\infty$
b	0	2	6	7	$\infty$	$\infty$	$\infty$
c	0	2	6	7	$\infty$	$\infty$	$\infty$
d	0	2	6	7	17	22	$\infty$
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

FINAL SOLUTION The shortest path from a to g is the path a - b - d - e - g,



and has length

$$2 + 5 + 10 + 2 = 19.$$

**Example 15.15**

A shipping company needs to route a package from Washington, D.C. to San Diego, CA. To minimize costs, the package will first be sent to their processing center in Baltimore, MD then sent as part of mass shipments between their various processing centers, ending up in their processing center in Bakersfield, CA. From there it will be delivered in a small truck to San Diego.

The travel times, in hours, between their processing centers are shown in the table below. Three hours has been added to each travel time for processing. Find the shortest path from Baltimore to Bakersfield.

	Baltimore	Denver	Dallas	Chicago	Atlanta	Bakersfield
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

While we could draw a graph, we can also work directly from the table.

Step 1: The ending vertex, Bakersfield, is marked as current.

Step 2: All cities connected to Bakersfield, in this case Denver and Dallas, have their distances calculated; we'll mark those distances in the column headers.

Step 3 & 4: Mark Bakersfield as visited. Here, we are doing it by shading the corresponding row and column of the table. We mark Denver as current, shown in bold, since it is the vertex with the shortest distance.

	Baltimore	<b>Denver</b> [19]	Dallas [25]	Chicago	Atlanta	<b>Bakersfield</b> [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

Step 2 (#2): For cities connected to Denver, calculate distance to the end. For example, Chicago is 18 hours from Denver, and Denver is 19 hours from the end, the distance for Chicago to the end is  $18 + 19 = 37$  (Chicago to Denver to Bakersfield). Atlanta is 24 hours from Denver, so the distance to the end is  $24 + 19 = 43$  (Atlanta to Denver to Bakersfield).

Step 3 & 4 (#2): We mark Denver as visited and mark Dallas as current.

	Baltimore	Denver [19]	Dallas [25]	Chicago [37]	Atlanta [43]	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

Step 2 (#3): For cities connected to Dallas, calculate the distance to the end. For Chicago, the distance from Chicago to Dallas is 18 and from Dallas to the end is 25, so the distance from Chicago to the end through Dallas would be  $18 + 25 = 43$ . Since this is longer than the currently marked distance for Chicago, we do not replace it. For Atlanta, we calculate  $15 + 25 = 40$ . Since this is shorter than the currently marked distance for Atlanta, we replace the existing distance.

Step 3 & 4 (#3): We mark Dallas as visited, and mark Chicago as current.

	Baltimore	Denver [19]	Dallas [25]	Chicago [37]	Atlanta [40]	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

Step 2 (#4): Baltimore and Atlanta are the only non-visited cities connected to Chicago. For Baltimore, we calculate  $15 + 37 = 52$  and mark that distance. For Atlanta, we calculate  $14 + 37 = 51$ . Since this is longer than the existing distance of 40 for Atlanta, we do not replace that distance.

Step 3 & 4 (#4): Mark Chicago as visited and Atlanta as current.

	Baltimore [52]	Denver [19]	Dallas [25]	Chicago [37]	Atlanta [40]	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

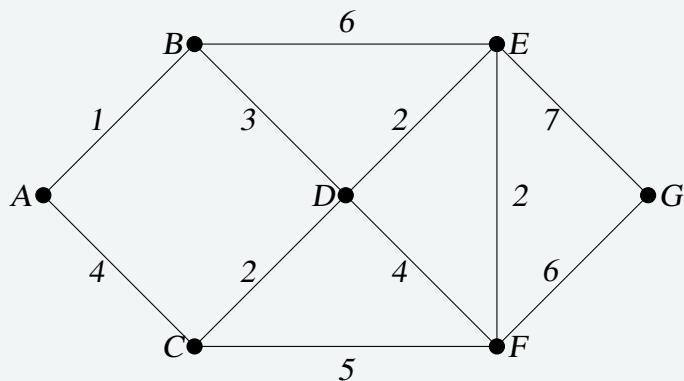
Step 2 (#5): The distance from Atlanta to Baltimore is 14. Adding that to the distance already calculated for Atlanta gives a total distance of  $14 + 40 = 54$  hours from Baltimore to Bakersfield through Atlanta. Since this is larger than the currently calculated distance, we do not replace the distance for Baltimore.

Step 3 & 4 (#5): We mark Atlanta as visited. All cities have been visited and we are done.

The shortest route from Baltimore to Bakersfield will take 52 hours, and will route through Chicago and Denver.

### Exercise 15.16

Find the shortest path between vertices A and G in the graph below.



## 15.5 Spanning Trees

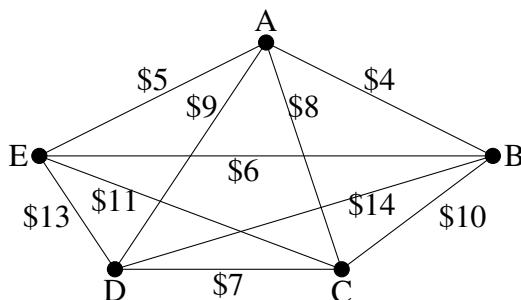
### Outcomes

- Find the smallest set of edges that connects a graph

### Resources

- YouTube Video: Kruskal's algorithm to find a minimum weight spanning tree

A company requires reliable internet and phone connectivity between their five offices (named A, B, C, D, and E for simplicity) in New York, so they decide to lease dedicated lines from the phone company. The phone company will charge for each link made. The costs, in thousands of dollars per year, are shown in the graph.

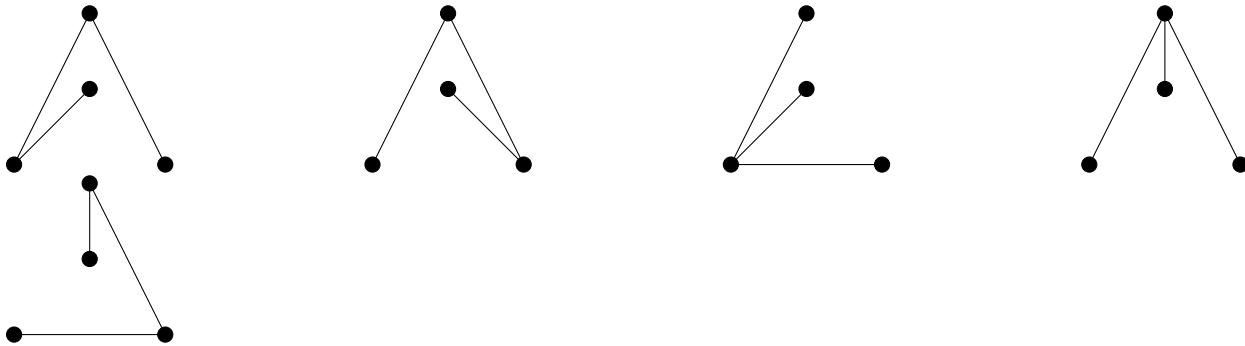


In this case, we don't need to find a circuit, or even a specific path; all we need to do is make sure we can make a call from any office to any other. In other words, we need to be sure there is a path from any vertex to any other vertex.

### Definition 15.17: Spanning Tree

*A spanning tree is a connected graph using all vertices in which there are no circuits. In other words, there is a path from any vertex to any other vertex, but no circuits.*

Some examples of spanning trees are shown below. Notice there are no circuits in the trees, and it is fine to have vertices with degree higher than two.



Usually we have a starting graph to work from, like in the phone example above. In this case, we form our spanning tree by finding a **subgraph** – a new graph formed using all the vertices but only some of the edges from the original graph. No edges will be created where they didn't already exist.

Of course, any random spanning tree isn't really what we want. We want the **minimum cost spanning tree (MCST)**.

### Definition 15.18: Minimum Cost Spanning Tree (MCST)

*The minimum cost spanning tree is the spanning tree with the smallest total edge weight.*

A nearest neighbor style approach doesn't make as much sense here since we don't need a circuit, so instead we will take an approach similar to sorted edges.

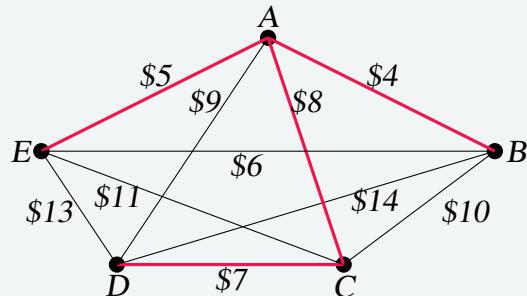
## Kruskal's Algorithm

1. Select the cheapest unused edge in the graph.
2. Repeat step 1, adding the cheapest unused edge, unless:
  - adding the edge would create a circuit.
3. Repeat until a spanning tree is formed.

**Example 15.19**

Using our phone line graph from above, begin adding edges:

AB	\$4	OK
AE	\$5	OK
BE	\$6	reject – closes circuit ABEA
DC	\$7	OK
AC	\$8	OK



At this point we stop – every vertex is now connected, so we have formed a spanning tree with cost \$24 thousand a year.

Remarkably, Kruskal's algorithm is both optimal and efficient; we are guaranteed to always produce the optimal MCST.

**Example 15.20**

The power company needs to lay updated distribution lines connecting the ten Oregon cities below to the power grid. How can they minimize the amount of new line to lay?

	Ashland	Astoria	Bend	Corvallis	Crater Lake	Eugene	Newport	Portland	Salem	Seaside
Ashland	-	374	200	223	108	178	252	285	240	356
Astoria	374	-	255	166	433	199	135	95	136	17
Bend	200	255	-	128	277	128	180	160	131	247
Corvalis	223	166	128	-	430	47	52	84	40	155
Crater Lake	108	433	277	430	-	453	478	344	389	423
Eugene	178	199	128	47	453	-	91	110	64	181
Newport	252	135	180	52	478	91	-	114	83	117
Portland	285	95	160	84	344	110	114	-	47	78
Salem	240	136	131	40	389	64	83	47	-	118
Seaside	356	17	247	155	423	181	117	78	118	-

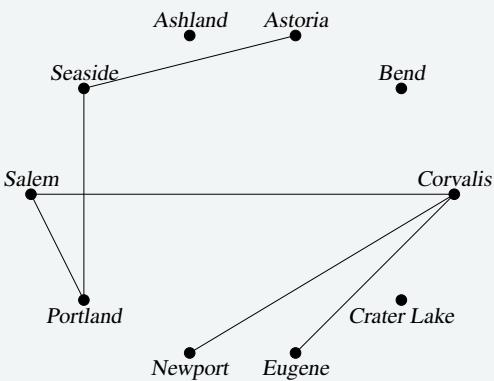
Using Kruskal's algorithm, we add edges from cheapest to most expensive, rejecting any that close a circuit. We stop when the graph is connected.

Seaside to Astoria	17 miles
Corvallis to Salem	40 miles
Portland to Salem	47 miles
Corvallis to Eugene	47 miles
Corvallis to Newport	52 miles
Salem to Eugene	reject – closes circuit
Portland to Seaside	78 miles

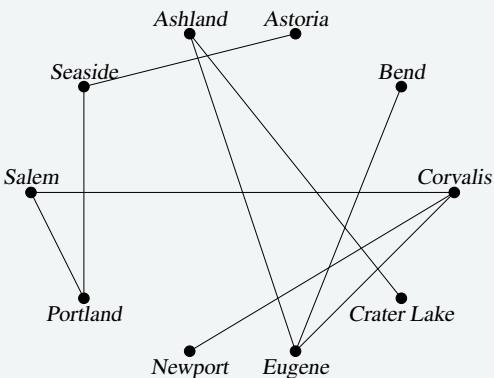
The graph up to this point is shown to the right.

Continuing,

Newport to Salem	reject
Corvallis to Portland	reject
Eugene to Newport	reject
Portland to Astoria	reject
Ashland to Crater Lake	108 miles
Eugene to Portland	reject
Newport to Portland	reject
Newport to Seaside	reject
Salem to Seaside	reject
Bend to Eugene	128 miles
Bend to Salem	reject
Astoria to Newport	reject
Salem to Astoria	reject
Corvallis to Seaside	reject
Portland to Bend	reject
Astoria to Corvallis	reject
Eugene to Ashland	178 miles

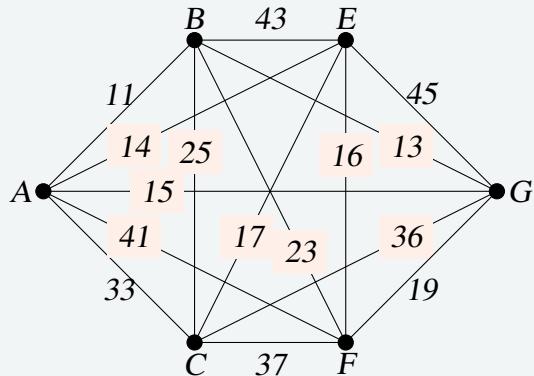


This connects the graph. The total length of cable to lay would be 695 miles.



**Exercise 15.21: Min Cost Spanning Tree**

Find a minimum cost spanning tree on the graph below using Kruskal's algorithm.



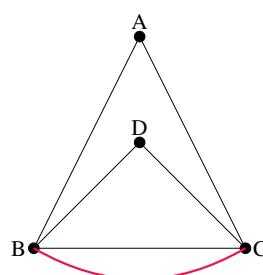
## 15.6 Exercise Answers

---

1. (a) 5 vertices, 10 edges  
 (b) Yes, it is connected.  
 (c) The vertex is degree 4.  
 (d) A path  
 (e) A circuit
2. The shortest path is ABDEG, with length 13.
3. Yes, all vertices have even degree so this graph has an Euler Circuit. There are several possibilities. One is: ABEGFCD(F)EDBCA

4.

This graph can be eulerized by duplicating the edge BC, as shown. One possible Euler circuit on the eulerized graph is ACDBCBA.



5. At each step, we look for the nearest location we haven't already visited.  
 From B the nearest computer is E with time 24.  
 From E, the nearest computer is D with time 11.  
 From D the nearest is A with time 12.  
 From A the nearest is C with time 34.

From C, the only computer we haven't visited is F with time 27.

From F, we return back to B with time 50.

The NNA circuit from B is BEDACFB with time 158 milliseconds.

Using NNA again from other starting vertices:

Starting at A: ADEBCFA: time 146

Starting at C: CDEBAFC: time 167

Starting at D: DEBCFAD: time 146

Starting at E: EDACFBE: time 158

Starting at F: FDEBCAF: time 158

The RNN found a circuit with time 146 milliseconds: ADEBCFA. We could also write this same circuit starting at B if we wanted: BCFADEB or BEDAFCB.

6.

AB: Add, cost 11

BG: Add, cost 13

AE: Add, cost 14

EF: Add, cost 15

EC: Skip (degree 3 at E)

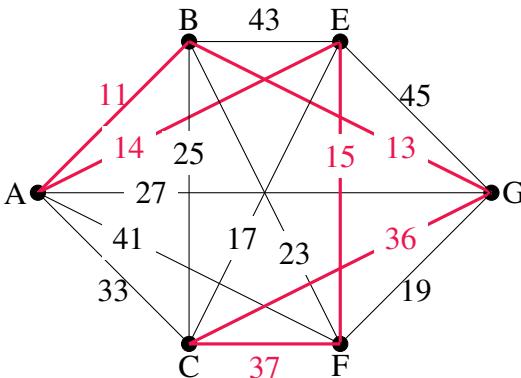
FG: Skip (would create a circuit not including C)

BF, BC, AG, AC: Skip (would cause a vertex to have degree 3)

GC: Add, cost 36

CF: Add, cost 37, completes the circuit

Final circuit: ABGCFEA



7. (??)

AB: Add, cost 11

BG: Add, cost 13

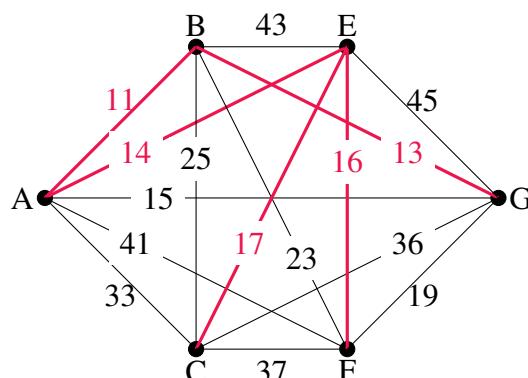
AE: Add, cost 14

AG: Skip, would create circuit ABGA

EF: Add, cost 16

EC: Add, cost 17

This completes the spanning tree.

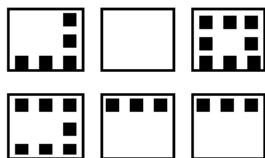


## 15.7 Additional Exercises

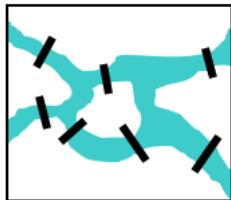
---

### Skills

1. To deliver mail in a particular neighborhood, the postal carrier needs to walk along each of the streets with houses (the dots). Create a graph with edges showing where the carrier must walk to deliver the mail.



2. Suppose that a town has 7 bridges as pictured below. Create a graph that could be used to determine if there is a path that crosses all bridges once.



3. The table below shows approximate driving times (in minutes, without traffic) between five cities in the Dallas area. Create a weighted graph representing this data.

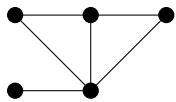
	Plano	Mesquite	Arlington	Denton
Fort Worth	54	52	19	42
Plano		38	53	41
Mesquite			43	56
Arlington				50

4. Shown in the table below are the one-way airfares between 5 cities<sup>2</sup>. Create a graph showing this data.

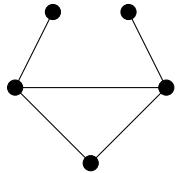
	Honolulu	London	Moscow	Cairo
Seattle	\$159	\$370	\$654	\$684
Honolulu		\$830	\$854	\$801
London			\$245	\$323
Moscow				\$329

5. Find the degree of each vertex in the graph below.

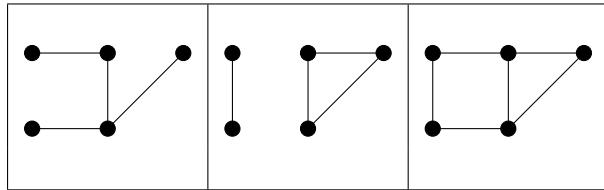
<sup>2</sup>Cheapest fares found when retrieved Sept. 1, 2009 for travel Sept. 22, 2009



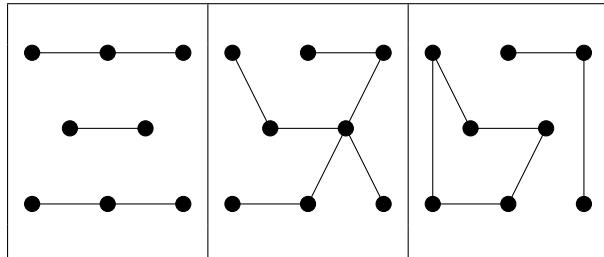
6. Find the degree of each vertex in the graph below.



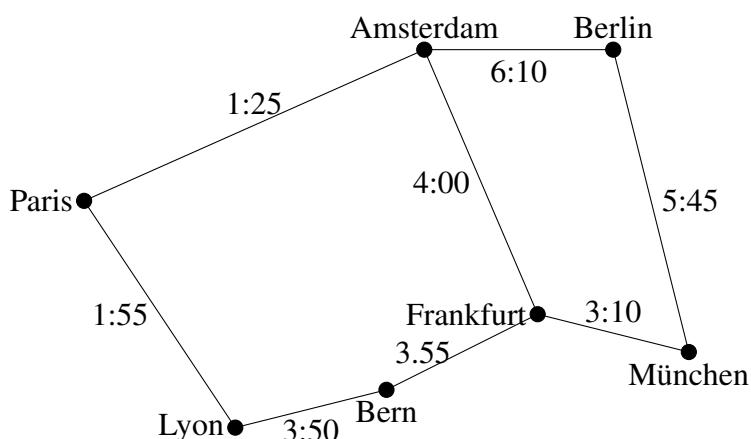
7. Which of these graphs are connected?



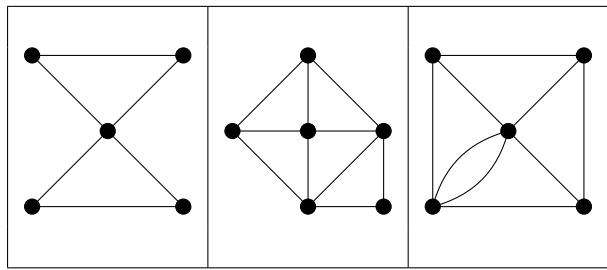
8. Which of these graphs are connected?



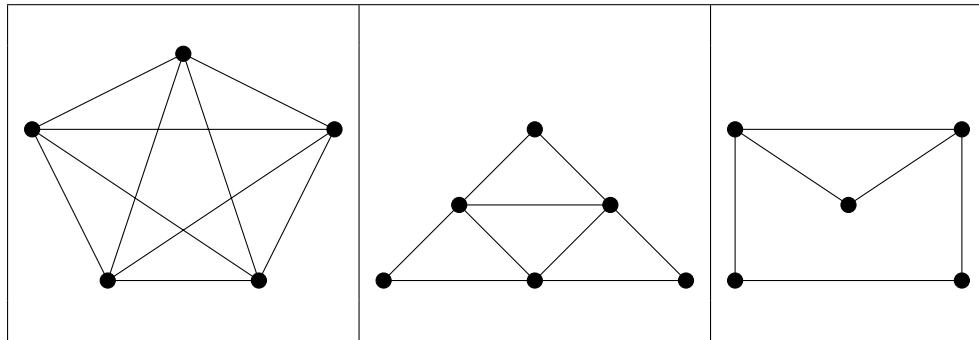
9. Travel times by rail for a segment of the Eurail system is shown below with travel times in hours and minutes. Find path with shortest travel time from Bern to Berlin by applying Dijkstra's algorithm.



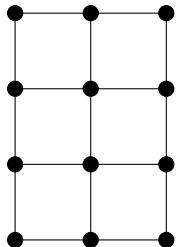
10. Using the graph from the previous problem, find the path with shortest travel time from Paris to München.
11. Does each of these graphs have an Euler circuit? If so, find it.



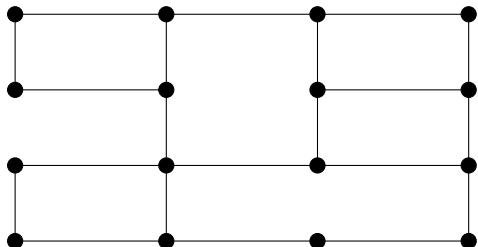
12. Does each of these graphs have an Euler circuit? If so, find it.



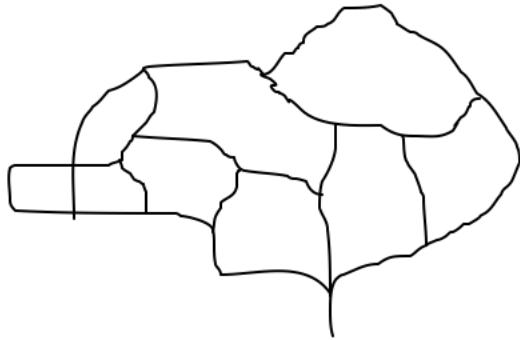
13. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.



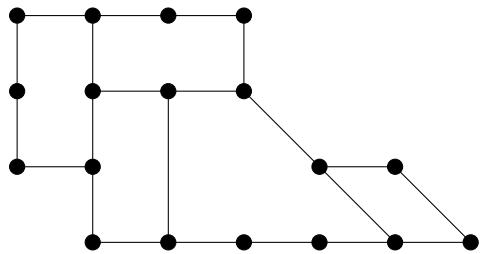
14. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.



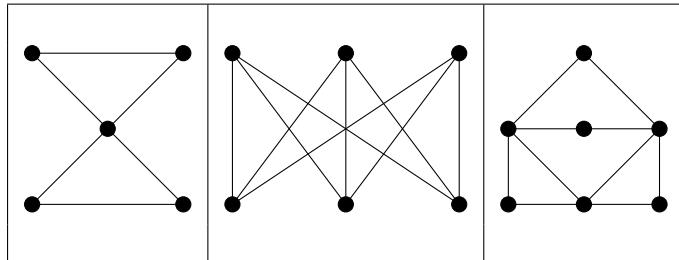
15. The maintenance staff at an amusement park need to patrol the major walkways, shown in the graph below, collecting litter. Find an efficient patrol route by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.



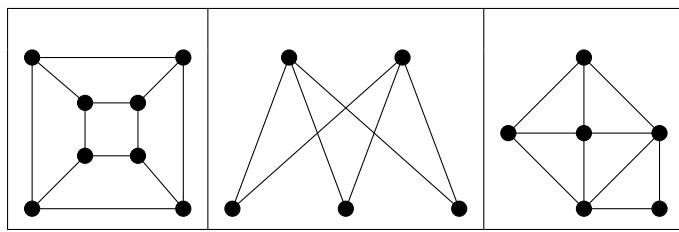
16. After a storm, the city crew inspects for trees or brush blocking the road. Find an efficient route for the neighborhood below by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.



17. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.



18. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.



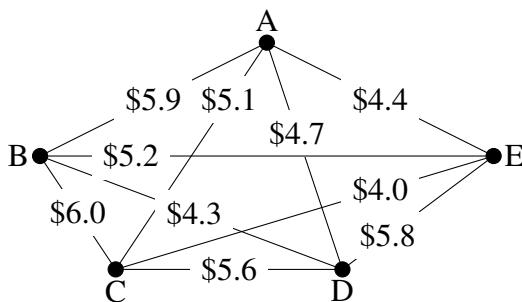
19. A company needs to deliver product to each of their 5 stores around the Dallas, TX area. Driving distances between the stores are shown below. Find a route for the driver to follow, returning to the distribution center in Fort Worth:

- (a) Using Nearest Neighbor starting in Fort Worth
- (b) Using Repeated Nearest Neighbor

## (c) Using Sorted Edges

	Plano	Mesquite	Arlington	Denton
Fort Worth	54	52	19	42
Plano		38	53	41
Mesquite			43	56
Arlington				50

20. A salesperson needs to travel from Seattle to Honolulu, London, Moscow, and Cairo. Use the table of flight costs from problem #4 to find a route for this person to follow:
- Using Nearest Neighbor starting in Seattle
  - Using Repeated Nearest Neighbor
  - Using Sorted Edges
21. When installing fiber optics, some companies will install a sonet ring; a full loop of cable connecting multiple locations. This is used so that if any part of the cable is damaged it does not interrupt service, since there is a second connection to the hub. A company has 5 buildings. Costs (in thousands of dollars) to lay cables between pairs of buildings are shown below. Find the circuit that will minimize cost:
- Using Nearest Neighbor starting at building A
  - Using Repeated Nearest Neighbor
  - Using Sorted Edges



22. A tourist wants to visit 7 cities in Israel. Driving distances, in kilometers, between the cities are shown below<sup>3</sup>. Find a route for the person to follow, returning to the starting city:
- Using Nearest Neighbor starting in Jerusalem
  - Using Repeated Nearest Neighbor
  - Using Sorted Edges

<sup>3</sup>From <http://www.ddtravel-acc.com/Israel-cities-distance.htm>

	Jerusalem	Tel Aviv	Haifa	Tiberias	Beer Sheba	Eilat
Jerusalem	—					
Tel Aviv	58	—				
Haifa	151	95	—			
Tiberias	152	134	69	—		
Beer Sheba	81	105	197	233	—	
Eilat	309	346	438	405	241	—
Nazareth	131	102	35	29	207	488

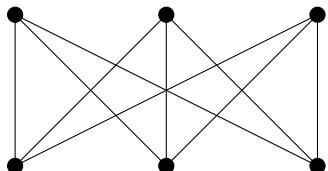
23. Find a minimum cost spanning tree for the graph you created in problem #3.

24. Find a minimum cost spanning tree for the graph you created in problem #22.

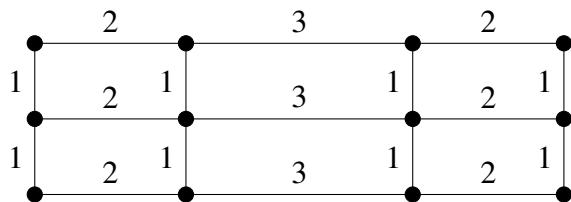
25. Find a minimum cost spanning tree for the graph from problem #21.

## Concepts

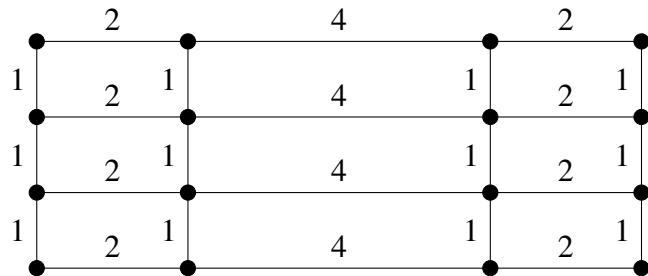
26. Can a graph have one vertex with odd degree? If not, are there other values that are not possible? Why?
27. A complete graph is one in which there is an edge connecting every vertex to every other vertex. For what values of  $n$  does complete graph with  $n$  vertices have an Euler circuit? A Hamiltonian circuit?
28. Create a graph by drawing  $n$  vertices in a row, then another  $n$  vertices below those. Draw an edge from each vertex in the top row to every vertex in the bottom row. An example when  $n = 3$  is shown below. For what values of  $n$  will a graph created this way have an Euler circuit? A Hamiltonian circuit?



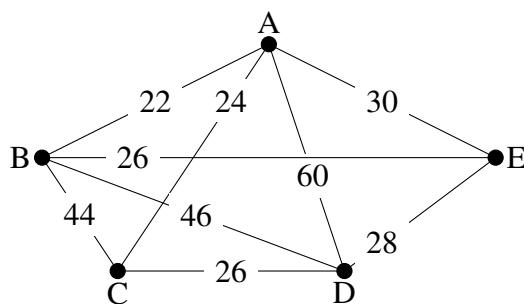
29. Eulerize this graph in the most efficient way possible, considering the weights of the edges.



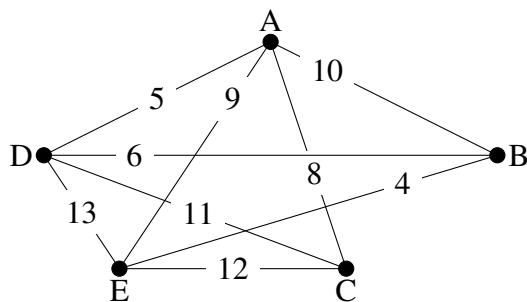
30. Eulerize this graph in the most efficient way possible, considering the weights of the edges.



31. Eulerize this graph in the most efficient way possible, considering the weights of the edges.



32. Eulerize this graph in the most efficient way possible, considering the weights of the edges.



# Explorations

33. Social networks such as Facebook and LinkedIn can be represented using graphs in which vertices represent people and edges are drawn between two vertices when those people are “friends.” The table below shows a friendship table, where an X shows that two people are friends.

- (a) Create a graph of this friendship table
- (b) Find the shortest path from A to D. The length of this path is often called the “degrees of separation” of the two people.
- (c) Extension: Split into groups. Each group will pick 10 or more movies, and look up their major actors ([www.imdb.com](http://www.imdb.com) is a good source). Create a graph with each actor as a vertex, and edges connecting two actors in the same movie (note the movie name on the edge). Find interesting paths between actors, and quiz the other groups to see if they can guess the connections.
34. A spell checker in a word processing program makes suggestions when it finds a word not in the dictionary. To determine what words to suggest, it tries to find similar words. One measure of word similarity is the Levenshtein distance, which measures the number of substitutions, additions, or deletions that are required to change one word into another. For example, the words spit and spot are a distance of 1 apart; changing spit to spot requires one substitution (i for o). Likewise, spit is distance 1 from pit since the change requires one deletion (the s). The word spite is also distance 1 from spit since it requires one addition (the e). The word soot is distance 2 from spit since two substitutions would be required.
- (a) Create a graph using words as vertices, and edges connecting words with a Levenshtein distance of 1. Use the misspelled word “moke” as the center, and try to find at least 10 connected dictionary words. How might a spell checker use this graph?
- (b) Improve the method from above by assigning a weight to each edge based on the likelihood of making the substitution, addition, or deletion. You can base the weights on any reasonable approach: proximity of keys on a keyboard, common language errors, etc. Use Dijkstra’s algorithm to find the length of the shortest path from each word to “moke”. How might a spell checker use these values?
35. The graph below contains two vertices of odd degree. To eulerize this graph, it is necessary to duplicate edges connecting those two vertices.
- (a) Use Dijkstra’s algorithm to find the shortest path between the two vertices with odd degree. Does this produce the most efficient eulerization and solve the Chinese Postman Problem for this graph?
- 
- (b) Suppose a graph has  $n$  odd vertices. Using the approach from part a, how many shortest paths would need to be considered? Is this approach going to be efficient?

### 15.7.1. Notes

---

A paper entitled 'A Note on Two Problems in Connexion with Graphs' was published in the journal 'Numerische Mathematik' in 1959. It was in this paper where the computer scientist named Edsger W. Dijkstra proposed the Dijkstra's Algorithm for the shortest path problem; a fundamental graph theoretic problem. This algorithm can be used to find the shortest path between two nodes or a more common variant of this algorithm is to find the shortest path between a specific 'source' node to any other nodes in the network. <https://www.overleaf.com/project/62472837411e2ce1b881337f>

# **Part IV**

# **Integer Programming**



# 16. Integer Programming Formulations

## Outcomes

- A. Learn classic integer programming formulations.
- B. Demonstrate different uses of binary and integer variables.
- C. Demonstrate the format for modeling an optimization problem with sets, parameters, variables, and the model.

## Resources

- The AIMMS modeling has many great examples. It can be book found here:[AIMMS Modeling Book](#).
- [MIT Open Courseware](#)
- For many real world examples, see this book *Case Studies in Operations Research Applications of Optimal Decision Making*, edited by Murty, Katta G. Or find it [here](#).
- *GUROBI modeling examples by GUROBI*
- *GUROBI modeling examples by Open Optimization* that are linked in this book

In this section, we will describe classical integer programming formulations. These formulations may reflect a real world problem exactly, or may be part of the setup of a real world problem.

## 16.1 Knapsack Problem

The *knapsack problem*<sup>1</sup> can take different forms depending on if the variables are binary or integer. The binary version means that there is only one item of each item type that can be taken. This is typically illustrated as a backpack (knapsack) and some items to put into it (see Figure 16.1), but has applications in many contexts.

### Binary Knapsack Problem:

*NP-Complete*

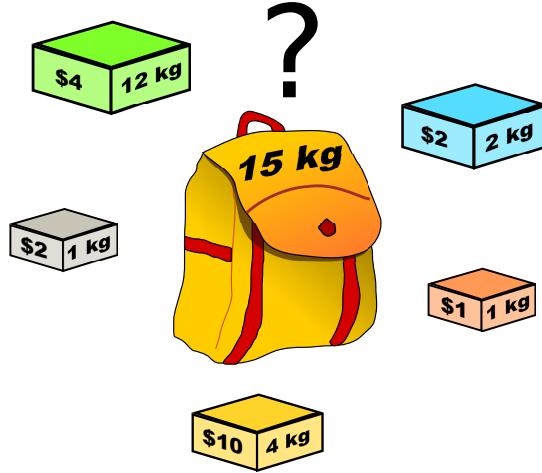
Given a non-negative weight vector  $a \in \mathbb{Q}_+^n$ , a capacity  $b \in \mathbb{Q}_+$ , and objective coefficients  $c \in \mathbb{Q}^n$ ,

$$\begin{aligned} & \max c^\top x \\ & \text{s.t. } a^\top x \leq b \end{aligned} \tag{16.1}$$

$$x \in \{0, 1\}^n$$

<sup>1</sup>Video! - Michel Belaire (EPFL) teaching knapsack problem

<sup>2</sup>[wiki/File/knapsack](#), from [wiki/File/knapsack](#). [wiki/File/knapsack](#), [wiki/File/knapsack](#).

© wiki/File/knapsack<sup>2</sup>

**Figure 16.1: Knapsack Problem: which items should we choose take in the knapsack that maximizes the value while respecting the 15kg weight limit?**

### Example: Knapsack

Code

You have a knapsack (bag) that can only hold  $W = 15$  kgs. There are 5 items that you could possibly put into your knapsack. The items (weight, value) are given as: (12 kg, \$4), (2 kg, \$2), (1kg, \$2), (1kg, \$1), (4kg, \$10). Which items should you take to maximize your value in the knapsack? See Figure 16.1.

#### Variables:

- let  $x_i = 0$  if item  $i$  is in the bag
- let  $x_i = 1$  if item  $i$  is not in the bag

#### Model:

$$\begin{aligned}
 & \text{max } 4x_1 + 2x_2 + 2x_3 + 1x_4 + 10x_5 && \text{(Total value)} \\
 & \text{s.t. } 12x_1 + 2x_2 + 1x_3 + 1x_4 + 4x_5 \leq 15 && \text{(Capacity bound)} \\
 & x_i \in \{0,1\} \text{ for } i = 1, \dots, 5 && \text{(Item taken or not)}
 \end{aligned}$$

In the integer case, we typically require the variables to be non-negative integers, hence we use the notation  $x \in \mathbb{Z}_+^n$ . This setting reflects the fact that instead of single individual items, you have item types of which you can take as many of each type as you like that meets the constraint.

### Integer Knapsack Problem:

#### *NP-Complete*

Given a non-negative weight vector  $a \in \mathbb{Q}_+^n$ , a capacity  $b \in \mathbb{Q}_+$ , and objective coefficients  $c \in \mathbb{Q}^n$ ,

$$\begin{aligned} & \max c^\top x \\ \text{s.t. } & a^\top x \leq b \\ & x \in \mathbb{Z}_+^n \end{aligned} \tag{16.2}$$

We can also consider an equality constrained version

### Equality Constrained Integer Knapsack Problem:

#### *NP-Hard*

Given a non-negative weight vector  $a \in \mathbb{Q}_+^n$ , a capacity  $b \in \mathbb{Q}_+$ , and objective coefficients  $c \in \mathbb{Q}^n$ ,

$$\max c^\top x \tag{16.3}$$

$$\text{s.t. } a^\top x = b \tag{16.4}$$

$$x \in \mathbb{Z}_+^n \tag{16.5}$$

### Example 16.1:

Using pennies, nickels, dimes, and quarters, how can you minimize the number of coins you need to make up a sum of 83¢?

#### Variables:

- Let  $p$  be the number of pennies used
- Let  $n$  be the number of nickels used
- Let  $d$  be the number of dimes used
- Let  $q$  be the number of quarters used

#### Model

$$\begin{array}{ll} \min & p + n + d + q && \text{total number of coins used} \\ \text{s.t.} & p + 5n + 10d + 25q = 83 && \text{sums to 83¢} \\ & p, d, n, q \in \mathbb{Z}_+ && \text{each is a non-negative integer} \end{array}$$

## 16.2 Capital Budgeting

The *capital budgeting* problem is a nice generalization of the knapsack problem. This problem has the same structure as the knapsack problem, except now it has multiple constraints. We will first describe the problem, give a general model, and then look at an explicit example.

### Capital Budgeting:

A firm has  $n$  projects it could undertake to maximize revenue, but budget limitations require that not all can be completed.

- Project  $j$  expects to produce revenue  $c_j$  dollars overall.
- Project  $j$  requires investment of  $a_{ij}$  dollars in time period  $i$  for  $i = 1, \dots, m$ .
- The capital available to spend in time period  $i$  is  $b_i$ .

Which projects should the firm invest in to maximize its expected return while satisfying its weekly budget constraints?

We will first provide a general formulation for this problem.

### **Capital Budgeting Model:**

#### **Sets:**

- Let  $I = \{1, \dots, m\}$  be the set of time periods.
- Let  $J = \{1, \dots, n\}$  be the set of possible investments.

#### **Parameters:**

- $c_j$  is the expected revenue of investment  $j$  for  $j \in J$
- $b_i$  is the available capital in time period  $i$  for  $i$  in  $I$
- $a_{ij}$  is the resources required for investment  $j$  in time period  $i$ , for  $i$  in  $I$ , for  $j$  in  $J$ .

#### **Variables:**

- let  $x_i = 0$  if investment  $i$  is chosen
- let  $x_i = 1$  if investment  $i$  is not chosen

#### **Model:**

$$\begin{aligned}
 & \max \quad \sum_{j=1}^n c_j x_j && \text{(Total Expected Revenue)} \\
 & s.t. \quad \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m && \text{(Resource constraint week } i) \\
 & \quad x_j \in \{0, 1\}, \quad j = 1, \dots, n
 \end{aligned}$$

Consider the example given in the following table.

Project	$\mathbb{E}[\text{Revenue}]$	Resources required in week 1	Resources required in week 2
1	10	3	4
2	8	1	2
3	6	2	1
Resources available		5	6

Given this data, we can setup our problem explicitly as follows

### Example: Capital Budgeting

Code

#### Sets:

- Let  $I = \{1, 2\}$  be the set of time periods.
- Let  $J = \{1, 2, 3\}$  be the set of possible investments.

#### Parameters:

- $c_j$  is given in column " $\mathbb{E}[\text{Revenue}]$ ".
- $b_i$  is given in row "Resources available".
- $a_{ij}$  given in row  $j$ , and column for week  $i$ .

#### Variables:

- let  $x_i = 0$  if investment  $i$  is chosen
- let  $x_i = 1$  if investment  $i$  is not chosen

The explicit model is given by

#### Model:

$$\begin{aligned}
 & \max \quad 10x_1 + 8x_2 + 6x_3 && \text{(Total Expected Revenue)} \\
 & \text{s.t. } 3x_1 + 1x_2 + 2x_3 \leq 5 && \text{(Resource constraint week 1)} \\
 & \quad 4x_1 + 2x_2 + 1x_3 \leq 6 && \text{(Resource constraint week 2)} \\
 & \quad x_j \in \{0, 1\}, \quad j = 1, 2, 3
 \end{aligned}$$

## 16.3 Set Covering

### Resources

Video! - Michel Belaire (EPFL) explaining set covering problem

The *set covering* problem can be used for a wide array of problems. We will see several examples in this section.

**Set Covering:***NP-Complete*

Given a set  $V$  with subsets  $V_1, \dots, V_l$ , determine the smallest subset  $S \subseteq V$  such that  $S \cap V_i \neq \emptyset$  for all  $i = 1, \dots, l$ .

The set cover problem can be modeled as

$$\begin{aligned} & \min 1^\top x \\ \text{s.t. } & \sum_{v \in V_i} x_v \geq 1 \text{ for all } i = 1, \dots, l \\ & x_v \in \{0, 1\} \text{ for all } v \in V \end{aligned} \tag{16.1}$$

where  $x_v$  is a 0/1 variable that takes the value 1 if we include item  $j$  in set  $S$  and 0 if we do not include it in the set  $S$ .

**Resources**

See AIMMS - Media Selection for an example of set covering applied to media selection.

Add flight crew scheduling example and images.

One specific type of set cover problem is the *vertex cover* problem.

**Example: Vertex Cover:***NP-Complete*

Given a graph  $G = (V, E)$  of vertices and edges, we want to find a smallest size subset  $S \subseteq V$  such that every for every  $e = (v, u) \in E$ , either  $u$  or  $v$  is in  $S$ .

We can write this as a mathematical program in the form:

$$\begin{aligned} & \min 1^\top x \\ \text{s.t. } & x_u + x_v \geq 1 \text{ for all } (u, v) \in E \\ & x_v \in \{0, 1\} \text{ for all } v \in V. \end{aligned} \tag{16.2}$$

**Example: Set cover: Fire station placement**

Code

In the fire station problem, we seek to choose locations for fire stations such that any district either contains a fire station, or neighbors a district that contains a fire station. Figure 16.2 depicts the set of districts and an example placement of locations of fire stations. How can we minimize the total number of fire stations that we need?

**Sets:**

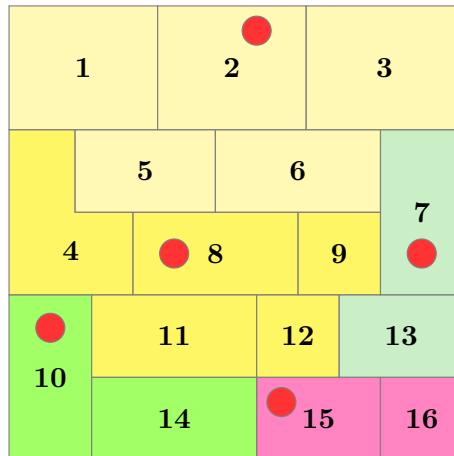
- Let  $V$  be the set of districts ( $V = \{1, \dots, 16\}$ )
- Let  $V_i$  be the set of districts that neighbor district  $i$  (e.g.  $V_1 = \{2, 4, 5\}$ ).

**Variables:**

- let  $x_i = 1$  if district  $i$  is chosen to have a fire station.
- let  $x_i = 0$  otherwise.

**Model:**

$$\begin{aligned}
 \min \quad & \sum_{i \in V} x_i && (\# \text{ open fire stations}) \\
 \text{s.t.} \quad & x_i + \sum_{j \in V_i} x_j \geq 1 && \forall i \in V \quad (\text{Station proximity requirement}) \\
 & x_i \in \{0, 1\} && \text{for } i \in V \quad (\text{station either open or closed})
 \end{aligned}$$



© tikz/Illustration1.pdf<sup>3</sup>

**Figure 16.2: Layout of districts and possible locations of fire stations.**

**Set Covering - Matrix description:**

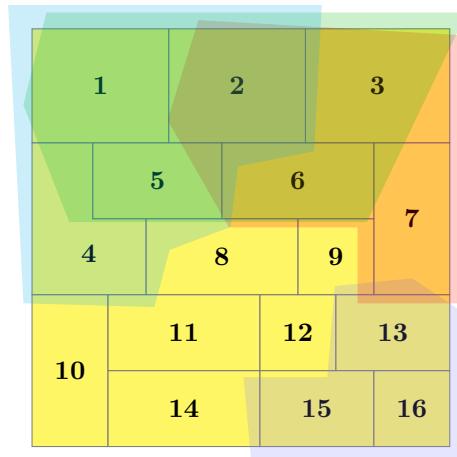
*NP-Complete*

Given a non-negative matrix  $A \in \{0, 1\}^{m \times n}$ , a non-negative vector, and an objective vector  $c \in \mathbb{R}^n$ , the

<sup>3</sup>tikz/Illustration1.pdf, from tikz/Illustration1.pdf. tikz/Illustration1.pdf, tikz/Illustration1.pdf.

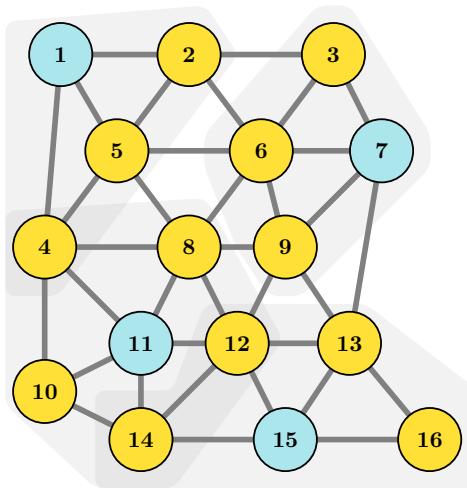
<sup>4</sup>tikz/Illustration2.pdf, from tikz/Illustration2.pdf. tikz/Illustration2.pdf, tikz/Illustration2.pdf.

<sup>5</sup>tikz/Illustration3.pdf, from tikz/Illustration3.pdf. tikz/Illustration3.pdf, tikz/Illustration3.pdf.



© tikz/Illustration2.pdf<sup>4</sup>

**Figure 16.3:** Set cover representation of fire station problem. For example, choosing district 16 to have a fire station covers districts 13, 15, and 16.



© tikz/Illustration3.pdf<sup>5</sup>

**Figure 16.4:** Graph representation of fire station problem. Every node is connected to a chosen node by an edge

set cover problem is

$$\begin{aligned}
 & \max c^\top x \\
 & \text{s.t. } Ax \geq 1 \\
 & \quad x \in \{0, 1\}^n.
 \end{aligned} \tag{16.3}$$

**Example: Vertex Cover with matrix**

An alternate way to solve ?? is to define the *adjacency matrix*  $A$  of the graph. The adjacency matrix is a  $|E| \times |V|$  matrix with  $\{0, 1\}$  entries. The each row corresponds to an edge  $e$  and each column corresponds to a node  $v$ . For an edge  $e = (u, v)$ , the corresponding row has a 1 in columns corresponding to the nodes  $u$  and  $v$ , and a 0 everywhere else. Hence, there are exactly two 1's per row. Applying the formulation above in Set Covering - Matrix description models the problem.

### 16.3.1. Covering (Generalizing Set Cover)

---

We could also allow for a more general type of set covering where we have non-negative integer variables and a right hand side that has values other than 1.

**Covering:**

*NP-Complete*

Given a non-negative matrix  $A \in \mathbb{Z}_+^{m \times n}$ , a non-negative vector  $b \in \mathbb{Z}^m$ , and an objective vector  $c \in \mathbb{R}^n$ , the set cover problem is

$$\begin{aligned} & \max \quad c^\top x \\ & \text{s.t..} \quad Ax \geq b \\ & \quad x \in \mathbb{Z}_+^n. \end{aligned} \tag{16.4}$$

## 16.4 Assignment Problem

---

The *assignment problem* (machine/person to job/task assignment) seeks to assign tasks to machines in a way that is most efficient. This problem can be thought of as having a set of machines that can complete various tasks (textile machines that can make t-shirts, pants, socks, etc) that require different amounts of time to complete each task, and given a demand, you need to decide how to alloacte your machines to tasks.

Alternatively, you could be an employer with a set of jobs to complete and a list of employees to assign to these jobs. Each employee has various abilities, and hence, can complete jobs in differing amounts of time. And each employee's time might cost a different amout. How should you assign your employees to jobs in order to minimize your total costs?

**Assignment Problem:**

Given  $m$  machines and  $n$  jobs, find a least cost assignment of jobs to machines. The cost of assigning job  $j$  to machine  $i$  is  $c_{ij}$ .

Include picture and example data

### Example: Machine Assignment

Code

#### Sets:

- Let  $I = \{0, 1, 2, 3\}$  set of machines.
- Let  $J = \{0, 1, 2, 3\}$  be the set of tasks.

#### Parameters:

- $c_{ij}$  - the cost of assigning machine  $i$  to job  $j$

#### Variables:

- Let

$$x_{ij} = \begin{cases} 1 & \text{if machine } i \text{ assigned to job } j \\ 0 & \text{otherwise.} \end{cases}$$

#### Model:

$$\begin{aligned} \min \quad & \sum_{i \in I, j \in J} c_{ij} x_{ij} && \text{(Minimize cost)} \\ \text{s.t.} \quad & \sum_{i \in I} x_{ij} = 1 && \text{for all } j \in J \quad \text{(All jobs are assigned one machine)} \\ & \sum_{j \in J} x_{ij} = 1 && \text{for all } i \in I \quad \text{(All machines are assigned to a job)} \\ & x_{ij} \in \{0, 1\} \forall i \in I, j \in J \end{aligned}$$

## 16.5 Facility Location

### Resources

- Wikipedia - Facility Location Problem
- See GUROBI Modeling Examples - Facility Location.

The basic model of the facility location problem is to determine where to place your stores or facilities in order to be close to all of your customers and hence reduce the costs transportation to your customers. Each customer is known to have a certain demand for a product, and each facility has a capacity on how

much of that demand it can satisfy. Furthermore, we need to consider the cost of building the facility in a given location.

This basic framework can be applied in many types of problems and there are a number of variants to this problem. We will address two variants: the *capacitated facility location problem* and the *uncapacitated facility location problem*.

### 16.5.1. Capacitated Facility Location

---

#### Capacitated Facility Location:

##### *NP-Complete*

Given costs connections  $c_{ij}$  and fixed building costs  $f_i$ , demands  $d_j$  and capacities  $u_i$ , the capacitated facility location problem is

##### Sets:

- Let  $I = \{1, \dots, n\}$  be the set of facilities.
- Let  $J = \{1, \dots, m\}$  be the set of customers.

##### Parameters:

- $f_i$  - the cost of opening facility  $i$ .
- $c_{ij}$  - the cost of fulfilling the complete demand of customer  $j$  from facility  $i$ .
- $u_i$  - the capacity of facility  $i$ .
- $d_j$  - the demand by customer  $j$ .

##### Variables:

- Let

$$x_i = \begin{cases} 1 & \text{if we open facility } i, \\ 0 & \text{otherwise.} \end{cases}$$

- Let  $y_{ij} \geq 0$  be the fraction of demand of customer  $j$  satisfied by facility  $i$ .

##### Model:

$$\begin{aligned} \min & \sum_{i=1}^n \sum_{j=1}^m c_{ij}y_{ij} + \sum_{i=1}^n f_i x_i && \text{(total cost)} \\ \text{s.t.} & \sum_{i=1}^n y_{ij} = 1 \text{ for all } j = 1, \dots, m && \text{(assign demand to facility)} \\ & \sum_{j=1}^m d_j y_{ij} \leq u_i x_i \text{ for all } i = 1, \dots, n && \text{(capacity of facility } i) \\ & y_{ij} \geq 0 \text{ for all } i = 1, \dots, n \text{ and } j = 1, \dots, m && \text{(nonnegative fraction of demand satisfied)} \\ & x_i \in \{0, 1\} \text{ for all } i = 1, \dots, n && \text{(open/not open facility)} \end{aligned}$$

### 16.5.2. Uncapacitated Facility Location

#### Uncapacitated Facility Location:

*NP-Complete*

Given costs connections  $c_{ij}$  and fixed building costs  $f_i$ , the uncapacitated facility location problem is

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^m c_{ij} z_{ij} + \sum_{i=1}^n f_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^n z_{ij} = 1 \text{ for all } j = 1, \dots, m \\ & \sum_{j=1}^m z_{ij} \leq M x_i \text{ for all } i = 1, \dots, n \\ & z_{ij} \in \{0, 1\} \text{ for all } i = 1, \dots, n \text{ and } j = 1, \dots, m \\ & x_i \in \{0, 1\} \text{ for all } i = 1, \dots, n \end{aligned} \tag{16.1}$$

Here  $M$  is a large number and can be chosen as  $M = m$ , but could be refined smaller if more context is known.

## 16.6 Basic Modeling Tricks - Using Binary Variables

#### Resources

- JuMP tips and tricks
- Mosek Modeling Cookbook

In this section, we describe ways to model a variety of constraints that commonly appear in practice. The goal is changing constraints described in words to constraints defined by math.

Binary variables can allow you to model many types of constraints. We discuss here various logical constraints where we assume that  $x_i \in \{0, 1\}$  for  $i = 1, \dots, n$ . We will take the meaning of the variable to be selecting an item.

1. If item  $i$  is selected, then item  $j$  is also selected.

$$x_i \leq x_j \tag{16.1}$$

- (a) If any of items  $1, \dots, 5$  are selected, then item  $6$  is selected.

$$x_1 + x_2 + \dots + x_5 \leq 5 \cdot x_6 \tag{16.2}$$

Alternatively!

$$x_i \leq x_6 \quad \text{for all } i = 1, \dots, 5 \tag{16.3}$$

2. If item  $j$  is not selected, then item  $i$  is not selected.

$$x_i \leq x_j \quad (16.4)$$

- (a) If item  $j$  is not selected, then all items  $1, \dots, i$  are not selected.

$$x_1 + x_2 + \dots + x_i \leq i \cdot x_j \quad (16.5)$$

3. If item  $j$  is not selected, then item  $i$  is not selected.

$$x_i \leq x_j \quad (16.6)$$

4. Either item  $i$  is selected or item  $j$  is selected, but not both.

$$x_i + x_j = 1 \quad (16.7)$$

5. Item  $i$  is selected or item  $j$  is selected or both.

$$x_i + x_j \geq 1 \quad (16.8)$$

6. If item  $i$  is selected, then item  $j$  is not selected.

$$x_j \leq (1 - x_i) \quad (16.9)$$

7. At most one of items  $i, j$ , and  $k$  are selected.

$$x_i + x_j + x_k \leq 1 \quad (16.10)$$

8. At most two of items  $i, j$ , and  $k$  are selected.

$$x_i + x_j + x_k \leq 2 \quad (16.11)$$

9. Exactly one of items  $i, j$ , and  $k$  are selected.

$$x_i + x_j + x_k = 1 \quad (16.12)$$

These tricks can be connected to create different function values.

### Example 16.2: Variable takes one of three values

Suppose that the variable  $x$  should take one of the three values  $\{4, 8, 13\}$ . This can be modeled using three binary variables as

$$x = 4z_1 + 8z_2 + 13z_3$$

$$z_1 + z_2 + z_3 = 1$$

$$z_i \in \{0, 1\} \text{ for } i = 1, 2, 3.$$

As a convenient addition, if we want to add the possibility that it takes the value 0, then we can

model this as

$$\begin{aligned}x &= 4z_1 + 8z_2 + 13z_3 \\z_1 + z_2 + z_3 &\leq 1 \\z_i &\in \{0, 1\} \text{ for } i = 1, 2, 3.\end{aligned}$$

We can also model variable increases at different amounts.

### Example 16.3: Discount for buying more

Suppose you can choose to buy 1, 2, or 3 units of a product, each with a decreasing cost. The first unit is \$10, the second is \$5, and the third unit is \$3.

$$\begin{aligned}x &= 10z_1 + 5z_2 + 3z_3 \\z_1 &\geq z_2 \geq z_3 \\z_i &\in \{0, 1\} \text{ for } i = 1, 2, 3.\end{aligned}$$

Here,  $z_i$  represents if we buy the  $i$ th unit. The inequality constraints impose that if we buy unit  $j$ , then we must buy all units  $i$  with  $i < j$ .

#### 16.6.1. Connecting to continuous variables

Let  $x_i \geq 0$  and  $y_i \in \{0, 1\}$  for all  $i = 1, \dots, n$ .

1. If  $x_i > 0$ , then  $y_i = 1$ .

$$x_i \leq M y_i \tag{16.13}$$

where  $M$  is a sufficiently large upper bound on the variable  $x_i$ .

2. If  $x_i = 0$ , then  $y_i = 0$ .

This is harder to model! Alternatively, we try modeling "if  $x_i$  is sufficiently small, then  $y_i = 0$ . For instance, if  $x_i \leq 0.0000001$ , then  $y_i = 0$ . This can be modeled as

$$x_i - 0.0000001 \geq y_i - 1. \tag{16.14}$$

3. If  $y_i = 1$ , then  $x_i \geq 5$

$$5y_i \leq x_i. \tag{16.15}$$

## 16.6.2. Exact absolute value

---

Suppose we need to model an exact equality

$$|x| = t$$

It defines a non-convex set, hence it is not conic representable. If we split  $x$  into positive and negative part  $x = x^+ - x^-$ , where  $x^+, x^- \geq 0$ , then  $|x| = x^+ + x^-$  as long as either  $x^+ = 0$  or  $x^- = 0$ . That last alternative can be modeled with a binary variable, and we get a model of :

$$\begin{aligned} x &= x^+ - x^- \\ t &= x^+ + x^- \\ 0 &\leq x^+, x^- \\ x^+ &\leq Mz \\ x^- &\leq M(1-z) \\ z &\in \{0, 1\} \end{aligned}$$

where the constant  $M$  is an a priori known upper bound on  $|x|$  in the problem.

### 16.6.2.1. Exact 1-norm

---

We can use the technique above to model the exact  $\ell_1$ -norm equality constraint

$$\sum_{i=1}^n |x_i| = c$$

where  $x \in \mathbb{R}^n$  is a decision variable and  $c$  is a constant. Such constraints arise for instance in fully invested portfolio optimizations scenarios (with short-selling). As before, we split  $x$  into a positive and negative part, using a sequence of binary variables to guarantee that at most one of them is nonzero:

$$\begin{aligned} x &= x^+ - x^- \\ 0 &\leq x^+, x^- \\ x^+ &\leq cz \\ x^- &\leq c(e-z), \\ \sum_i x_i^+ + \sum_i x_i^- &= c, \\ z &\in \{0, 1\}^n, x^+, x^- \in \mathbb{R}^n \end{aligned}$$

### 16.6.2.2. Maximum

---

The exact equality  $t = \max \{x_1, \dots, x_n\}$  can be expressed by introducing a sequence of mutually exclusive indicator variables  $z_1, \dots, z_n$ , with the intention that  $z_i = 1$  picks the variable  $x_i$  which actually achieves maximum. Choosing a safe bound  $M$  we get a model:

$$\begin{aligned} x_i &\leq t \leq x_i + M(1 - z_i), i = 1, \dots, n \\ z_1 + \dots + z_n &= 1, \\ z &\in \{0, 1\}^n \end{aligned}$$

## 16.7 Network Flow

---

Fix up this section



### 16.7.1. Example - Multicommodity Flow

---

[https://en.wikipedia.org/wiki/Multi-commodity\\_flow\\_problem](https://en.wikipedia.org/wiki/Multi-commodity_flow_problem) The **multi-commodity flow problem** is a network flow problem with multiple commodities (flow demands) between different source and sink nodes.

**PROBLEM DEFINITION** Given a flow network  $G(V, E)$ , where edge  $(u, v) \in E$  has capacity  $c(u, v)$ . There are  $k$  commodities  $K_1, K_2, \dots, K_k$ , defined by  $K_i = (s_i, t_i, d_i)$ , where  $s_i$  and  $t_i$  is the **source** and **sink** of commodity  $i$ , and  $d_i$  is its demand. The variable  $f_i(u, v)$  defines the fraction of flow  $i$  along edge  $(u, v)$ , where  $f_i(u, v) \in [0, 1]$  in case the flow can be split among multiple paths, and  $f_i(u, v) \in \{0, 1\}$  otherwise (i.e. "single path routing"). Find an assignment of all flow variables which satisfies the following four constraints:

**(1) Link capacity:** The sum of all flows routed over a link does not exceed its capacity.

$$\forall (u, v) \in E : \sum_{i=1}^k f_i(u, v) \cdot d_i \leq c(u, v)$$

**(2) Flow conservation on transit nodes:** The amount of a flow entering an intermediate node  $u$  is the same that exits the node.

$$\sum_{w \in V} f_i(u, w) - \sum_{w \in V} f_i(w, u) = 0 \quad \text{when } u \neq s_i, t_i$$

**(3) Flow conservation at the source:** A flow must exit its source node completely.

$$\sum_{w \in V} f_i(s_i, w) - \sum_{w \in V} f_i(w, s_i) = 1$$

**(4) Flow conservation at the destination:** A flow must enter its sink node completely.

$$\sum_{w \in V} f_i(w, t_i) - \sum_{w \in V} f_i(t_i, w) = 1$$

## 16.7.2. Corresponding optimization problems

---

**Load balancing** is the attempt to route flows such that the utilization  $U(u, v)$  of all links  $(u, v) \in E$  is even, where

$$U(u, v) = \frac{\sum_{i=1}^k f_i(u, v) \cdot d_i}{c(u, v)}$$

The problem can be solved e.g. by minimizing  $\sum_{u, v \in V} (U(u, v))^2$ . A common linearization of this problem is the minimization of the maximum utilization  $U_{max}$ , where

$$\forall (u, v) \in E : U_{max} \geq U(u, v)$$

In the **minimum cost multi-commodity flow problem**, there is a cost  $a(u, v) \cdot f(u, v)$  for sending a flow on  $(u, v)$ . You then need to minimize

$$\sum_{(u, v) \in E} \left( a(u, v) \sum_{i=1}^k f_i(u, v) \right)$$

In the **maximum multi-commodity flow problem**, the demand of each commodity is not fixed, and the total throughput is maximized by maximizing the sum of all demands  $\sum_{i=1}^k d_i$

## 16.7.3. Relation to other problems

---

The minimum cost variant of the multi-commodity flow problem is a generalization of the minimum cost flow problem (in which there is merely one source  $s$  and one sink  $t$ ). Variants of the circulation problem are generalizations of all flow problems. That is, any flow problem can be viewed as a particular circulation problem.<sup>6</sup>

---

6

#### 16.7.4. Usage

---

Routing and wavelength assignment (RWA) in optical burst switching of Optical Network would be approached via multi-commodity flow formulas.

## 16.8 Transportation Problem

---

Add discussion of transportation problem and picture.

Youtube! - TRANSPORTATION PROBLEM with PuLP in PYTHON

Notebook: Solution with Pyomo

## 16.9 Other examples

---

- Sudoku
- AIMMS - Employee Training
- AIMMS - Media Selection
- AIMMS - Diet Problem
- AIMMS - Farm Planning Problem
- AIMMS - Pooling Probem
- INFORMS - Impact
- INFORMS - Success Story - Bus Routing

## 16.10 Notes from AIMMS modeling book.

---

- AIMMS - Practical guidelines for solving difficult MILPs
- AIMMS - Linear Programming Tricks
- AIMMS - Formulating Optimization Models
- AIMMS - Practical guidelines for solving difficult linear programs

### 16.10.1. Further Topics

---

- Precedence Constraints

## 16.11 MIP Solvers and Modeling Tools

---

- AMPL
- GAMS
- AIMMS
- Python-MIP
- Pyomo
- PuLP
- JuMP
- GUROBI
- CPLEX (IBM)
- Express
- SAS
- Coin-OR (CBC, CLP, IPOPT)
- SCIP

# 17. Algorithms and Complexity

---

## Outcomes

- (a) *Describe asymptotic growth of functions using Big-O notation.*
- (b) *Analyze algorithms for the asymptotic runtime.*
- (c) *Classify problem types with respect to notions of how difficult they are to solve.*

## Resources

- *MIT Lecture Notes - Big O*
- *Youtube! - P versus NP*

How long will an algorithm take to run? How difficult might it be to solve a certain problem? Is the knapsack problem easier to solve than the traveling salesman problem? Or the matching problem? How can we compare the difficulty to solve these problems?

We will understand these questions through complexity theory. We will first use "Big-O" notation to simplify asymptotic analysis of the runtime of algorithms and the size of the input data of an algorithm.

We will then classify problem types as being either easy (in the class P) or probably very hard (in the class NP Hard). We will also learn about the problem classes NP, and NP-Complete.

To begin, watch these videos ([Video 1](#), [Video 2](#)) about sorting algorithms. Notice how a different algorithm can produce a much different number of steps needed to solve the problem. The first video explains bubble sort and quick sort. The second video explains insertion sort, and then described the analysis of the algorithms (how many comparisons they make as the number of balls to sort grows). Pay attention to this analysis as this is very crucial in this module.

This video is a great introduction to the basic idea of Big-O notation. We will go over the more formal definition.

Here are two great videos about P versus NP ([Video 1](#), [Video 2](#)).

## 17.1 Big-O Notation

---

We begin with some definitions that relate the rate of growth of functions. The functions we will look at in the next section will describe the runtime of an algorithm.

**Example 17.1: Relations of functions**

We want to understand the asymptotic growth of the following functions:

- $f(n) = n^2 + 5$ ,
- $g(n) = n^3 - 10n^2 - 10$ .

When we discuss asymptotic growth, we don't care so much what happens for small values of  $n$ , and instead, we want know what happens for large values of  $n$ .

Notice that

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0. \quad (17.1)$$

This is because as  $n$  gets large,  $g(n) \gg f(n)$ . However, this does not preclude the possibility that  $g(n) < f(n)$  for some small values of  $n$ , (i.e.,  $n = 1, 2, 3$ ).

We can, however see that  $g(n) > f(n)$  whenever  $n \geq N := 20$  (it is probably true for a smaller value of  $n$ , but for the sake of the analysis, we don't care).

Thus, we want to say that  $g(n)$  grows faster than  $f(n)$ .

**Example 17.2: Asymptotic Technicality**

It may be that we consider functions that are not strictly increasing after some point. For example,

- $f(n) = \sin(n)(n^2 + 5)$ ,
- $g(n) = 10n^2 - 10$ .

Still, we would like to say that  $f(n)$  is bounded somehow by  $g(n)$ . But! The limit  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$  does not exist!

For this, we use the limsup notation. That is, we notice that

$$\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty. \quad (17.2)$$

This completely captures our goal here. However, we will give an alternative definition that allows us to not have to think about the limsup.

**Definition 17.3: Big-O**

For two functions  $f(n)$  and  $g(n)$ , we say that  $f(n) = O(g(n))$  if there exist positive constants  $c$  and  $n_0$  such that

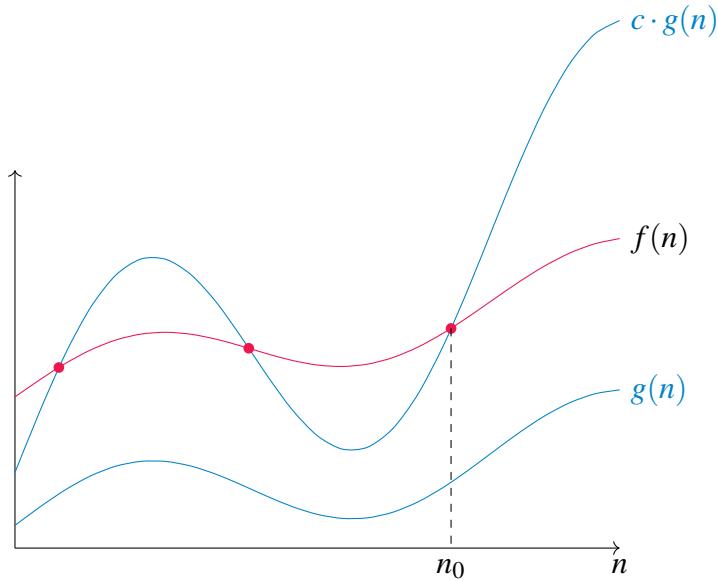
$$0 \leq f(n) \leq c g(n) \quad \text{for all } n \geq n_0. \quad (17.3)$$

**Example 17.4:**

Consider  $f(n) = 5n^2 + 10n + 7$  and  $g(n) = n^2$ . We want to show that  $f(n) = O(g(n))$ .

Let's try  $c = 22$  and  $n_0 = 1$ . We need to show that Equation 17.3 is satisfied.

Note first that we always have



**Figure 17.1: Example of Big-O notation:**  $f(n) = O(g(n))$ . We see that for all  $n \geq n_0$ , we have  $c \cdot g(n) \geq f(n)$ .

$$(a) n^2 \leq n^2 \text{ and therefore } 5n^2 \leq 5n^2$$

Note that if  $n \geq 1$ , then

$$(a) n \leq n^2 \text{ and therefore } 10n \leq 10n^2$$

$$(b) 1 \leq n^2 \text{ and therefore } 7 \leq 7n^2$$

Since all inequalities 1, 2, and 3 are valid for  $n \geq 1$ , by adding them, we obtain a new inequality that is also valid for  $n \geq 1$ , which is

$$5n^2 + 10n + 7 \leq 5n^2 + 10n^2 + 7n^2 \quad \text{for all } n \geq 1, \quad (17.4)$$

$$\Rightarrow 5n^2 + 10n + 7 \leq 22n^2 \quad \text{for all } n \geq 1. \quad (17.5)$$

Hence, we have shown that Equation 17.3 holds for  $c = 22$  and  $n_0 = 1$ . Hence  $f(n) = O(g(n))$ .

Correct uses:

- $2^n + n^5 + \sin(n) = O(2^n)$
- $2^n = O(n!)$
- $n! + 2^n + 5n = O(n!)$
- $n^2 + n = O(n^3)$
- $n^2 + n = O(n^2)$
- $\log(n) = O(n)$
- $10\log(n) + 5 = O(n)$

Notice that not all examples above give a tight bound on the asymptotic growth. For instance,  $n^2 + n = O(n^3)$  is true, but a tighter bound is  $n^2 + n = O(n^2)$ .

In particular, the goal of big O notation is to give an upper bound on the asymptotic growth of a

function. But we would prefer to give a strong upper bound as opposed to a weak upper bound. For instance, if you order a package online, you will typically be given a bound on the latest date that it will arrive. For example, if it will arrive within a week, you might be guaranteed that it will arrive by next Tuesday. This sounds like a reasonable bound. But if instead, they tell you it will arrive before 1 year from today, this may not be as useful information. In the case of big O notation, we would like to give a least upper bound that most simply describes the growth behavior of the function.

In that example,  $n^2 + n = O(n^2)$ , this literally means that there is some number  $c$  and some value  $n_0$  that  $n^2 + n \leq cn^2$  for all  $n \geq n_0$ , that is, for all values of  $n_0$  larger than  $n$ , the function  $cn^2$  dominates  $n^2 + n$ .

For example, a valid choice is  $c = 2$  and  $n_0 = 1$ . Then it is true that  $n^2 + n \leq 2n^2$  for all  $n \geq 1$ .

But it is also true that  $n^2 + n = O(n^3)$ . For example, a valid choice is again  $c = 2$  and  $n_0 = 1$ , then  $n^2 + n \leq 2n^3$  for all  $n \geq 1$ .

In this example,  $O(n^3)$  is the case where the internet tells you the package will arrive before 1 year from today. The bound is true, but it is not as useful information as we would like to have. Let's compare these upper bounds. Let  $f(n) = n^2 + n$ ,  $g(n) = 2n^2$ ,  $h(n) = 2n^3$ .

Then we have

$n = 10 .$	$n = 100 .$	$n = 1000 .$	$n = . 10000$
$f(n)$	110,	10100,	1001000,
$g(n)$	200,	20000,	2000000,
$h(n) .$	2000,	2000000,	200000000000

So, here we see that  $g(n)$  and  $h(n)$  are both upper bounds on  $f(n)$ , but the nice part about  $g(n)$  is that is growing at a similar rate to  $f(n)$ . In particular, it is always within a factor of 2 of  $f(n)$ .

Alternatively, the bound  $h(n)$  is true, but it grows so much faster than  $f(n)$  that is doesn't give a good idea of the asymptotic growth of  $f(n)$ .

Some common classes of functions:

$O(1)$	Constant
$O(\log(n))$	Logarithmic
$O(n)$	Linear
$O(n^c)$ (for $c > 1$ )	Polynomial
$O(c^n)$ (for $c > 1$ )	Exponential

---

<sup>1</sup>time-of-algorithms, from time-of-algorithms. time-of-algorithms, time-of-algorithms.

**Exponential Time Algorithms do not currently solve reasonable-sized problems in reasonable time.**

Polynomial	$\log n$	3	4
	$n$	10	20
	$n \log n$	33	86
	$n^2$	100	400
	$n^3$	1,000	8,000
	$n^5$	100,000	3,200,000
	$n^{10}$	10,000,000,000	10,240,000,000,000
Exponential	$n$	10	20
	$n^{\log n}$	2,099	419,718
	$2^n$	1,024	1,048,576
	$5^n$	9,765,625	95,367,431,640,625
	$n!$	3,628,800	2,432,902,008,176,640,000
	$n^n$	10,000,000,000	104,857,600,000,000,000,000,000,000

© time-of-algorithms<sup>1</sup>

Figure 17.2: time-of-algorithms

## 17.2 Algorithms - Example with Bubble Sort

---

The following definition comes from Merriam-Webster's dictionary.

### Definition 17.5: Algorithm

An algorithm is a procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation; broadly: a step-by-step procedure for solving a problem or accomplishing some end.

### 17.2.1. Sorting

---

#### Resources

- [Wikipedia](#)

The problem of sorting a list is a simple problem to think about that has many algorithms to consider. We will describe one such algorithm: Bubble Sort.

#### Sorting Problem:

*Polynomial time (P)*

Given a list of numbers  $(x_1, \dots, x_n)$  sort them into increasing order.

### Example 17.6: Sorting Problem

*Suppose you have the list of number  $(10, 35, 9, 4, 15, 22)$ .  
The sorted list of numbers is  $(4, 9, 10, 15, 22, 35)$ .*

What process or algorithm should we use to compute the sorted list?

#### Bubble sort algorithm:

The *Bubble Sort* algorithm works as follows:

- (a) Compare numbers in position 1 and 2. If numbers are out of order, then swap them.
  - (b) Next, compare numbers in position 2 and 3. If numbers are out of order, then swap them.
  - (c) Continue this process of comparing subsequent numbers until you get to the end of the list (and compare numbers in position  $n - 1$  and  $n$ ).
- Now the largest number should be in last position!
- (d) If no swaps had to be made, then the whole list is sorted!
  - (e) Otherwise, if any swaps were needed, then set the last number aside, and start over from the beginning and sort the remaining list.

#### Example: Bubble Sort

Let try using Bubble Sort to sort this list.

##### First pass through the list.

Step 1:

$$(10, 35, 9, 4, 15, 22) \rightarrow (10, 35, 9, 4, 15, 22)$$

Step 2:

$$(10, 35, 9, 4, 15, 22) \rightarrow (10, 9, 35, 4, 15, 22)$$

Step 3:

$$(10, 9, 35, 4, 15, 22) \rightarrow (10, 9, 4, 35, 15, 22)$$

Step 4:

$$(10, 9, 4, 35, 15, 22) \rightarrow (10, 9, 4, 15, 35, 22)$$

Step 5:

$$(10, 9, 4, 15, 35, 22) \rightarrow (10, 9, 4, 15, 22, 35)$$

Now 35 is in the last spot!

**Second pass through the list**

Step 1:

$$(10, 9, 4, 15, 22 | 35) \rightarrow (9, 10, 4, 15, 22 | 35)$$

Step 2:

$$(9, 10, 4, 15, 22 | 35) \rightarrow (9, 4, 10, 15, 22 | 35)$$

Step 3:

$$(9, 4, 10, 15, 22 | 35) \rightarrow (9, 4, 10, 15, 22 | 35)$$

Step 4:

$$(9, 4, 10, 15, 22 | 35) \rightarrow (9, 4, 10, 15, 22 | 35)$$

Now 22 is in the correct spot!

**Third pass through the list**

Step 1:

$$(9, 4, 10, 15, | 22, 35) \rightarrow (4, 9, 10, 15, | 22, 35)$$

Step 2:

$$(4, 9, 10, 15, | 22, 35) \rightarrow (4, 9, 10, 15, | 22, 35)$$

Step 3:

$$(4, 9, 10, 15, | 22, 35) \rightarrow (4, 9, 10, 15, | 22, 35)$$

**Fourth pass through the list**

Step 1:

$$(4, 9, 10, | 15, 22, 35) \rightarrow (4, 9, 10, | 15, 22, 35)$$

Step 2:

$$(4, 9, 10, | 15, 22, 35) \rightarrow (4, 9, 10, | 15, 22, 35)$$

No swaps were necessary! We must be done!

**How many comparisons were needed?**

- In the first pass, we needed 5 comparisions
- In the second pass, we needed 4 comparisions
- In the third pass, we needed 3 comparisions
- In the fourth pass, we needed 2 comparisions

Thus we used

$$5 + 4 + 3 + 2 = 14$$

comparisons.

**Example: Worst Case Analysis****What is the worst case number of comparisons?**

For a list of  $n$  numbers, the worst case would be

$$(n - 1) + (n - 2) + \cdots + 2 + 1.$$

Notice that we can compute thus sum exactly in a shorter form. To do so, let's count the number of pairs that we can get to add up to  $n$ . Suppose that  $n$  is an even number.

$$\begin{aligned} (n - 1) + 1 &= n \\ (n - 2) + 2 &= n \\ (n - 3) + 3 &= n \\ &\vdots \\ (n/2 + 1) + (n/2 - 1) &= n \end{aligned}$$

Then we also have the number  $n/2$  left over.

Adding all this up, we have  $(n/2 + 1)$  pairs that add up to  $n$ , plus one  $n/2$  left over.

Hence, the sum is

$$n\left(\frac{n}{2} - 1\right) + \frac{n}{2} = \frac{n(n - 1)}{2}.$$

Hence, we have proved that

$$\sum_{i=1}^{n-1} i = \frac{n(n - 1)}{2}.$$

Since we just care about the Big-O expression, we can upper bound this by  $O(n^2)$ .

Hence, we will say that

These can be verified experimentally as seen in the following plot. The random case grows quadratically just as the worst case does.

There are some other relations that hold:

---

<sup>2</sup>[bubble-sort-computational-example](#), from [bubble-sort-computational-example](#). [bubble-sort-computational-example](#), [bubble-sort-computational-example](#).

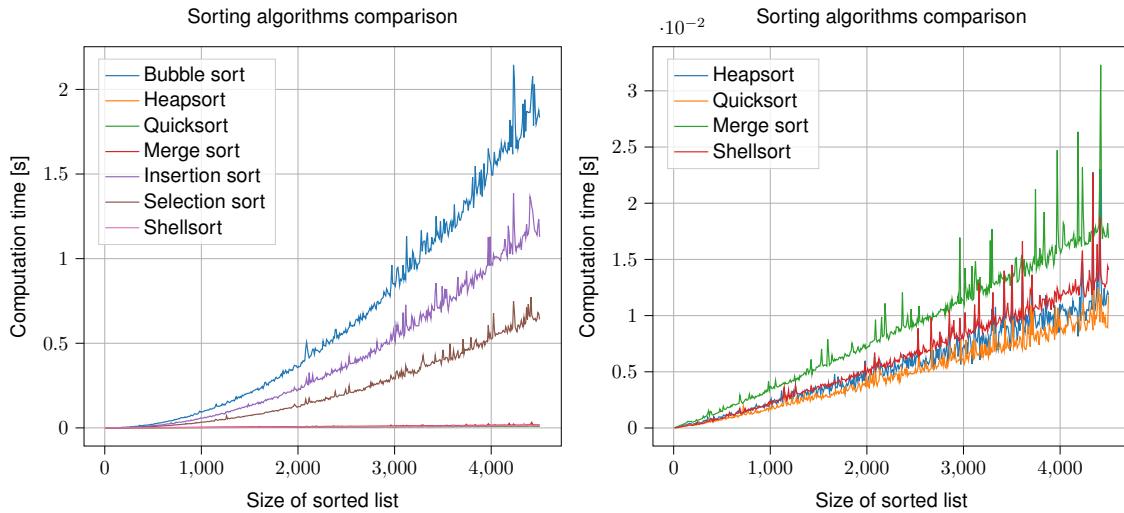


Figure 17.3: Comparison of runtimes of sorting algorithms.

### Time elapsed in computer for bubble sort

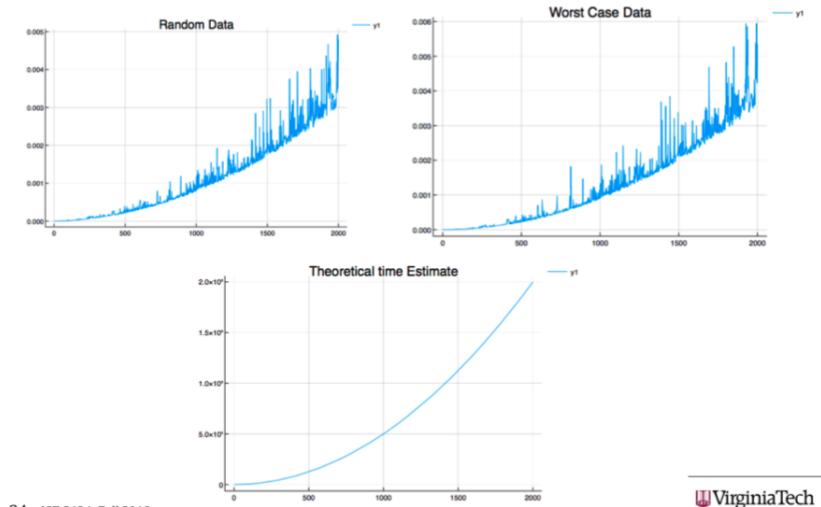


Figure 17.4: bubble-sort-computational-example

### Theorem 17.7: Summations

- $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ .
- $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$ .
- $\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$ .

There are other formulas, but they get more complicated. In general, we know that

$$\bullet \sum_{i=1}^n i^k = O(n^{k+1}).$$

## 17.3 Problem, instance, size

---

### 17.3.1. Problem, instance

---

**Definition 17.8: Problem**

*Is a generic question/task that needs to be answered/solved.*

*A problem is a “collection of instances” (see below).*

A particular realization of a problem is defined next.

**Definition 17.9: Instance**

*An instance is a specific case of a problem. For example, for the problem of sorting, an instance we saw already is (4, 9, 10, 15, 22, 35).*

### 17.3.2. Format and examples of problems/instances

---

A problem is an abstract concept. We will write problems in the following format:

- **INPUT:** Generic data/instance.
- **OUTPUT:** Question to be answered and/or task to be performed with the data.

**Examples of problems/instances:**

- Typical problems: optimization problems, decision problems, feasibility problems.
- LP and IP feasibility, TSP, IP minimization, Maximum cardinality independent set.

### 17.3.3. Size of an instance

---

The size of an instance is the *amount of information* required to represent the instance (in the computer). Typically, this information is bounded by the quantity of numbers in the problem and the size of the numbers.

**Example 17.10: Size of Sorting Problem**

*Most of the time, we will think of the size of the sorting problem as*

$$n,$$

*which is the number of numbers taht we need to sort.*

*However, we should also keep in mind that the size of the numbers is also important. That is, if the numbers we are asked to sort take up 1 gigabyte of space to write down, then merely comparing these numbers could take a long time.*

So to be more precise, the size of the problem is

$\# \text{ of bits to encode the problem}$

which can be upper bounded by

$$n\phi_{\max}$$

where  $\phi_{\max}$  is the maximum encoding size of a number given in the data.

For the sake of simplicity, we will typically ignore the size  $\phi_{\max}$  in our complexity discussion. A more rigorous discussion of complexity will be given in later (advanced) parts of the book.

### Example 17.11: Size of Matching Problem

The matching problem is presented as a graph  $G = (V, E)$  and a set of costs  $c_e$  for all  $e \in E$ . Thus, the size of the problem can be described as  $|V| + |E|$ , that is, in terms of the number of nodes and the number of edges in the graph.

## 17.4 Complexity Classes

In this subsection we will discuss the complexity classes P, NP, NP-Complete, and NP-Hard. These classes help measure how difficult a problem is. **Informally**, these classes can be thought of as

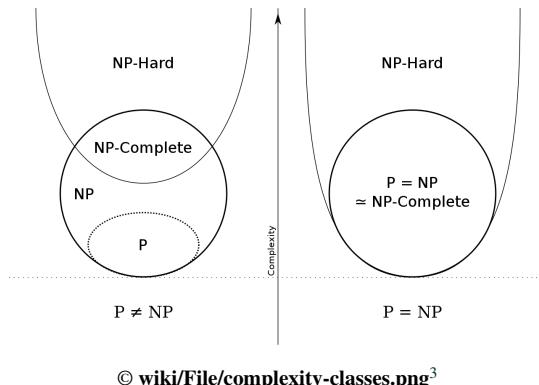
- P - the class of efficiently solvable problems
- NP - the class of efficiently checkable problems
- NP-Hard - the class of problems that can solve any problem in NP
- NP-Complete - the class of problems that are in both NP and are NP-Hard.

It is not known if P is the same as NP, but it is conjectured that these are very different classes. This would mean that the NP-Hard problems (and NP-Complete problems) are necessarily much more difficult than the problems in P. See Figure 17.5 .

We will now discuss these classes more formally.

---

<sup>3</sup>wiki/File/complexity-classes.png, from wiki/File/complexity-classes.png.      wiki/File/complexity-classes.png,  
wiki/File/complexity-classes.png.



**Figure 17.5: Complexity class possibilities. Most academics agree that the case  $P \neq NP$  is more likely.**

### 17.4.1. P

#### Definition 17.12: P

*P* is the class of polynomially solvable problems. *P* contains all problems for which there exists an algorithm that solves the problem in a run time bounded by a polynomial of input size. That is,  $O(n^c)$  for some constant  $c$ .

#### Example 17.13: Sorting

The sorting problem can be solved in  $O(n^2)$  time. Thus, this problem is in *P*

#### Example 17.14: Complexity Minimum Spanning Tree

The minimum size spanning tree problem is in *P*. It can be solved, for instance, by Prim's algorithm, which runs in time  $O(m \log n)$ , where  $m$  is the number of edges in the graph and  $n$  is the number of nodes in the graph.

#### Example 17.15: Complexity Linear Programming

Linear programming is in *P*. It can be solved by interior point methods in  $O(n^{3.5}\phi)$  where  $\phi$  represents the number of binary bits that are required to encode the problem. These bits describe the matrix  $A$ , and vectors  $c$  and  $b$  that define the linear program.

## 17.4.2. NP

In this section, we will be more specific about the types of problems we want to consider. In particular, we will consider *decisions problems*. These are problems where we only request an answer of "yes" or "no".

We can rephrase maximization problems as problems that ask "does there exist a solution with objective value greater than some number?"

### Example 17.16: Maximum Matching as a decisions problem

**Input:** A graph  $G = (V, E)$  with weights  $w_e$  for  $e \in E$  and an objective goal  $W$ .

Does there exist a matching with objective value greater than  $W$ ?

**Output:** Either "yes" or "no".

We can now define the class of NP.

### Definition 17.17: The class NP

Is the set of all decision problems for which a YES answer for a particular instance can be verified in polytime when provided a certificate.

A certificate can be any additional information to help convince someone of a solution. This should be describable in a compact way (polynomial in the size of the data). Typically the certificate is simply a feasible solution.

#### Examples:

- All problems in  $\mathcal{P}$
- Integer programming
- TSP
- Binary knapsack
- Maximum independent set
- Subset sum
- Partition
- SAT,  $k$ -SAT
- Clique

Thus, to show that a problem is in NP, you must do the following:

- Describe a format for a certificate to the problem.
- Show that given such a certificate, it is easy to verify the solution to the problem.

### Example 17.18

Integer Linear Programming is in NP. More explicitly, the feasibility question of  
"Does there exist an integer vector  $x$  that satisfies  $Ax \leq b$ "

is in NP.

Although it turns out to be difficult to find such an  $x$  or even prove that one exists, this problem is in NP for the following reason: if you are given a particular  $x$  and someone claims to you

that it is a feasible solution to the problem, then you can easily check if they are correct. In this case, the vector  $x$  that you were given is called a certificate.

Note that it is easy to verify if  $x$  is a solution to the problem because you just have to

- (a) Check if  $x$  is integer.
- (b) Use matrix multiplication to check if  $Ax \leq b$  holds.

### 17.4.3. Problem Reductions

---

We can compare different types of problems by showing that we can use one to solve the other.

A simple example of this is the problem *Integer Programming* and the *Matching Problem*.

Since we can model the *Matching Problem* as an *Integer Program*, then we know that we can solve the *Matching Problem* provided that we can solve *Integer Programs*.

$$\text{Matching Problem} \leq \text{Integer Programming}.$$

#### Definition 17.19: Reduction

Given two problems  $\mathcal{A}, \mathcal{B}$ , we say  $\mathcal{A}$  is reduced to  $\mathcal{B}$  (and we write  $\mathcal{A} \leq \mathcal{B}$  when we can assert that if we can solve  $\mathcal{B}$  in polynomial time, then we can also solve  $\mathcal{A}$  in polynomial time.

### 17.4.4. NP-Hard

---

The class of problems that are called *NP-Hard* are those that can be used to solve any other problem in the NP class. That is, problem A is NP-Hard provided that for any problem B in NP there is a transformation of problem B that preserves the size of the problem, up to a polynomial factor, into a new problem that problem A can be used to solve.

Here we think of “if problem A could be solved efficiently, then all problems in NP could be solved efficiently”.

More specifically, we assume that we have an oracle for problem A that runs in polynomial time. An oracle is an algorithm that for the problem that returns the solution of the problem in a time polynomial in the input. This oracle can be thought of as a magic computer that gives us the answer to the problem. Thus, we say that problem A is NP-Complete provided that given an oracle for problem A, one can solve any other problem B in NP in polynomial time.

Note: These problems are not necessarily in NP.

### 17.4.5. NP-Complete

---

The class of problems that are call *NP-Complete* are those which are in NP and also NP-Hard.

We know of many problems that are NP-Complete. For example, binary integer programming feasibility is NP-Complete. One can show that another problem is NP-complete by

- (a) showing that it can be used to solve binary integer programming feasibility,
- (b) showing that the problem is in NP.

The first problem proven to be NP-Complete is called *3-SAT* [1]. 3-SAT is a special case of the *satisfiability problem*. In a satisfiability problem, we have variables  $X_1, \dots, X_n$  and we want to assign them values as either `true` or `false`. The problem is described with *AND* operations, denoted as  $\wedge$ , with *OR* operations, denoted as  $\vee$ , and with *NOT* operations, denoted as  $\neg$ . The *AND* operation  $X_1 \wedge X_2$  returns `true` if BOTH  $X_1$  and  $X_2$  are true. The *OR* operation  $X_1 \vee X_2$  returns `true` if AT LEAST ONE OF  $X_1$  and  $X_2$  are true. Lastly, the *NOT* operation  $\neg X_1$  returns there opposite of the value of  $X_1$ .

These can be described in the following table

$$\text{true} \wedge \text{true} = \text{true} \tag{17.1}$$

$$\text{true} \wedge \text{false} = \text{false} \tag{17.2}$$

$$\text{false} \wedge \text{false} = \text{false} \tag{17.3}$$

$$\text{false} \wedge \text{true} = \text{false} \tag{17.4}$$

$$\text{true} \vee \text{true} = \text{true} \tag{17.5}$$

$$\text{true} \vee \text{false} = \text{true} \tag{17.6}$$

$$\text{false} \vee \text{false} = \text{false} \tag{17.7}$$

$$\text{false} \vee \text{true} = \text{true} \tag{17.8}$$

$$\neg \text{true} = \text{false} \tag{17.9}$$

$$\neg \text{false} = \text{true} \tag{17.10}$$

For example, **Missing code here** A *logical expression* is a sequence of logical operations on variables  $X_1, \dots, X_n$ , such that

$$(X_1 \wedge \neg X_2 \vee X_3) \wedge (X_1 \vee \neg X_3) \vee (X_1 \wedge X_2 \wedge X_3). \tag{17.11}$$

A *clause* is a logical expression that only contains the operations  $\vee$  and  $\neg$  and is not nested (with parentheses), such as

$$X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4. \tag{17.12}$$

A fundamental result about logical expressions is that they can always be reduced to a sequence of clauses that are joined by  $\wedge$  operations, such as

$$(X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4) \wedge (X_1 \vee X_2 \vee X_3) \wedge (X_2 \vee \neg X_3 \vee \neg X_4 \vee X_5). \quad (17.13)$$

The satisfiability problem takes as input a logical expression in this format and asks if there is an assignment of `true` or `false` to each variable  $X_i$  that makes the expression `true`. The 3-SAT problem is a special case where the clauses have only three variables in them.

### 3-SAT:

#### *NP-Complete*

Given a logical expression in  $n$  variables where each clause has only 3 variables, decide if there is an assignment to the variables that makes the expression `true`.

### Binary Integer Programming:

#### *NP-Complete*

Binary Integer Programming can easily be shown to be in NP, since verifying solutions to BIP can be done by checking a linear system of inequalities.

Furthermore, it can be shown to be NP-Complete since it can be used to solve 3-SAT. That is, given an oracle for BIP, since 3-SAT can be modeled as a BIP, the 3-SAT could be solved in oracle-polynomial time.

## 17.5 Problems and Algorithms

---

We will discuss the following concepts:

- Feasible solutions
- Optimal solutions
- Approximate solutions
- Heuristics
- Exact Algorithms
- Approximation Algorithms
- Complexity class relations

### 17.5.1. Matching Problem

#### Definition 17.20: Matching

Given a graph  $G = (V, E)$ , a matching is a subset  $E' \subseteq E$  such that no vertex  $v \in V$  is contained in more than one edge in  $E'$ .

A perfect matching is a matching where every vertex is connected to an edge in  $E'$ .

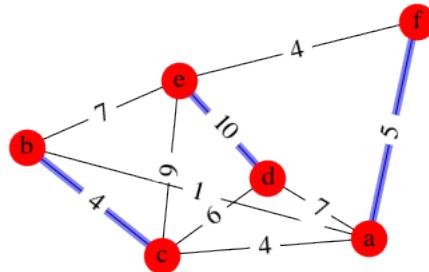
A maximal matching is a matching  $E'$  such that there is no matching  $E''$  that strictly contains it.

#### INCLUDE PICTURES OF MATCHINGS

**Figure 17.6:** Two possible matchings. On the left, we have a perfect matchings (all nodes are matched). On the right, a feasible matching, but not a perfect matching since not all nodes are matched.

#### Definition 17.21: Maximum Weight Matching

Given a graph  $G = (V, E)$ , with associated weights  $w_e \geq 0$  for all  $e \in E$ , a maximum weight matching is a matching that maximizes the sum of the weights in the matching.



© graph-for-matching-maximal<sup>4</sup>

**Figure 17.7: graph-for-matching-maximal**

<sup>4</sup>graph-for-matching-maximal, from graph-for-matching-maximal.

graph-for-matching-maximal,

### 17.5.1.1. Greedy Algorithm for Maximal Matching

The greedy algorithm iteratively adds the edge with largest weight that is feasible to add.

#### Greedy Algorithm for Maximal Matching:

Complexity:  $O(|E| \log(|V|))$

- Begin with an empty graph ( $M = \emptyset$ )
- Label the edges in the graph such that  $w(e_1) \geq w(e_2) \geq \dots \geq w(e_m)$
- For  $i = 1, \dots, m$   
If  $M \cup \{e_i\}$  is a valid matching (i.e., no vertex is incident with two edges), then set  $M \leftarrow M \cup \{e_i\}$  (i.e., add edge  $e_i$  to the graph  $M$ )
- Return  $M$

#### Theorem 17.22: Greedy algorithm gives a 2-approximation [[Avis83]]

The greedy algorithm finds a 2-approximation of the maximum weighted matching problem. That is,  $w(M_{\text{greedy}}) \geq \frac{1}{2}w(M^*)$ .

### 17.5.1.2. Other algorithms to look at

- Improved Algorithm [DRAKE2003211]
- Blossom Algorithm Wikipedia

## 17.5.2. Minimum Spanning Tree

#### Definition 17.23: Spanning Tree

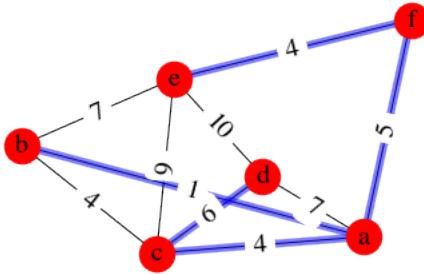
Given a graph  $G = (V, E)$ , a spanning tree connected, acyclic subgraph  $T$  that contains every node in  $V$ .

© spanning-tree<sup>5</sup>

Figure 17.8: spanning-tree

#### Definition 17.24: Max weight spanning tree

Given a graph  $G = (V, E)$ , with associated weights  $w_e \geq 0$  for all  $e \in E$ , a maximum weight spanning tree is a spanning tree maximizes the sum of the edge weights.



© spanning-tree-MST<sup>6</sup>  
**Figure 17.9: spanning-tree-MST**

### Lemma 17.25: Edges and Spanning Trees

Let  $G$  be a connected graph with  $n$  vertices.

- (a)  $T$  is a spanning tree of  $G$  if and only if  $T$  has  $n - 1$  edges and is connected.
- (b) Any subgraph  $S$  of  $G$  with more than  $n - 1$  edges contains a cycle.

See Section 19.2 for integer programming formulations of this problem.

### 17.5.3. Kruskal's algorithm

#### Resources

[Wikipedia](#)

#### Kruskal - for Minimum Spanning tree:

Complexity:  $O(|E| \log(|V|))$

- (a) Begin with an empty tree ( $T = \emptyset$ )
- (b) Label the edges in the graph such that  $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$
- (c) For  $i = 1, \dots, m$   
If  $T \cup \{e_i\}$  is acyclic, then set  $T \leftarrow T \cup \{e_i\}$
- (d) Return  $T$

<sup>5</sup>spanning-tree, from [spanning-tree](#). [spanning-tree](#), [spanning-tree](#).

<sup>6</sup>spanning-tree-MST, from [spanning-tree-MST](#). [spanning-tree-MST](#), [spanning-tree-MST](#).

### 17.5.3.1. Prim's Algorithm

---

#### Resources

- [Wikipedia](#)
- [TeXample - Figure for min spanning tree](#)

### 17.5.4. Traveling Salesman Problem

---

#### Resources

- 

See Section 19.3 for integer programming formulations of this problem. Also, hill climbing algorithms for this problem such as 2-Opt, simulated annealing, and tabu search will be discussed in Section ??.

#### 17.5.4.1. Nearest Neighbor - Construction Heuristic

---

#### Resources

- [Wikipedia](#)

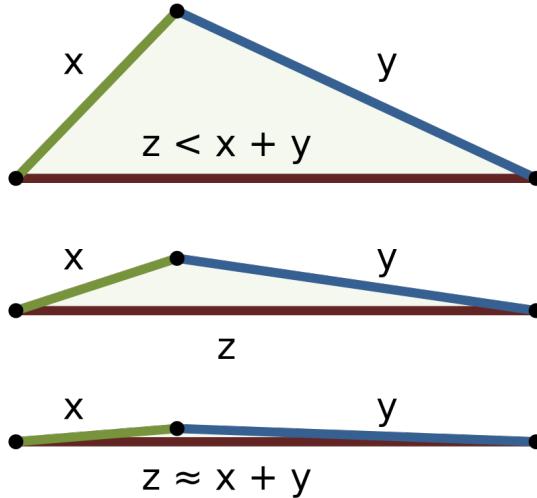
We will discuss heuristics more later in this book. For now, present this construction heuristic as a simple algorithmic example.

Starting from any node, add the edge to the next closest node. Continue this process.

#### Nearest Neighbor:

Complexity:  $O(n^2)$

- (a) Start from any node (lets call this node 1) and label this as your current node.
- (b) Pick the next current node as the one that is closest to the current node that has not yet been visited.
- (c) Repeat step 2 until all nodes are in the tour.



© wiki/File/triangle\_inequality.png<sup>7</sup>

**Figure 17.10: wiki/File/triangle\_inequality.png**

#### 17.5.4.2. Double Spanning Tree - 2-Apx

We can use a minimum spanning tree algorithm to find a provably okay solution to the TSP, provided certain properties of the graph are satisfied.

Graphs with nice properties are often easier to handle and typically graphs found in the real world have some nice properties. The *triangle inequality* comes from the idea of a triangle that the sum of the lengths of two sides always are longer than the length of the third side. See Figure 17.10

##### Definition 17.26: Triangle Inequality on a Graph

A complete, weighted graph  $G$  (i.e., a graph that has all possible edges and a weight assigned to each edge) satisfies the triangle inequality provided that for every triple of vertices  $a, b, c$  and edges  $e_{ab}, e_{bc}, e_{ac}$ , we have that

$$w(e_{ab}) + w(e_{bc}) \geq w(e_{ac}).$$

Let  $S$  be the resulting tour and let  $S^*$  be an optimal tour. Since the resulting tour is feasible, it will satisfy

$$w(S^*) \leq w(S).$$

But we also know that the weight of a minimum spanning tree  $T$  is less than that of the optimal tour, hence

$$w(T) \leq w(S^*).$$

Lastly, due to the triangle inequality we know that

$$w(S) \leq 2w(T),$$

<sup>7</sup>wiki/File/triangle\_inequality.png, from wiki/File/triangle\_inequality.png. wiki/File/triangle\_inequality.png, wiki/File/triangle\_inequality.png.

**Algorithm 5** Double Spanning Tree

---

**Require:**A graph  $G = (V, E)$  that satisfies the triangle inequality

**Ensure:**A tour that is a 2-Apx of the optimal solution

- 1: Compute a minimum spanning tree  $T$  of  $G$ .
  - 2: Double each edge in the minimum spanning tree (i.e., if edge  $e_{ab}$  is in  $T$ , add edge  $e_{ba}$ .
  - 3: Compute an Eulerian Tour using these edges.
  - 4: Return tour that visits vertices in the order the Eulerian Tour visits them, but without repeating any vertices.
- 

since replacing any edge in the Eulerian tour with a more direct edge only reduces the total weight.

Putting this together, we have

$$w(S) \leq 2w(T) \leq 2w(S^*)$$

and hence,  $S$  is a 2-approximation of the optimal solution.

#### 17.5.4.3. Christofides - Approximation Algorithm - (3/2)-Apx

---

If we combine algorithms for minimum spanning tree and matching, we can find a better approximation algorithm. This is given by Christofides. Again, this is in the case where the graph satisfies the triangle inequality. See Wikipedia - Christofides Algorithm or Ola Svensson Lecture Slides for more detail.

# 18. Introduction to computational complexity

---

## 18.1 Introduction

---

**Motivation:** We want to understand *what* is a problem and *when* a problem is *easy/hard*.

**Our strategy:** An intuitive review of the basic ideas in Computational Complexity theory.

**Key concepts:**

- Problem types: optimization problems, decision problems, feasibility problems.
- Instance (of a problem)
- A problem is a collection of instances.
- Size of an instance
- Algorithm
- Running time of an algorithm; worst-case time complexity
- Complexity classes:  $P$  and  $NP$

## 18.2 Problem, instance, size

---

### 18.2.1. Problem, instance

---

**Definition 18.1: Problem**

*Is a generic question/task that needs to be answered/solved.*

*A problem is a “collection of instances” (see below).*

A *particular* realization of a problem is define next.

**Definition 18.2: Instance**

*Is a specific case of a problem. In other words, we can say “ $\text{Instance} \in \text{Problem}$ ”.*

## 18.2.2. Format and examples of problems/instances

---

A problem is an abstract concept. We will write problems in the following format:

- **INPUT:** Generic data-instance.
- **OUTPUT:** Question to be answered and/or task to be performed with the data.

**Examples of problems/instances:**

- Typical problems: optimization problems, decision problems, feasibility problems.
- LP and IP feasibility, TSP, IP minimization, Maximum cardinality independent set.

## 18.2.3. Size of an instance

---

The size of an instance is the *amount of information* required to represent the instance (in the computer).

**Definition 18.3: Binary size/length**

*Is the number of bits that are needed in order to give the problem to a computer.*

**Examples of sizes:**

- Size of an integer/rational number.
- Size of a rational matrix.
- Size of a graph (node-edge matrix representation).

# 18.3 Algorithms, running time, Big-O notation

---

## 18.3.1. Basics

---

**Definition 18.4: Algorithm**

*List of instructions to solve a problem.*

**Definition 18.5: Running time of an algorithm**

*Is the number of steps (as a function of the size) that the algorithm takes in order to solve an instance.*

### 18.3.2. Worst-time complexity

---

Given an algorithm to solve a problem  $P$ , the *running time of*, as a function of the size  $\sigma \in \mathbb{Z}_+$  will be defined as follows:

- The (generic) running time will be a function  $f : \mathbb{Z}_+ \rightarrow \mathbb{Z}_+$ .
- Given  $\sigma$ , the function  $f$  is defined as follows:

$$f(\sigma) = \max\{\text{running time of } z \text{, where } \text{size}(z) \leq \sigma\}.$$

**Remark:** This is a very conservative/pessimistic measure of running time.

### 18.3.3. Big-O notation

---

A function  $f : \mathbb{Z}_+ \rightarrow \mathbb{Z}_+$  belongs to the class of functions  $O(g(n))$  (that is,  $f \in O(g(n))$ ) if there exists  $c > 0$ ,  $n_0 \in \mathbb{Z}_+$  such that

$$f(n) \leq cg(n), \quad \text{for all } n \geq n_0.$$

We usually say “ $f$  is  $O(g(n))$ ” or “ $f$  is order  $O(g(n))$ ”.

### 18.3.4. Examples

---

- Basic examples of Big-O notation:  $O(1)$ ,  $O(n^k)$ ,  $O(c^n)$ ,  $O(\log(n))$ , etc.
- An illustration of the fact that the running time depends on the size of the instance: the algorithm for the binary knapsack problem that is  $O(nb)$  is not polynomial, since the size of the instance is  $\log(b)$ .

## 18.4 Basics

---

### Definition 18.6: Polynomial time algorithm

An algorithm is said to be a polynomial time algorithm if its running time is  $O(n^k)$  for some  $k \geq 1$  (where  $n$  represents the size of a generic instance).

**Remark:** *Polynomial time algorithms* are also known as *Polytime algorithms*.

### Definition 18.7: Decision problem

A decision problem is any problem whose only acceptable answers are either YES or NO (but not both at the same time).

**Some examples:** feasibility problems, decision version of optimization problems, etc.

## 18.5 Complexity classes

---

We will introduce 3 complexity classes (For see at least 495 more classes, please see [http://complexityzoo.uwaterloo.ca/Complexity\\_Zoo](http://complexityzoo.uwaterloo.ca/Complexity_Zoo)).

### 18.5.1. Polynomial time problems

---

**Definition 18.8: The class  $\mathcal{P}$**

*Is the set of all decision problems for which a YES or NO answer for a particular instance can be obtained in polytime.*

**Remark:** For a particular problem  $P$ , there are 3 possibilities: (1)  $P \in \mathcal{P}$ , (2)  $P \notin \mathcal{P}$ , and  $P ? \mathcal{P}$  (i.e., we don't know).

**Examples:**

- Shortest path
- Max flow
- Min cut
- Matroid optimization
- Matchings
- Linear programming

### 18.5.2. Non-deterministic polynomial time problems

---

**Definition 18.9: The class NP**

*Is the set of all decision problems for which a YES answer for a particular instance can be verified in polytime.*

**Examples:**

- All problems in  $\mathcal{P}$
- Integer programming
- TSP
- Binary knapsack
- Maximum independent set
- Subset sum
- Partition
- SAT,  $k$ -SAT
- Clique

### 18.5.3. Complements of problems in NP

---

**Definition 18.10: The class NP**

*Is the set of all decision problems for which a NO answer for a particular instance can be verified in polytime.*

**Examples:**

- All problems in  $\mathcal{P}$
- PRIMES
- Every “complement” of an NP problem

**Remark:** Actually,  $\text{PRIMES} \in \text{NP}$  (see *Pratt's certificates*), even better, it was recently proven that  $\text{PRIMES} \in \mathcal{P}$  (by Manindra Agrawal, Neeraj Kayal, Nitin Saxena in 2004).

## 18.6 Relationship between the classes

---

### 18.6.1. A basic result

---

**Theorem 18.11**

*The following relationship holds:*

$$\mathcal{P} \subseteq \text{NP} \cap \text{coNP}.$$

### 18.6.2. An \$1,000,000 open question

---

The question “Is  $\mathcal{P} = \text{NP}$ ?” is one of the most important problems in mathematics and computer science. A correct answer is worth 1 Million dollars! Most people believe that  $\mathcal{P} \neq \text{NP}$ .

## 18.7 Comparing problems, Polynomial time reductions

---

**Motivation:** We would like to solve problem  $P_1$  by efficiently *reducing* it to another problem  $P_2$  (why? Perhaps we know how to solve  $P_2$ !!!).

**Definition 18.12: Polynomial time reductions**

Let  $P_1, P_2$  be decision problems. We say that  $P_1$  is polynomially reducible to  $P_2$  (denoted  $P_1 \leq_{\mathcal{P}} P_2$ ) if there exists a function

$$f : P_1 \rightarrow P_2$$

such that

- (a) For all  $w \in P_1$  the answer to  $w$  is YES if and only if the answer to  $f(w)$  is YES.
- (b) For all  $w \in P_1$   $f(w)$  can be computed in polynomial time w.r.t to  $\text{size}(w)$ . In particular, we must have that  $\text{size}(f(w))$  is polynomially bounded by  $\text{size}(w)$ .

**Remarks:**

- (a) In the above definition “ $f$  efficiently transforms an instance of  $P_1$  into an instance of  $P_2$ ”. In particular, if we know how to solve problem  $P_2$ , then we can solve problem  $P_1$ .
- (b) Therefore, the notation  $P_1 \leq_{\mathcal{P}} P_2$  makes sense: we are saying that  $P_1$  is “easier” to solve than  $P_2$ , as any algorithm for  $P_2$  would work for  $P_1$ .

## 18.8 Comparing problems, Polynomial time reductions

---

### 18.8.1. Definition

**Motivation:** We would like to solve problem  $P_1$  by efficiently *reducing* it to another problem  $P_2$  (why? Perhaps we know how to solve  $P_2$ !!!).

**Definition 18.13: Polynomial time reductions**

Let  $P_1, P_2$  be decision problems. We say that  $P_1$  is polynomially reducible to  $P_2$  (denoted  $P_1 \leq_{\mathcal{P}} P_2$ ) if there exists a function

$$f : P_1 \rightarrow P_2$$

such that

- (a) For all  $w \in P_1$  the answer to  $w$  is YES if and only if the answer to  $f(w)$  is YES.
- (b) For all  $w \in P_1$   $f(w)$  can be computed in polynomial time w.r.t to  $\text{size}(w)$ . In particular, we must have that  $\text{size}(f(w))$  is polynomially bounded by  $\text{size}(w)$ .

**Remarks:**

- (a) In the above definition “ $f$  efficiently transforms an instance of  $P_1$  into an instance of  $P_2$ ”. In particular, if we know how to solve problem  $P_2$ , then we can solve problem  $P_1$ .
- (b) Therefore, the notation  $P_1 \leq_{\mathcal{P}} P_2$  makes sense: we are saying that  $P_1$  is “easier” to solve than  $P_2$ , as any algorithm for  $P_2$  would work for  $P_1$ .

## 18.8.2. Basic properties

---

### Proposition 18.14

Let  $P_1, P_2$  be two problems such that  $P_1 \leq_{\mathcal{P}} P_2$  and assume that  $P_2 \in \mathcal{P}$ . Then  $P_1 \in \mathcal{P}$ .

### Proposition 18.15

Let  $P_1, P_2$  be two problems such that  $P_1 \leq_{\mathcal{P}} P_2$  and assume that  $P_2 \in NP$ . Then  $P_1 \in NP$ .

### Proposition 18.16

Let  $P_1, P_2, P_3$  be three problems assume that  $P_1 \leq_{\mathcal{P}} P_2$  and  $P_2 \leq_{\mathcal{P}} P_3$ . Then  $P_1 \leq_{\mathcal{P}} P_3$ .

# 18.9 NP-Completeness

---

## 18.9.1. The basics

---

### Definition 18.17: NP-Completeness

A decision problem  $P$  is said to be NP-complete if:

- (a)  $P \in NP$
- (b)  $Q \leq_{\mathcal{P}} P$  for all  $Q \in NP$  (that is, every problem  $Q$  in  $NP$  can be polynomially reduced to  $P$ ).

### Proposition 18.18

If  $P$  is NP-complete and  $P \in \mathcal{P}$  then  $\mathcal{P} = NP$ .

### 18.9.2. Do NP-complete problems exist?

**Theorem 18.19: S. Cook, 1971**

*SAT* is NP-complete.

## 18.10 NP-Hardness

**Definition 18.20: NP-Completeness**

A problem  $P$  is said to be NP-hard if there exists a NP-complete decision problem that can be reduced to it.

**Remarks:**

- NP-complete problems are NP-hard.
- Problems in NP-hard not need to be decision problems.
- Optimization versions of NP-complete decision problems are NP-hard.
- If  $P$  is NP-hard and  $P \in \mathcal{P}$  then  $\mathcal{P} = \text{NP}$ .

## 18.11 Exercises

- (a) Let  $P, Q$  be decision problems such that every instance of  $Q$  is an instance of  $P$  (that is  $\{\text{instances in } Q\} \subseteq \{\text{instances in } P\}$ ).
- Give an example of  $P, Q$  such that  $Q \in \mathcal{P}$  and  $P \in \text{NP} - \text{complete}$ .
  - Give an example of  $P, Q$  such that  $Q, P \in \text{NP} - \text{complete}$ .

**Note:** You must prove that your problem belongs to the corresponding class unless we have proved or sketched the proof of that fact in class.

**Solution:**

- Let  $P$  be the *Knapsack problem*. Then  $P$  is NP-complete (see Problem 5).
  - Let  $Q$  be the special case of the *Knapsack problem* where all weights are 1, that is,  $a_1, \dots, a_n = 1$ . The following algorithm decides this special case:  
**ALGORITHM:**
    - List the objects  $1, \dots, n$  in decreasing order according to  $c_1, \dots, c_n$ .

2. Select the first  $b$  objects from that list. Call this set  $S$ .
3. If  $\sum_{i \in S} c_i \leq k$ , then the output of the algorithm is YES. Else, the output is NO.

Clearly, this algorithm is correct and runs in polynomial time w.r.t. to the instance. Hence,  $Q \in \mathcal{P}$ .

- ii.
  - Let  $P$  be SAT.
  - Let  $Q$  be 3-SAT.

We showed in class that both  $P$  and  $Q$  are NP-complete problems.

(b) A *Hamiltonian cycle* in a graph  $G = (V, E)$  is a simple cycle that contains all the vertices. A *Hamiltonian  $s - t$  path* in a graph is a simple path from  $s$  to  $t$  that contains all of the vertices. The associated decision problems are:

- **Hamiltonian Cycle Input:**  $G = (V, E)$ . **Question:** Does there exist a hamiltonian cycle in  $G$ ?
  - **Hamiltonian Path Input:**  $G = (V, E)$ ,  $s, t \in V, s \neq t$ . **Question:** Does there exist a hamiltonian  $s-t$  path in  $G$ ?
- i. Given that *Hamiltonian Cycle* is NP-complete, prove that *Hamiltonian Path* is NP-complete.
  - ii. Given that *Hamiltonian Cycle* is NP-complete, prove that the optimization version of the TSP problem is NP-hard.

## Solution:

### i. Step 1: Hamiltonian Path is in NP.

It is clear that *Hamiltonian Path* is in NP, the certificate to a YES answer is the path itself. Given a Hamiltonian path from  $s$  to  $t$ . We only need to travel along it to check that in fact it visits every vertex once and that starts in  $s$  and ends in  $t$ . This takes  $O(|E|)$  time.

### Step 2: $\text{Hamiltonian Cycle} \leq_P \text{Hamiltonian Path}$ .

Given an instance  $G = (V, E)$  of *Hamiltonian Cycle*, we construct an instance of *Hamiltonian Path* as follows:

- Let  $v \in V$ , the we construct the graph  $G' = (V', E')$ , where:
  - $V' = V \setminus \{v\} \cup \{v_1, v_2\}$  (this takes  $O(1)$ ).
  - $E' = (E \setminus \{\{v, u\} : \{v, u\} \in E\}) \cup \{\{v_1, u\} : \{v, u\} \in E\} \cup \{\{v_2, u\} : \{v, u\} \in E\}$  (this takes  $O(|V|)$ ).
- The instance is:  $G' = (V', E')$ ,  $s = v_1$ ,  $t = v_2$ .

Notice the construction takes polynomial time w.r.t. the size of the instance.

Finally, the fact that a YES to an instance of *Hamiltonian Cycle* is equivalent to a YES to the associated *Hamiltonian Path* instance follows by noticing that there exists a Hamiltonian cycle of the form

$$vu_1u_2 \dots u_{n-1}v$$

in  $G$  if and only if there exists and Hamiltonian  $v_1-v_2$  path in  $G'$  of the form

$$v_1u_1u_2 \dots u_{n-1}v_2$$

in  $G'$ .

**Note:** we are denoting  $n = |V|$  and the notation  $u_0u_1 \dots u_k$  represents the path/cycle that uses the edges  $\{u_0, u_1\} \{u_1, u_2\} \dots \{u_{k-1}, u_k\}$  (in that order).

**Conclusion:** *Hamiltonian Path* is NP-complete.

- ii. Given an instance  $G = (V, E)$  of *Hamiltonian Cycle*, the following algorithm decides whether the answer to this instance is yes or no:

**ALGORITHM:**

1. Given  $V = \{v_1, \dots, v_n\}$ , consider the cities  $\{1, \dots, n\}$ .
2. Construct the objective function  $c$  given by:

$$c_{ij} = \begin{cases} 0, & \{v_i, v_j\} \in E \\ 1, & \text{else.} \end{cases}$$

3. Solve the TSP instance given above. Let  $\alpha$  be the cost of the optimal tour.
4. If  $\alpha = 0$ , then the output of the algorithm is YES. Else, the output is NO.

The algorithm is correct since the definition of the objective function implies that the optimal tour has cost equals to zero if and only if the tour only travels between pairs of cities associated to edges in  $E$ .

Notice that the algorithm only need to solve the TSP problem once and that every other step takes polynomial time. Therefore, this is a valid polynomial time reduction since if we were able to solve the optimization version of the TSP in polynomial time, we would also be able to decide *Hamiltonian Cycle* in polynomial time.

**Conclusion:** the optimization version of the TSP problem is NP-hard.

- (c) Given that the *node packing problem* is NP – complete, show that the following problems are also NP – complete:
- i. *Node cover*: **Input:**  $G = (V, E)$ ,  $k \in \mathbb{Z}_+$ . **Question:** Does there exist a set  $S \subseteq V$  of size at most  $k$  such that every edge of  $G$  is incident to a node of  $S$ ?
  - ii. *Uncapacitated facility location*. **Input:** sets  $M, N$  and integers  $k, c_{ij}, f_j$  for  $i \in M, j \in N$ . **Question:** Is there a set  $S \subseteq N$  such that  $\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k$ ?

Recall that *Node packing problem* is: **Input:**  $G = (V, E)$ ,  $k \in \mathbb{Z}_+$ . **Question:** Does there exist an independent set of size at least  $k$  in  $G$ ?

**Solution:**i. **Step 1: Node cover is in NP.**

It is clear that *Node cover* is in NP, the certificate is the node cover itself. Verifying that the set of nodes is a cover can be done by checking that every edge is connected to a node in the given set. This takes  $O(|E||V|)$  time.

**Step 2: Node packing  $\leq_P$  Node cover.**

Given an instance  $G = (V, E)$ ,  $l \in \mathbb{Z}_+$  of *Node Packing*, we construct an instance of *Node cover* as follows:

- We construct the graph  $G' = (V', E')$ , where:
  - $V' = V$  (this takes  $O(1)$ ).
  - $E' = E$  (this takes  $O(1)$ ).
- We take  $k = |V| - l$  (this takes  $O(1)$ ).
- The instance is:  $G' = (V', E')$ ,  $k \in \mathbb{Z}_+$ .

Notice the construction takes polynomial time w.r.t. the size of the instance.

Finally, the fact that a YES to an instance of *Node packing* is equivalent to a YES to the associated *Node cover* instance follows by noticing that a set  $U \subseteq V$  is a node packing in  $G$  if and only if no edge in  $E$  has both end points in  $U$ , which is equivalent to say that every edge in  $E$  has at least one end point in  $V \setminus U$ . Equivalently, this is saying that the set  $V \setminus U$  is a node cover in  $G'$ .

**Conclusion:** *Node cover* is NP-complete.

ii. **Step 1: Uncapacitated facility location is in NP.**

It is clear that *Uncapacitated facility location* is in NP, because given  $S \subseteq N$ , we can verify in  $O(|M||N| + |N|)$  if

$$\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k$$

**Step 2: Node packing  $\leq_P$  Uncapacitated facility location.**

Given an instance  $G = (V, E)$ ,  $l \in \mathbb{Z}_+$  of *Node Packing*, we construct an instance of *Uncapacitated facility location* as follows:

- $M = E$ ,  $N = V$  (this takes  $O(|E|)$ ).
- The objective

$$c_{ij} = \begin{cases} |V| + 1, & \text{if } j = uv, \text{ where } u, v \neq i \\ 0, & \text{otherwise.} \end{cases}$$

(This takes  $O(|E|^2)$ .)

- $k = |V| - l$  (this takes  $O(1)$ ).
- $f_j = 1$ , for all  $j \in N$  (this takes  $O(|V|)$ ).

Notice the construction takes polynomial time w.r.t. the size of the instance.

- **YES to Node Packing  $\Rightarrow$  YES to Uncapacitated facility location:**

If  $U$  is a node packing, with  $|U| \geq l$ , then  $S = V \setminus U$  is a vertex cover, hence for all  $i \in M$ :

$$\min\{c_{ij} : j \in S\} = 0.$$

And

$$\sum_{j \in S} f_j = |S| = |V| - |U| \leq |V| - l \leq k.$$

This implies

$$\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k.$$

Thus, the set  $S \subseteq N$  gives a YES answer to *Uncapacitated facility location*.

- **YES to Uncapacitated facility location  $\Rightarrow$  YES to Node Packing:**

There exists  $S \subseteq N$  such that

$$\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k.$$

By definition of the reduction from node packing, we have

- (i) For all  $i \in M$ ,  $\min\{c_{ij} : j \in S\} \leq |V|$ , which implies  $\min\{c_{ij} : j \in S\} = 0$ .
- (ii) Let  $U = V \setminus S$ . By (i),  $\sum_{j \in S} f_j = |S| = |V| - |U|$ .
- (iii) By (i), if  $u, v \in U$ , then we must have  $\{u, v\} \notin E$ .
- (iv) By (ii), we have  $|U| \geq l$ .

This implies that the set  $U \subseteq V$  is a node packing with  $|U| \geq l$ , which gives a YES answer to *Node Packing*.

**Conclusion:** *Uncapacitated facility location* is NP-complete.



# 19. Exponential Size Formulations

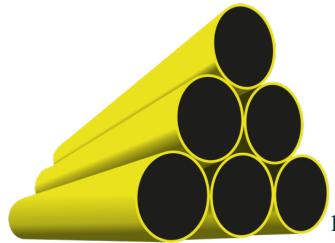
Although typically models need to be a reasonable size in order for us to code them and send them to a solver, there are some ways that we can allow having models of exponential size. The first example here is the cutting stock problem, where we will model with exponentially many variables. The second example is the traveling salesman problem, where we will model with exponentially many constraints. We will also look at some other models for the traveling salesman problem.

## 19.1 Cutting Stock

This is a classic problem that works excellent for a technique called *column generation*. We will discuss two versions of the model and then show how we can use column generation to solve the second version more efficiently. First, let's describe the problem.

### Cutting Stock:

You run a company that sells pipes of different lengths. These lengths are  $L_1, \dots, L_k$ . To produce these pipes, you have one machine that produces pipes of length  $L$ , and then cuts them into a collection of shorter pipes as needed.



You have an order come in for  $d_i$  pipes of length  $i$  for  $i = 1, \dots, k$ . How can you fill the order while cutting up the fewest number of pipes?

### Example 19.1: Cutting stock with pipes

A plumber stocks standard lengths of pipe, all of length 19 m. An order arrives for:

- 12 lengths of 4m
- 15 lengths of 5m
- 22 lengths of 6m

How should these lengths be cut from standard stock pipes so as to minimize the number of standard pipes used?

An initial model for this problem could be constructed as follows:

- Let  $N$  be an upper bound on the number of pipes that we may need.
- Let  $z_j = 1$  if we use pipe  $i$  and  $z_j = 0$  if we do not use pipe  $j$ , for  $j = 1, \dots, N$ .
- Let  $x_{ij}$  be the number of cuts of length  $L_i$  in pipe  $j$  that we use.

Then we have the following model

$$\begin{aligned}
 & \min \sum_{j=1}^N z_j \\
 \text{s.t. } & \sum_{i=1}^k L_i x_{ij} \leq L z_j \text{ for } j = 1, \dots, N \\
 & \sum_{j=1}^N x_{ij} \geq d_i \text{ for } i = 1, \dots, k \\
 & z_j \in \{0, 1\} \text{ for } j = 1, \dots, N \\
 & x_{ij} \in \mathbb{Z}_+ \text{ for } i = 1, \dots, k, j = 1, \dots, N
 \end{aligned} \tag{19.1}$$

### Exercise 19.2: Show Bound

In the example above, show that we can choose  $N = 16$ .

For our example above, using  $N = 16$ , we have

$$\begin{aligned}
 & \min \sum_{j=1}^{16} z_j \\
 \text{s.t. } & 4x_{1j} + 5x_{2j} + 6x_{3j} \leq 19z_j \\
 & \sum_{j=1}^{16} x_{1j} \geq 12 \\
 & \sum_{j=1}^{16} x_{2j} \geq 15 \\
 & \sum_{j=1}^{16} x_{3j} \geq 22 \\
 & z_j \in \{0, 1\} \text{ for } j = 1, \dots, 16 \\
 & x_{ij} \in \mathbb{Z}_+ \text{ for } i = 1, \dots, 3, j = 1, \dots, 16
 \end{aligned} \tag{19.2}$$

Additionally, we could break the symmetry in the problem. That is, suppose the solution uses 10 of the 16 pipes. The current formulation does not restrict which 10 pipes are used. Thus, there are many possible solutions. To reduce this complexity, we can state that we only use the first 10 pipes. We can write a constraint that says *if we don't use pipe  $j$ , then we also will not use any subsequent*

*pipes*. Hence, by not using pipe 11, we enforce that pipes 11, 12, 13, 14, 15, 16 are not used. This can be done by adding the constraints

$$z_1 \geq z_2 \geq z_3 \geq \cdots \geq z_N. \quad (19.3)$$

See ?? for code for this formulation.

Unfortunately, this formulation is slow and does not scale well with demand. In particular, the number of variables is  $N + kN$  and the number of constraints is  $N$  (plus integrality and non-negativity constraints on the variables). The solution times for this model are summarized in the following table:

#### INPUT TABLE OF COMPUTATIONS

##### 19.1.1. Pattern formulation

---

We could instead list all patterns that are possible to cut each pipe. A pattern is an vector  $a \in \mathbb{Z}_+^k$  such that for each  $i$ ,  $a_i$  lengths of  $L_i$  can be cut from a pipe of length  $L$ . That is

$$\sum_{i=1}^k L_i a_i \leq L \quad (19.4)$$

$a_i \in \mathbb{Z}_+$  for all  $i = 1, \dots, k$

In our running example, we have

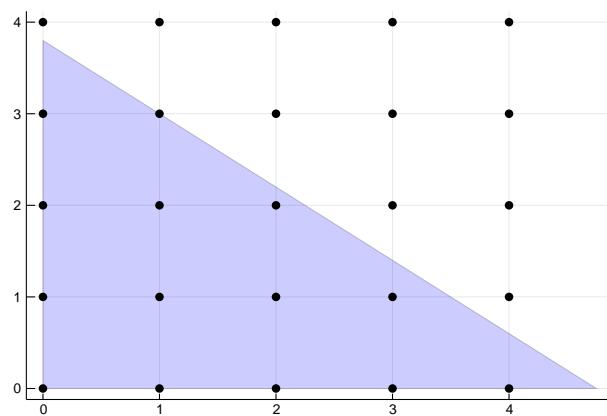
$$4a_1 + 5a_2 + 6a_3 \leq 19 \quad (19.5)$$

$a_i \in \mathbb{Z}_+$  for all  $i = 1, \dots, 3$

For visualization purposes, consider the patterns where  $a_3 = 0$ . That is, only patterns with cuts of length 4m or 5m. All patterns of this type are represented by an integer point in the polytope

$$P = \{(a_1, a_2) : 4a_1 + 5a_2 \leq 19, a_1 \geq 0, a_2 \geq 0\} \quad (19.6)$$

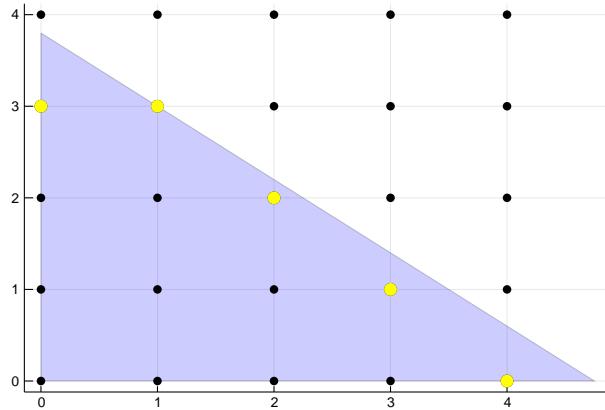
which we can see here:



© knapsack\_fig.pdf<sup>2</sup>

<sup>2</sup>knapsack\_fig.pdf, from knapsack\_fig.pdf. knapsack\_fig.pdf, knapsack\_fig.pdf.

where  $P$  is the blue triangle and each integer point represents a pattern. Feasible patterns lie inside the polytope  $P$ . Note that we only need patterns that are maximal with respect to number of each type we cut. Pictorially, we only need the patterns that are integer points represented as yellow dots in the picture below.



© knapsack\_fig\_maximal.pdf<sup>3</sup>

For example, the pattern  $[3,0,0]$  is not needed (only cut 3 of length 4m) since we could also use the pattern  $[4,0,0]$  (cut 4 of length 4m) or we could even use the pattern  $[3,1,0]$  (cut 3 of length 4m and 1 of length 5m).

### Example 19.3: Pattern Formulation

Let's list all the possible patterns for the cutting stock problem:

	Patterns									
Cuts of length 4m	0	0	1	0	2	1	2	3	4	1
Cuts of length 5m	0	1	0	2	1	2	2	1	0	3
Cuts of length 6m	3	2	2	1	1	0	0	0	0	0

We can organize these patterns into a matrix.

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 2 & 1 & 2 & 3 & 4 & 1 \\ 0 & 1 & 0 & 2 & 1 & 2 & 2 & 1 & 0 & 3 \\ 3 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (19.7)$$

Let  $p$  be the number of patterns that we have. We create variables  $x_1, \dots, x_p \in \mathbb{Z}_+$  that denote the number of times we use each pattern.

Now, we can recast the optimization problem as

$$\min \sum_{i=1}^p x_i \quad (19.8)$$

$$\text{such that } Ax \geq \begin{bmatrix} 12 \\ 15 \\ 22 \end{bmatrix} \quad (19.9)$$

$$x \in \mathbb{Z}_+^p \quad (19.10)$$

### 19.1.2. Column Generation

---

Consider the linear program(8.9), but in this case we are instead minimizing.

Thus we can write it as

$$\begin{aligned} \min \quad & (c_N - c_B A_B^{-1} A_N) x_N + c_B A_B^{-1} b \\ \text{s.t.} \quad & x_B + A_B^{-1} A_N x_N = A_B^{-1} b \\ & x \geq 0 \end{aligned} \quad (19.11)$$

In our LP we have  $c = 1$ , that is,  $c_i = 1$  for all  $i = 1, \dots, k$ . Hence, we can write it as

$$\begin{aligned} \min \quad & (1_N - 1_B A_B^{-1} N) x_N + 1_B A_B^{-1} b \\ \text{s.t.} \quad & x_B + A_B^{-1} A_N x_N = A_B^{-1} b \\ & x \geq 0 \end{aligned} \quad (19.12)$$

Now, if there exists a non-basic variable that could enter the basis and improve the objective, then there is one with a reduced cost that is negative. For a particular non-basic variable, the coefficient on it is

$$(1 - 1_B A_B^{-1} A_N^i) x_i \quad (19.13)$$

where  $A_N^i$  is the  $i$ -th column of the matrix  $A_N$ . Thus, we want to look for a column  $a$  of  $A_N$  such that

$$1 - 1_B A_B^{-1} a < 0 \Rightarrow 1 < 1_B A_B^{-1} a \quad (19.14)$$

#### Pricing Problem:

(knapsack problem!)

Given a current basis  $B$  of the *master* linear program, there exists a new column to add to the basis that improves the LP objective if and only if the following problem has an objective value strictly larger than 1.

$$\begin{aligned} \max \quad & 1_B A_B^{-1} a \\ \text{s.t.} \quad & \sum_{i=1}^k L_i a_i \leq L \\ & a_i \in \mathbb{Z}_+ \text{ for } i = 1, \dots, k \end{aligned} \quad (19.15)$$

### Example 19.4: Pricing Problem

Let's make the initial choice of columns easy. We will do this by selecting columns

	Patterns		
Cuts of length 4m	4	0	0
Cuts of length 5m	0	3	0
Cuts of length 6m	0	0	3

So our initial A matrix is

$$A = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{pmatrix} \quad (19.16)$$

Notice that there are enough patterns in the initial A matrix to produce feasible solution. Let's also append an arbitrary column to the A matrix as a potential new pattern.

$$A = \begin{pmatrix} 4 & 0 & 0 & a_1 \\ 0 & 3 & 0 & a_2 \\ 0 & 0 & 3 & a_3 \end{pmatrix} \quad (19.17)$$

Now, let's solve the linear relaxation and compute the tabluea.

$$\begin{aligned} \max \quad & \dots \dots a \\ \text{s.t.} \quad & 4a_1 + 5a_2 + 6a_3 \leq 19 \\ & a_i \in \mathbb{Z}_+ \text{ for } i = 1, \dots, k \end{aligned} \quad (19.18)$$

### 19.1.3. Cutting Stock - Multiple widths

#### Resources

Gurobi has an excellent demonstration application to look at: [Gurobi - Cutting Stock Demo](#)  
[Gurobi - Multiple Master Rolls](#)

Here are some solutions:

- <https://github.com/fzsun/cutstock-gurobi>.
- <http://www.dcc.fc.up.pt/~jpp/mpa/cutstock.py>

Here is an AIMMS description of the problem: [AIMMS Cutting Stock](#)

## 19.2 Spanning Trees

### Resources

See [Abdelmaguid2018] for a list of 11 models for the minimum spanning tree and a comparison using CPLEX.

## 19.3 Traveling Salesman Problem

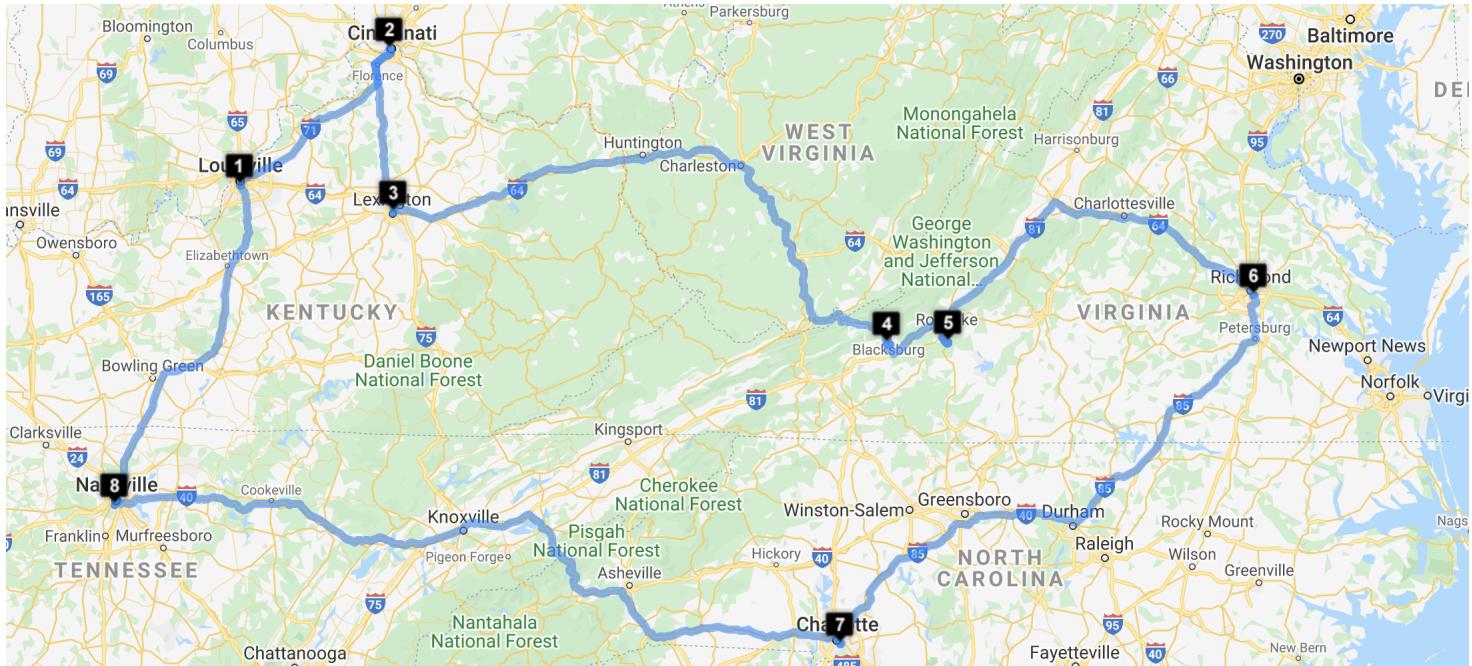
### Resources

See [math.watloo.ca](http://math.watloo.ca) for excellent material on the TSP.

See also this chapter *A Practical Guide to Discrete Optimization*.

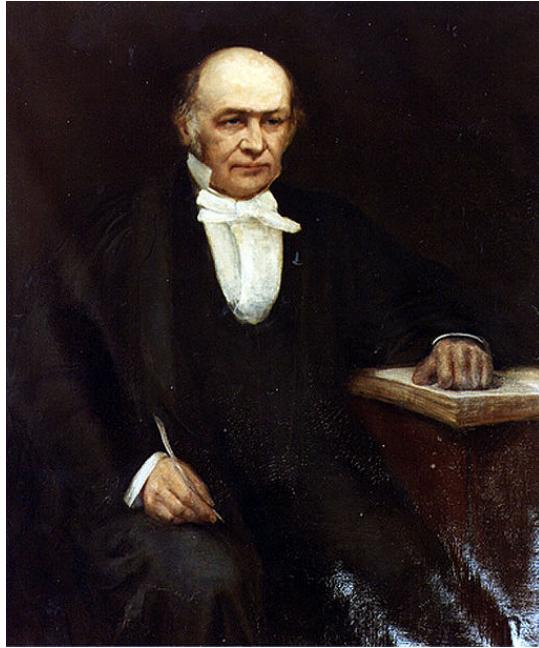
Also, watch this excellent talk by Bill Cook "Postcards from the Edge of Possibility":  
[Youtube!](https://www.youtube.com/watch?v=JyfXWzqjwIY)

Google Maps!



**Figure 19.1:** Optimal tour through 8 cities. Generated by Gebweb - Optimap. See it also on Google Maps!.

We consider a directed graph, graph  $G = (N, A)$  of nodes  $N$  and arcs  $A$ . Arcs are directed edges. Hence the arc  $(i, j)$  is the directed path  $i \rightarrow j$ .



© wiki/File/William\_Rowan\_Hamilton\_painting.jpg<sup>4</sup>

**Figure 19.2:** wiki/File/William\_Rowan\_Hamilton\_painting.jpg

A *tour*, or Hamiltonian cycle (see Figure 19.2), is a cycle that visits all the nodes in  $N$  exactly once and returns back to the starting node.

Given costs  $c_{ij}$  for each arc  $(i, j) \in A$ , the goal is to find a minimum cost tour.

**Traveling Salesman Problem:**

*NP-Hard*

Given a directed graph  $G = (N, A)$  and costs  $c_{ij}$  for all  $(i, j) \in A$ , find a tour of minimum cost.

**ADD TSP FIGURE**

In the figure, the nodes  $N$  are the cities and the arcs  $A$  are the directed paths city  $i \rightarrow$  city  $j$ .

**MODELS** When constructing an integer programming model for TSP, we define variables  $x_{ij}$  for all  $(i, j) \in A$  as

$$x_{ij} = 1 \text{ if the arc } (i, j) \text{ is used and } x_{ij} = 0 \text{ otherwise.}$$

We want the model to satisfy the fact that each node should have exactly one incoming arc and one leaving arc. Furthermore, we want to prevent self loops. Thus, we need the constraints:

---

<sup>4</sup>wiki/File/William\_Rowan\_Hamilton\_painting.jpg, from wiki/File/William\_Rowan\_Hamilton\_painting.jpg, wiki/File/William\_Rowan\_Hamilton\_painting.jpg, wiki/File/William\_Rowan\_Hamilton\_painting.jpg.

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \text{ [outgoing arc]} \quad (19.1)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \text{ [incoming arc]} \quad (19.2)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \text{ [no self loops]} \quad (19.3)$$

Unfortunately, these constraints are not enough to completely describe the problem. The issue is that *subtours* may arise. For instance

#### ADD SUBTOURS FIGURE

### 19.3.1. Miller Tucker Zemlin (MTZ) Model

---

The Miller-Tucker-Zemlin (MTZ) model for the TSP uses variables to mark the order for which cities are visited. This model introduce general integer variables to do so, but in the process, creates a formulation that has few inequalities to describe.

Some feature of this model:

- This model adds variables  $u_i \in \mathbb{Z}$  with  $1 \leq u_i \leq n$  that decide the order in which nodes are visited.
- We set  $u_1 = 1$  to set a starting place.
- Crucially, this model relies on the following fact

*Let  $x$  be a solution to (19.1)-(19.3) with  $x_{ij} \in \{0, 1\}$ . If there exists a subtour in this solution that contains the node 1, then there also exists a subtour that does not contain the node 1.*

The following model adds constraints

$$\text{If } x_{ij} = 1, \text{ then } u_i + 1 \leq u_j. \quad (19.4)$$

This if-then statement can be modeled with a big-M, choosing  $M = n$  is a sufficient upper bound. Thus, it can be written as

$$u_i + 1 \leq u_j + n(1 - x_{ij}) \quad (19.5)$$

Setting these constraints to be active enforces the order  $u_i < u_j$ .

Consider a subtour now  $2 \rightarrow 5 \rightarrow 3 \rightarrow 2$ . Thus,  $x_{25} = x_{53} = x_{32} = 1$ . Then using the constraints from (19.5), we have that

$$u_2 < u_5 < u_3 < u_2, \quad (19.6)$$

but this is infeasible since we cannot have  $u_2 < u_2$ .

As stated above, if there is a subtour containing the node 1, then there is also a subtour not containing the node 1. Thus, we can enforce these constraints to only prevent subtours that don't contain the node 1. Thus, the full tour that contains the node 1 will still be feasible.

This is summarized in the following model:

**Traveling Salesman Problem - MTZ Model:**

$$\min \sum_{i,j \in N} c_{ij} x_{ij} \quad (19.7)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \quad [\text{outgoing arc}] \quad (19.8)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \quad [\text{incoming arc}] \quad (19.9)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \quad [\text{no self loops}] \quad (19.10)$$

$$u_i + 1 \leq u_j + n(1 - x_{ij}) \quad \text{for all } i, j \in N, i, j \neq 1 \quad [\text{prevents subtours}] \quad (19.11)$$

$$u_1 = 1 \quad (19.12)$$

$$2 \leq u_i \leq n \quad \text{for all } i \in N, i \neq 1 \quad (19.13)$$

$$u_i \in \mathbb{Z} \quad \text{for all } i \in N \quad (19.14)$$

$$x_{ij} \in \{0, 1\} \quad \text{for all } i, j \in N \quad (19.15)$$

**Example 19.5: TSP with 4 nodes**

*Distance Matrix:*

A \ B	1	2	3	4
-----				
1	0	1	2	3
2	1	0	1	2
3	2	1	0	4
4	3	2	4	0

**Example 19.6: MTZ model for TSP with 4 nodes**

Here is the full MTZ model:

$$\begin{aligned} \min \quad & x_{1,2} + 2x_{1,3} + 3x_{1,4} + x_{2,1} + x_{2,3} + 2x_{2,4} + \\ & 2x_{3,1} + x_{3,2} + 4x_{3,4} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3} \end{aligned}$$

Subject to

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1 \quad \text{outgoing from node 1}$$

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1 \quad \text{outgoing from node 2}$$

$$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} = 1 \quad \text{outgoing from node 3}$$

$$x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} = 1 \quad \text{outgoing from node 4}$$

$$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} = 1 \quad \text{incoming to node 1}$$

$$x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1 \quad \text{incoming to node 2}$$

$$x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} = 1 \quad \text{incoming to node 3}$$

$$x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} = 1 \quad \text{incoming to node 4}$$

$$x_{1,1} = 0 \quad \text{No self loop with node 1}$$

$$x_{2,2} = 0 \quad \text{No self loop with node 2}$$

$$x_{3,3} = 0 \quad \text{No self loop with node 3}$$

$$x_{4,4} = 0 \quad \text{No self loop with node 4}$$

$$u_1 = 1 \quad \text{Start at node 1}$$

$$2 \leq u_i \leq 4, \quad \forall i \in \{2, 3, 4\}$$

$$u_2 + 1 \leq u_3 + 4(1 - x_{2,3})$$

$$u_2 + 1 \leq u_4 + 4(1 - x_{2,4}) \leq 3$$

$$u_3 + 1 \leq u_2 + 4(1 - x_{3,2}) \leq 3$$

$$u_3 + 1 \leq u_4 + 4(1 - x_{3,4}) \leq 3$$

$$u_4 + 1 \leq u_2 + 4(1 - x_{4,2}) \leq 3$$

$$u_4 + 1 \leq u_3 + 4(1 - x_{4,3}) \leq 3$$

$$x_{i,j} \in \{0, 1\} \quad \forall i \in \{1, 2, 3, 4\}, j \in \{1, 2, 3, 4\}$$

$$u_i \in \mathbb{Z}, \quad \forall i \in \{1, 2, 3, 4\}$$

**Example 19.7: MTZ model for TSP with 5 nodes**

$$\min \quad x_{1,2} + 2x_{1,3} + 3x_{1,4} + 4x_{1,5} + x_{2,1} + x_{2,3} + 2x_{2,4} + 2x_{2,5} + 2x_{3,1} + \\ x_{3,2} + 4x_{3,4} + x_{3,5} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3} + 2x_{4,5} + \\ 4x_{5,1} + 2x_{5,2} + x_{5,3} + 2x_{5,4}$$

*Subject to*

$$\begin{aligned} x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} &= 1 \\ x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} &= 1 \\ x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} + x_{3,5} &= 1 \\ x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} + x_{4,5} &= 1 \\ x_{5,1} + x_{5,2} + x_{5,3} + x_{5,4} + x_{5,5} &= 1 \end{aligned}$$

$$\begin{aligned} x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} + x_{5,1} &= 1 \\ x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} + x_{5,2} &= 1 \\ x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} + x_{5,3} &= 1 \\ x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} + x_{5,4} &= 1 \\ x_{1,5} + x_{2,5} + x_{3,5} + x_{4,5} + x_{5,5} &= 1 \end{aligned}$$

$$\begin{aligned} x_{1,1} &= 0 \\ x_{2,2} &= 0 \\ x_{3,3} &= 0 \\ x_{4,4} &= 0 \\ x_{5,5} &= 0 \end{aligned}$$

$$\begin{aligned} u_1 &= 1 \\ 2 \leq u_i \leq 5 &\quad \forall i \in \{1, 2, 3, 4, 5\} \\ u_2 + 1 &\leq u_3 + 5(1 - x_{2,3}) \\ u_2 + 1 &\leq u_4 + 5(1 - x_{2,4}) \\ u_2 + 1 &\leq u_5 + 5(1 - x_{2,5}) \\ u_3 + 1 &\leq u_2 + 5(1 - x_{3,2}) \\ u_3 + 1 &\leq u_4 + 5(1 - x_{3,4}) \\ u_4 + 1 &\leq u_2 + 5(1 - x_{4,2}) \\ u_4 + 1 &\leq u_3 + 5(1 - x_{4,3}) \\ u_3 + 1 &\leq u_5 + 5(1 - x_{3,5}) \\ u_4 + 1 &\leq u_5 + 5(1 - x_{4,5}) \\ u_5 + 1 &\leq u_2 + 5(1 - x_{5,2}) \\ u_5 + 1 &\leq u_3 + 5(1 - x_{5,3}) \\ u_5 + 1 &\leq u_4 + 5(1 - x_{5,4}) \end{aligned}$$

$$x_{i,j} \in \{0, 1\} \quad \forall i \in \{1, 2, 3, 4, 5\}, j \in \{1, 2, 3, 4, 5\}$$

$$u_i \in \mathbb{Z}, \quad \forall i \in \{1, 2, 3, 4, 5\}$$

## PROS OF THIS MODEL

- Small description
- Easy to implement

## CONS OF THIS MODEL

- Linear relaxation is not very tight. Thus, the solver may be slow when given this model.

**Example 19.8: Subtour elimination constraints via MTZ model**

Consider the subtour  $2 \rightarrow 4 \rightarrow 5 \rightarrow 2$ .

For this subtour to exist in a solution, we must have

$$x_{2,4} = 1$$

$$x_{4,5} = 1$$

$$x_{5,2} = 1.$$

Consider the three corresponding inequalities to these variables:

$$u_2 + 1 \leq u_4 + 5(1 - x_{2,4})$$

$$u_4 + 1 \leq u_5 + 5(1 - x_{4,5})$$

$$u_5 + 1 \leq u_2 + 5(1 - x_{5,2}).$$

Since  $x_{2,4} = x_{4,5} = x_{5,2} = 1$ , these reduce to

$$u_2 + 1 \leq u_5$$

$$u_4 + 1 \leq u_5$$

$$u_5 + 1 \leq u_2.$$

Now, lets add these inequalities together. This produces the inequality

$$u_2 + u_4 + u_5 + 3 \leq u_2 + u_4 + u_5,$$

which reduces to

$$3 \leq 0.$$

This inequality is invalid, and hence no solution can have the values  $x_{2,4} = x_{4,5} = x_{5,2} = 1$ .

**Example 19.9: Weak Model**

Consider again the same tour in the last example, that is, the subtour  $2 \rightarrow 4 \rightarrow 5 \rightarrow 2$ . We are interested to know how strong the inequalties of the problem description are if we allow the variables to be continuous variables. That is, suppose we relax  $x_{ij} \in \{0, 1\}$  to be  $x_{ij} \in [0, 1]$ .

Consider the inequalities related to this tour:

$$\begin{aligned} u_2 + 1 &\leq u_4 + 5(1 - x_{2,4}) \\ u_4 + 1 &\leq u_5 + 5(1 - x_{4,5}) \\ u_5 + 1 &\leq u_2 + 5(1 - x_{5,2}). \end{aligned}$$

A valid solution to this is

$$\begin{aligned} u_2 &= 2 \\ u_4 &= 3 \\ u_5 &= 4 \end{aligned}$$

$$\begin{aligned} 3 &\leq 3 + 5(1 - x_{2,4}) \\ 4 &\leq 4 + 5(1 - x_{4,5}) \\ 5 &\leq 2 + 5(1 - x_{5,2}). \end{aligned}$$

$$\begin{aligned} 0 &\leq 1 - x_{2,4} \\ 0 &\leq 1 - x_{4,5} \\ 3/5 &\leq 1 - x_{5,2}. \end{aligned}$$

$$\begin{aligned} 2 + 1 &\leq 3 + 5(1 - x_{2,4}) \\ 3 + 1 &\leq 4 + 5(1 - x_{4,5}) \\ 4 + 1 &\leq 2 + 5(1 - x_{5,2}). \end{aligned}$$

### 19.3.2. Dantzig-Fulkerson-Johnson (DFJ) Model

#### Resources

- Gurobi Modeling Example: TSP

This model does not add new variables. Instead, it adds constraints that conflict with the subtours. For instance, consider a subtour

$$2 \rightarrow 5 \rightarrow 3 \rightarrow 2. \quad (19.16)$$

We can prevent this subtour by adding the constraint

$$x_{25} + x_{53} + x_{32} \leq 2 \quad (19.17)$$

meaning that at most 2 of those arcs are allowed to happen at the same time. In general, for any subtour  $S$ , we can have the *subtour elimination constraint*

$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad \text{Subtour Elimination Constraint.} \quad (19.18)$$

In the previous example with  $S = \{(2,5), (5,3), (3,2)\}$  we have  $|S| = 3$ , where  $|S|$  denotes the size of the set  $S$ .

This model suggests that we just add all of these subtour elimination constraints.

#### Traveling Salesman Problem - DFJ Model:

$$\min \sum_{i,j \in N} c_{ij} x_{ij} \quad (19.19)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \quad [\text{outgoing arc}] \quad (19.20)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \quad [\text{incoming arc}] \quad (19.21)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \quad [\text{no self loops}] \quad (19.22)$$

$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad \text{for all subtours } S \quad [\text{prevents subtours}] \quad (19.23)$$

$$x_{ij} \in \{0, 1\} \quad \text{for all } i, j \in N \quad (19.24)$$

Distance Matrix:

A \ B	1	2	3	4
1	0	1	2	3
2	1	0	1	2
3	2	1	0	4
4	3	2	4	0

**Example 19.10: DFJ Model for  $n = 4$  nodes**

$$\begin{aligned} \min \quad & x_{1,2} + 2x_{1,3} + 3x_{1,4} + x_{2,1} + x_{2,3} + 2x_{2,4} \\ & + 2x_{3,1} + x_{3,2} + 4x_{3,4} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3} \end{aligned}$$

*Subject to*

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1 \quad \text{outgoing from node 1}$$

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1 \quad \text{outgoing from node 2}$$

$$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} = 1 \quad \text{outgoing from node 3}$$

$$x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} = 1 \quad \text{outgoing from node 4}$$

$$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} = 1 \quad \text{incoming to node 1}$$

$$x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1 \quad \text{incoming to node 2}$$

$$x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} = 1 \quad \text{incoming to node 3}$$

$$x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} = 1 \quad \text{incoming to node 4}$$

$$x_{1,1} = 0 \quad \text{No self loop with node 1}$$

$$x_{2,2} = 0 \quad \text{No self loop with node 2}$$

$$x_{3,3} = 0 \quad \text{No self loop with node 3}$$

$$x_{4,4} = 0 \quad \text{No self loop with node 4}$$

$$x_{1,2} + x_{2,1} \leq 1 \quad S = [(1,2), (2,1)]$$

$$x_{1,3} + x_{3,1} \leq 1 \quad S = [(1,3), (3,1)]$$

$$x_{1,4} + x_{4,1} \leq 1 \quad S = [(1,4), (4,1)]$$

$$x_{2,3} + x_{3,2} \leq 1 \quad S = [(2,3), (3,2)]$$

$$x_{2,4} + x_{4,2} \leq 1 \quad S = [(2,4), (4,2)]$$

$$x_{3,4} + x_{4,3} \leq 1 \quad S = [(3,4), (4,3)]$$

$$x_{2,1} + x_{1,3} + x_{3,2} \leq 2 \quad S = [(2,1), (1,3), (3,2)]$$

$$x_{1,2} + x_{2,3} + x_{3,1} \leq 2 \quad S = [(1,2), (2,3), (3,1)]$$

$$x_{3,1} + x_{1,4} + x_{4,3} \leq 2 \quad S = [(3,1), (1,4), (4,3)]$$

$$x_{1,3} + x_{3,4} + x_{4,1} \leq 2 \quad S = [(1,3), (3,4), (4,1)]$$

$$x_{2,1} + x_{1,4} + x_{4,2} \leq 2 \quad S = [(2,1), (1,4), (4,2)]$$

$$x_{1,2} + x_{2,4} + x_{4,1} \leq 2 \quad S = [(1,2), (2,4), (4,1)]$$

$$x_{3,2} + x_{2,4} + x_{4,3} \leq 2 \quad S = [(3,2), (2,4), (4,3)]$$

$$x_{2,3} + x_{3,4} + x_{4,2} \leq 2 \quad S = [(2,3), (3,4), (4,2)]$$

$$x_{i,j} \in \{0,1\} \quad \forall i \in \{1,2,3,4\}, j \in \{1,2,3,4\}$$

**Example 19.11**

Consider a graph on 5 nodes.

Here are all the subtours of length at least 3 and also including the full length tours.  
Hence, there are many subtours to consider.

**PROS OF THIS MODEL**

- Very tight linear relaxation

**CONS OF THIS MODEL**

- Exponentially many subtours  $S$  possible, hence this model is too large to write down.

**SOLUTION:** ADD SUBTOUR ELIMINATION CONSTRAINTS AS NEEDED. WE WILL DISCUSS THIS IN A FUTURE SECTION ON *cutting planes* .

### 19.3.3. Traveling Salesman Problem - Branching Solution

---

We will see in the next section

- That the constraint (19.1)-(19.3) always produce integer solutions as solutions to the linear relaxation.
- A way to use branch and bound (the topic of the next section) in order to avoid subtours.

### 19.3.4. Traveling Salesman Problem Variants

---

#### 19.3.4.1. Many salespersons (m-TSP)

---

**m-Traveling Salesman Problem - DFJ Model:**

$$\min \sum_{i,j \in N} c_{ij} x_{ij} \quad (19.25)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \text{ [outgoing arc]} \quad (19.26)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \text{ [incoming arc]} \quad (19.27)$$

$$\sum_{j \in N} x_{Dj} = m \quad \text{for all } i \in N \text{ [outgoing arc]} \quad (19.28)$$

$$\sum_{i \in N} x_{iD} = m \quad \text{for all } j \in N \text{ [incoming arc]} \quad (19.29)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \cup \{D\} \text{ [no self loops]} \quad (19.30)$$

$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad \text{for all subtours } S \subseteq N \text{ [prevents subtours]} \quad (19.31)$$

$$x_{ij} \in \{0, 1\} \quad \text{for all } i, j \in N \cup \{D\} \quad (19.32)$$

When using the MTZ model, you can also easily add in a constraint that restricts any subtour through the deopt to have at most  $T$  stops on the tour. This is done by restricting  $u_i \leq T$ . This could also be done in the DFJ model above, but the algorithm for subtour elimination cuts would need to be modified.

**m-Traveling Salesman Problem - MTZ Model - :**

## Python Code

$$\min \sum_{i,j \in N} c_{ij} x_{ij} \quad (19.33)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \text{ [outgoing arc]} \quad (19.34)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \text{ [incoming arc]} \quad (19.35)$$

$$\sum_{j \in N} x_{Dj} = m \quad \text{for all } i \in N \text{ [outgoing arc]} \quad (19.36)$$

$$\sum_{i \in N} x_{iD} = m \quad \text{for all } j \in N \text{ [incoming arc]} \quad (19.37)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \cup \{D\} \text{ [no self loops]} \quad (19.38)$$

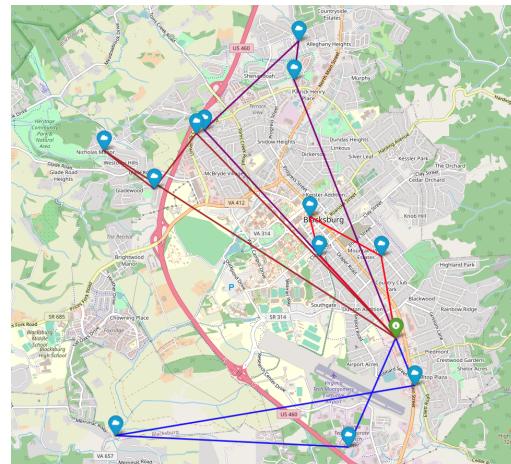
$$u_i + 1 \leq u_j + n(1 - x_{ij}) \quad \text{for all } i, j \in N, i, j \neq 1 \text{ [prevents subtours]} \quad (19.39)$$

$$u_1 = 1 \quad (19.40)$$

$$2 \leq u_i \leq T \quad \text{for all } i \in N, i \neq 1 \quad (19.41)$$

$$u_i \in \mathbb{Z} \quad \text{for all } i \in N \quad (19.42)$$

$$x_{ij} \in \{0, 1\} \quad \text{for all } i, j \in N \cup \{D\} \quad (19.43)$$



© m-tsp\_solution<sup>5</sup>

Image html

---

<sup>5</sup>m-tsp\_solution, from m-tsp\_solution. m-tsp\_solution, m-tsp\_solution.

### 19.3.4.2. TSP with order variants

---

Using the MTZ model, it is easy to provide order variants. Such as, city 2 must come before city 3

$$u_2 \leq u_3$$

or city 2 must come directly before city 3

$$u_2 + 1 = u_3.$$

## 19.4 Vehicle Routing Problem (VRP)

---

The VRP is a generalization of the TSP and comes in many many forms. The major difference is now we may consider multiple vehicles visiting the around cities. Obvious examples are creating bus schedules and mail delivery routes.

Variations of this problem include

- Time windows (for when a city needs to be visited)
- Prize collecting (possibly not all cities need to be visited, but you gain a prize for visiting each city)
- Multi-depot vehicle routing problem (fueling or drop off stations)
- Vehicle rescheduling problem (When delays have been encountered, how do you adjust the routes)
- Inhomogeneous vehicles (vehicles have different abilities (speed, distance, capacity, etc.).

To read about the many variants, see: *Vehicle Routing: Problems, Methods, and Applications*, Second Edition. Editor(s): Paolo Toth and Daniele Vigo. MOS-SIAM Series on Optimization.

For one example of a VRP model, see GUROBI Modeling Examples - technician routing scheduling.

### 19.4.1. Case Study: Bus Routing in Boston

---

Review this case study after studying algorithms and heuristics for integer programming.

[https://www.informs.org/Impact/O.R.-Analytics-Success-Stories/  
Optimized-school-bus-routing-helps-school-districts-design-better-policies](https://www.informs.org/Impact/O.R.-Analytics-Success-Stories/Optimized-school-bus-routing-helps-school-districts-design-better-policies)

<https://pubsonline.informs.org/doi/abs/10.1287/inte.2019.1015>

[https://www.informs.org/Resource-Center/Video-Library/  
Edelman-Competition-Videos/2019-Edelman-Competition-Videos/  
2019-Edelman-Finalist-Boston-Public-Schools](https://www.informs.org/Resource-Center/Video-Library/Edelman-Competition-Videos/2019-Edelman-Competition-Videos/2019-Edelman-Finalist-Boston-Public-Schools)

[https://www.youtube.com/watch?v=LFeeaNp\\_rbY](https://www.youtube.com/watch?v=LFeeaNp_rbY)

## 19.5 Steiner Tree Problem

---

Model 1

$$\min \sum_{(u,v) \in E} w_{uv} x_{uv}$$

such that

$$\begin{aligned} x_t &= 1 && \forall t \in T \\ 2x_{uv} - x_u - x_v &\leq 0 && \forall (u, v) \in E \\ x_v - \sum_{(u,v) \in E} x_{uv} &\leq 0 \forall v \in V \\ \sum_{(u,v) \in \delta(S)} x_{uv} &\geq x_w \forall S \subseteq V, \forall w \in S \end{aligned}$$

Model 2

$$\begin{aligned} \min \sum_{(u,v) \in E} w_{uv} x_{uv} \\ x_t &= 1 && \forall t \in T \\ 2x_{uv} - x_u - x_v &\leq 0 && \forall (u, v) \in E \\ x_v - \sum_{(u,v) \in E} x_{uv} &\leq 0 && \forall v \in V \\ x_{uv} + x_{vu} &\leq 1 \\ \sum_{v \in V} x_v - \sum_{(u,v) \in E} x_{uv} &= 1 \\ nx_{uv} + l_v - l_u &\geq 1 - n(1 - x_{vu}) && \forall (u, v) \in E \\ nx_{vu} + l_u - l_v &\geq 1 - n(1 - x_{uv}) && \forall (u, v) \in E \end{aligned}$$

Optimizing the shop footprint:

Step 1: Machine learning model predicts effect of opening or closing shops

Step 2: ILP Breaks down shop into manageable clusters

Step 3: ILP for optimal footprint planning

## 19.6 Literature and other notes

---

- Gilmore-Gomory Cutting Stock [**Gilmore-Gomory**]
- A Column Generation Algorithm for Vehicle Scheduling and Routing Problems
- The Integrated Last-Mile Transportation Problem

- [http://www.optimization-online.org/DB\\_FILE/2017/11/6331.pdf](http://www.optimization-online.org/DB_FILE/2017/11/6331.pdf) A BRANCH-AND-PRICE ALGORITHM FOR CAPACITATED HYPERGRAPH VERTEX SEPARATION

### **19.6.1. Google maps data**

---

Blog - Python | Calculate distance and duration between two places using google distance matrix API

### **19.6.2. TSP In Excel**

---

TSP with excel solver

# 20. Algorithms to Solve Integer Programs

## 20.1 LP to solve IP

Recall that the linear relaxation of an integer program is the linear programming problem after removing the integrality constraints

Integer Program:

$$\begin{aligned} \max \quad & z_{IP} = c^\top x \\ & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

Linear Relaxation:

$$\begin{aligned} \max \quad & z_{LP} = c^\top x \\ & Ax \leq b \\ & x \in \mathbb{R}^n \end{aligned}$$

### Theorem 20.1: LP Bounds

*It always holds that*

$$z_{IP}^* \leq z_{LP}^*. \quad (20.1)$$

*Furthermore, if  $x_{LP}^*$  is integral (feasible for the integer program), then*

$$x_{LP}^* = x_{IP}^* \quad \text{and} \quad z_{LP}^* = z_{IP}^*. \quad (20.2)$$

### Example 20.2

*Consider the problem*

$$\begin{aligned} \max z = & 3x_1 + 2x_2 \\ & 2x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0; x_1, x_2 \text{ integer} \end{aligned}$$

### 20.1.1. Rounding LP Solution can be bad!

---

#### Resources

*Video! - Michel Belaire (EPFL) looking at rounding the LP solution to an IP solution*

Consider the two variable knapsack problem

$$\max 3x_1 + 100x_2 \quad (20.3)$$

$$x_1 + 100x_2 \leq 100 \quad (20.4)$$

$$x_i \in \{0, 1\} \text{ for } i = 1, 2. \quad (20.5)$$

Then  $x_{LP}^* = (1, 0.99)$  and  $z_{LP}^* = 1 \cdot 3 + 0.99 \cdot 100 = 3 + 99 = 102$ .

But  $x_{IP}^* = (0, 1)$  with  $z_{IP}^* = 0 \cdot 3 + 1 \cdot 100 = 100$ .

Suppose that we rounded the LP solution.

$x_{LP-Rounded-Down}^* = (1, 0)$ . Then  $z_{LP-Rounded-Down}^* = 1 \cdot 3 = 3$ . Which is a terrible solution!

How can we avoid this issue?

Cool trick! Using two different strategies gives you at least a 1/2 approximation to the optimal solution.

### 20.1.2. Rounding LP solution can be infeasible!

---

Now only could it produce a poor solution, it is not always clear how to round to a feasible solution.

### 20.1.3. Fractional Knapsack

---

The fractional knapsack problem has an exact greedy algorithm.

#### Resources

- [Youtube!](#)
- [Blog](#)

## 20.2 Branch and Bound

---

#### Resources

*Video! - Michel Belaire (EPFL) Teaching Branch and Bound Theory Video! - Michel Belaire (EPFL) Teaching Branch and Bound with Example*

*See Module by Miguel Casquilho for some nice notes on branch and bound.*

## 20.2.1. Algorithm

### Algorithm 6 Branch and Bound - Maximization

**Require:** Integer Linear Problem with max objective

**Ensure:** Exact Optimal Solution  $x^*$

- 1: Set  $LB = -\infty$ .
- 2: Solve LP relaxation.
  - a: If  $x^*$  is integer, stop!
  - b: Otherwise, choose fractional entry  $x_i^*$  and branch onto subproblems: (i)  $x_i \leq \lfloor x_i^* \rfloor$  and (ii)  $x_i \geq \lceil x_i^* \rceil$ .
- 3: Solve LP relaxation of any subproblem.
  - a: If LP relaxation is infeasible, prune this node as "**Infeasible**"
  - b: If  $z^* < LB$ , prune this node as "**Suboptimal**"
  - c:  $x^*$  is integer, prune this nodes as "**Integer**" and update  $LB = \max(LB, z^*)$ .
  - d: Otherwise, choose fractional entry  $x_i^*$  and branch onto subproblems: (i)  $x_i \leq \lfloor x_i^* \rfloor$  and (ii)  $x_i \geq \lceil x_i^* \rceil$ .
- Return to step 2 until all subproblems are pruned.

- 4: Return best integer solution found.

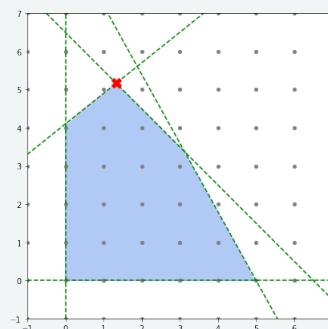
Here is an example of branching on general integer variables.

### Example 20.3

Consider the two variable example with

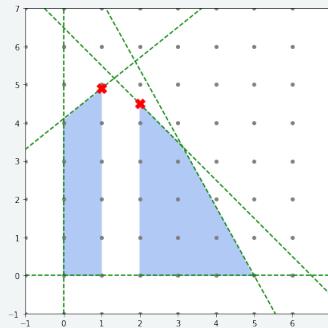
$$\begin{aligned}
 & \max -3x_1 + 4x_2 \\
 & 2x_1 + 2x_2 \leq 13 \\
 & -8x_1 + 10x_2 \leq 41 \\
 & 9x_1 + 5x_2 \leq 45 \\
 & 0 \leq x_1 \leq 10, \text{ integer} \\
 & 0 \leq x_2 \leq 10, \text{ integer}
 \end{aligned}$$

$$x = [1.33, 5.167] \text{ obj} = 16.664$$



$$x = [1, 4.9] \text{ obj} = 16.5998$$

$$x = [2, 4.5] \text{ obj} = 12.0$$



© branch-and-bound2<sup>2</sup>

---

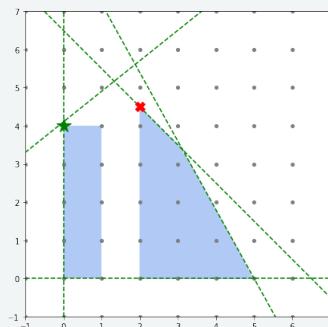
branch-and-bound1, from branch-and-bound1. branch-and-bound1, branch-and-bound1.  
 branch-and-bound2, from branch-and-bound2. branch-and-bound2, branch-and-bound2.

#### Example 20.4: Example continued

*Infeasible Region*

$$x = [0, 4] \text{ obj} = 16.0$$

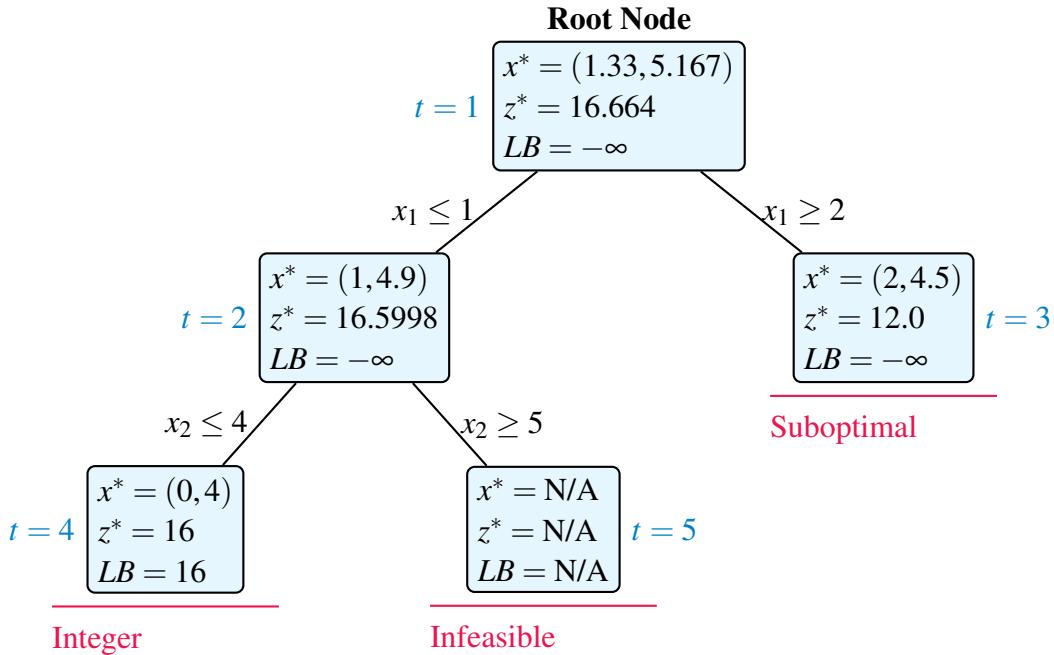
$$x = [2, 4.5] \text{ obj} = 12.0$$



© branch-and-bound3<sup>3</sup>

---

branch-and-bound3, from branch-and-bound3. branch-and-bound3, branch-and-bound3.



## 20.2.2. Knapsack Problem and 0/1 branching

---

Consider the problem

$$\begin{aligned} \max \quad & 16x_1 + 22x_2 + 12x_3 + 8x_4 \\ \text{s.t. } & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & 0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4 \\ & x_i \in \{0, 1\} \quad i = 1, 2, 3, 4 \end{aligned}$$

**Question:** What is the optimal solution if we remove the binary constraints?

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 \\ \text{s.t. } & a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 \leq b \\ & 0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4 \end{aligned}$$

**Question:** How do I find the solution to this problem?

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 \\ \text{s.t. } & (a_1 - A)x_1 + (a_2 - A)x_2 + (a_3 - A)x_3 + (a_4 - A)x_4 \leq 0 \\ & 0 \leq x_i \leq m_i \quad i = 1, 2, 3, 4 \end{aligned}$$

**Question: How do I find the solution to this problem?**

Consider the problem

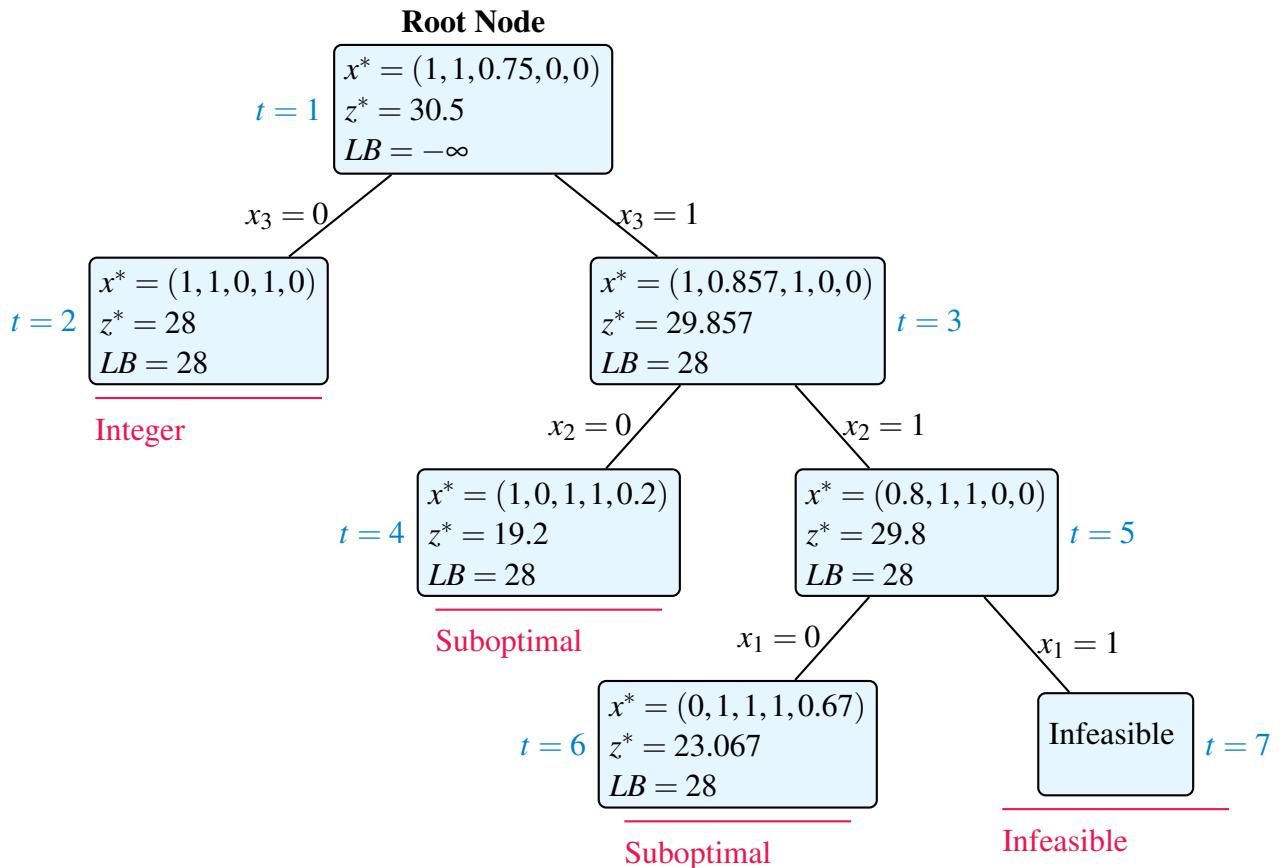
$$\begin{aligned} \max \quad & 16x_1 + 22x_2 + 12x_3 + 8x_4 \\ \text{s.t. } & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & 0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4 \\ & x_i \in \{0, 1\} \quad i = 1, 2, 3, 4 \end{aligned}$$

We can solve this problem with branch and bound.

The optimal solution was found at  $t = 5$  at subproblem 6 to be  $x^* = (0, 1, 1, 1)$ ,  $z^* = 42$ .

**Example: Binary Knapsack** Solve the following problem with branch and bound.

$$\begin{aligned} \max \quad & z = 11x_1 + 15x_2 + 6x_3 + 2x_4 + x_5 \\ \text{Subject to: } & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 15x_5 \leq 15 \\ & x_i \text{ binary}, i = 1, \dots, 5 \end{aligned}$$



### 20.2.3. Traveling Salesman Problem solution via Branching

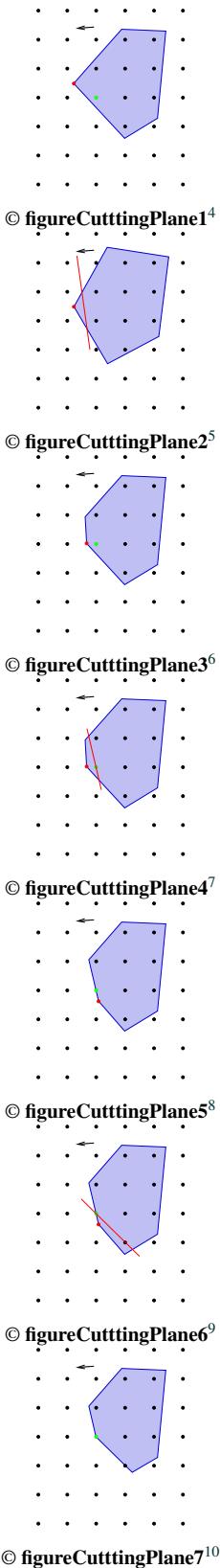
escribe solving TSP via a generalized branching method that removes subtours (instead of adding constraints).

## 20.3 Cutting Planes

Cutting planes are inequalities  $\pi^\top x \leq \pi_0$  that are valid for the feasible integer solutions that the cut off part of the LP relaxation. Cutting planes can create a tighter description of the feasible region that allows for the optimal solution to be obtained by simply solving a strengthened linear relaxation.

The cutting plane procedure, as demonstrated in Figure 20.1. The procedure is as follows:

- Solve the current LP relaxation.
- If solution is integral, then return that solution. STOP
- Add a cutting plane (or many cutting planes) that cut off the LP-optimal solution.
- Return to Step 1.



**Figure 20.1: The cutting plane procedure.**

In practice, this procedure is integrated in some with with branch and bound and also other primal

heuristics.

### 20.3.1. Chvátal Cuts

Chvátal Cuts are a general technique to produce new inequalities that are valid for feasible integer points.

#### Chvátal Cuts:

Suppose

$$a_1x_1 + \cdots + a_nx_n \leq d \quad (20.1)$$

is a valid inequality for the polyhedron  $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ , then

$$\lfloor a_1 \rfloor x_1 + \cdots + \lfloor a_n \rfloor x_n \leq \lfloor d \rfloor \quad (20.2)$$

is valid for the integer points in  $P$ , that is, it is valid for the set  $P \cap \mathbb{Z}^n$ . Equation (20.2) is called a Chvátal Cut.

We will illustrate this idea with an example.

#### Example 20.5

Recall example ?? . The model was

##### Model

$$\begin{array}{lll} \min & p + n + d + q & \text{total number of coins used} \\ \text{s.t.} & p + 5n + 10d + 25q = 83 & \text{sums to } 83\text{¢} \\ & p, d, n, q \in \mathbb{Z}_+ & \text{each is a non-negative integer} \end{array}$$

From the equality constraint we can derive several inequalities.

(a) Divide by 25 and round down both sides:

$$\frac{p + 5n + 10d + 25q}{25} = 83/25 \Rightarrow q \leq 3$$

(b) Divide by 10 and round down both sides:

$$\frac{p + 5n + 10d + 25q}{10} = 83/10 \Rightarrow d + 2q \leq 8$$

(c) Divide by 5 and round down both sides:

$$\frac{p + 5n + 10d + 25q}{5} = 83/5 \Rightarrow n + 2d + 5q \leq 16$$

(d) Multiply by 0.12 and round down both sides:

$$0.12(p + 5n + 10d + 25q) = 0.12(83) \Rightarrow d + 3q \leq 9$$

These new inequalities are all valid for the integer solutions. Consider the new model:

### New Model

$$\begin{array}{ll}
 \min & p + n + d + q && \text{total number of coins used} \\
 \text{s.t.} & p + 5n + 10d + 25q = 83 && \text{sums to } 83\text{¢} \\
 & q \leq 3 \\
 & d + 2q \leq 8 \\
 & n + 2d + 5q \leq 16 \\
 & d + 3q \leq 9 \\
 & p, d, n, q \in \mathbb{Z}_+ && \text{each is a non-negative integer}
 \end{array}$$

The solution to the LP relaxation is exactly  $q = 3, d = 0, n = 1, p = 3$ , which is an integral feasible solution, and hence it is an optimal solution.

### 20.3.2. Gomory Cuts

---

#### Resources

- Pascal Van Hyndryk (Georgia Tech) Teaching Gomory Cuts
- Michel Bierlaire (EPFL) Teaching Gomory Cuts

Gomory cuts are a type of Chvátal cut that is derived from the simplex tableau. Specifically, suppose that

$$x_i + \sum_{i \in N} \tilde{a}_i x_i = \tilde{b}_i \quad (20.3)$$

is an equation in the optimal simplex tableau.

#### Gomory Cut:

The Gomory cut corresponding to the tableau row (20.3) is

$$\sum_{i \in N} (\tilde{a}_i - \lfloor \tilde{a}_i \rfloor) x_i \geq \tilde{b}_i - \lfloor \tilde{b}_i \rfloor \quad (20.4)$$

We will solve the following problem using only Gomory Cuts.

$$\begin{array}{ll}
 \min & x_1 - 2x_2 \\
 \text{s.t.} & -4x_1 + 6x_2 \leq 9 \\
 & x_1 + x_2 \leq 4 \\
 & x \geq 0 , \quad x_1, x_2 \in \mathbb{Z}
 \end{array}$$

**Step 1:** The first thing to do is to put this into standard form by appending slack variables.

$$\begin{array}{ll} \min & x_1 - 2x_2 \\ \text{s.t.} & -4x_1 + 6x_2 + s_1 = 9 \\ & x_1 + x_2 + s_2 = 4 \\ & x \geq 0, \quad x_1, x_2 \in \mathbb{Z} \end{array} \quad (20.5)$$

We can apply the simplex method to solve the LP relaxation.

	Basis	RHS	$x_1$	$x_2$	$s_1$	$s_2$
Initial Basis	$z$	0.0	1.0	-2.0	0.0	0.0
	$s_1$	9.0	-4.0	6.0	1.0	0.0
	$s_2$	4.0	1.0	1.0	0.0	1.0
	⋮	⋮				
Optimal Basis	$z$	-3.5	0.0	0.0	0.3	0.2
	$x_1$	1.5	1.0	0.0	-0.1	0.6
	$x_2$	2.5	0.0	1.0	0.1	0.4

This LP relaxation produces the fractional basic solution  $x_{LP} = (1.5, 2.5)$ .

### Example 20.6

**(Gomory cut removes LP solution)** We now identify an integer variable  $x_i$  that has a fractional basic solution. Since both variables have fractional values, we can choose either row to make a cut. Let's focus on the row corresponding to  $x_1$ .

The row from the tableau expresses the equation

$$x_1 - 0.1s_1 + -0.6s_2 = 1.5. \quad (20.6)$$

Applying the Gomory Cut (20.4), we have the inequality

$$0.9s_1 + 0.4s_2 \geq 0.5. \quad (20.7)$$

The current LP solution is  $(x_{LP}, s_{LP}) = (1.5, 2.5, 0, 0)$ . Trivially, since  $s_1, s_2 = 0$ , the inequality is violated.

### Example 20.7: (Gomory Cut in Original Space)

The Gomory Cut (20.7) can be rewritten in the original variables using the equations from (20.5). That is, we can use the equations

$$\begin{aligned} s_1 &= 9 + 4x_1 - 6x_2 \\ s_2 &= 4 - x_1 - x_2, \end{aligned} \quad (20.8)$$

which transforms the Gomory cut into the original variables to create the inequality

$$0.9(9 + 4x_1 - 6x_2) + 0.4(4 - x_1 - x_2) \geq 0.5.$$

or equivalently

$$-3.2x_1 + 5.8x_2 \leq 9.2. \quad (20.9)$$

As you can see, this inequality does cut off the current LP relaxation.

### Example 20.8: (Gomory cuts plus new tableau)

Now we add the slack variable  $s_3 \geq 0$  to make the equation

$$0.9s_1 + 0.4s_2 - s_3 = 0.5. \quad (20.10)$$

Next, we need to solve the linear programming relaxation (where we assume the variables are continuous).

#### 20.3.3. Cover Inequalities

---

Consider the binary knapsack problem

$$\begin{aligned} \max \quad & x_1 + 2x_2 + x_3 + 7x_4 \\ \text{s.t.} \quad & 100x_1 + 70x_2 + 50x_3 + 60x_4 \leq 150 \\ & x_i \text{ binary for } i = 1, \dots, 4 \end{aligned}$$

A *cover*  $S$  is any subset of the variables whose sum of weights exceed the capacity of the right hand side of the inequality.

For example,  $S = \{1, 2, 3, 4\}$  is a cover since  $100 + 70 + 50 + 60 > 150$ .

Since not all variables in the cover  $S$  can be in the knapsack simultaneously, we can enforce the *cover inequality*

$$\sum_{i \in S} x_i \leq |S| - 1 \Rightarrow x_1 + x_2 + x_3 + x_4 \leq 4 - 1 = 3. \quad (20.11)$$

Note, however, that there are other covers that use fewer variables.

A *minimal cover* is a subset of variables such that no other subset of those variables is also a cover. For example, consider the cover  $S' = \{1, 2\}$ . This is a cover since  $100 + 70 > 150$ . Since  $S'$  is a subset of  $S$ , the cover  $S$  is not a minimal cover. In fact,  $S'$  is a minimal cover since there are no smaller subsets of the set  $S'$  that also produce a cover. In this case, we call the corresponding inequality a *minimal cover inequality*. That is, the inequality

$$x_1 + x_2 \leq 2 - 1 = 1 \quad (20.12)$$

is a minimal cover inequality for this problem. The minimal cover inequalities are the "strongest" of all cover inequalities.

*Find the two other minimal covers (one of size 2 and one of size 3) and write their corresponding minimal cover inequalities.*

**Solution.** The other minimal covers are

$$S = \{1, 4\} \Rightarrow x_1 + x_4 \leq 1 \quad (20.13)$$

and

$$S = \{2, 3, 4\} \Rightarrow x_2 + x_3 + x_4 \leq 2 \quad (20.14)$$



## 20.4 Branching Rules

---

There is a few clever ideas out there on how to choose which variables to branch on. We will not go into this here. But for the interested reader, look into

- Strong Branching
- Pseudo-cost Branching

## 20.5 Lagrangian Relaxation for Branch and Bound

---

At each note in the branch and bound tree, we want to bound the objective value. One way to get a good bound can be using the Lagrangian.

### Resources

See [Fisher2004] ([link](#)) for a description of this.

## 20.6 Benders Decomposition

---

### Resources

*Benders Decomposition - Julia Opt*  
*Youtube! SCIP lecture*

## 20.7 Literature

---

## 20.8 Other material for Integer Linear Programming

---

Recall the problem on lemonade and lemon juice from Chapter 9.1:

**Problem.** Say you are a vendor of lemonade and lemon juice. Each unit of lemonade requires 1 lemon and 2 litres of water. Each unit of lemon juice requires 3 lemons and 1 litre of water. Each unit of lemonade gives a profit of \$3. Each unit of lemon juice gives a profit of \$2. You have 6 lemons and 4 litres of water available. How many units of lemonade and lemon juice should you make to maximize profit?

Letting  $x$  denote the number of units of lemonade to be made and letting  $y$  denote the number of units of lemon juice to be made, the problem could be formulated as the following linear programming problem:

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & x + 3y \leq 6 \\ & 2x + y \leq 4 \\ & x \geq 0 \\ & y \geq 0. \end{aligned}$$

The problem has a unique optimal solution at  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.6 \end{bmatrix}$  for a profit of 6.8. But this solution requires us to make fractional units of lemonade and lemon juice. What if we require the number of units to be integers? In other words, we want to solve

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & x + 3y \leq 6 \\ & 2x + y \leq 4 \\ & x \geq 0 \\ & y \geq 0 \\ & x, y \in \mathbb{Z}. \end{aligned}$$

This problem is no longer a linear programming problem. But rather, it is an integer linear programming problem.

A **mixed-integer linear programming problem** is a problem of minimizing or maximizing a linear function subject to finitely many linear constraints such that the number of variables are finite and at least one of which is required to take on integer values.

If all the variables are required to take on integer values, the problem is called a **pure integer linear programming problem** or simply an **integer linear programming problem**. Normally, we assume the problem data to be rational numbers to rule out some pathological cases.

Mixed-integer linear programming problems are in general difficult to solve yet they are too important to ignore because they have a wide range of applications (e.g. transportation planning, crew scheduling, circuit design, resource management etc.) Many solution methods for these problems have been devised and some of them first solve the **linear programming relaxation** of the original problem, which is the problem obtained from the original problem by dropping all the integer requirements on the variables.

### Example 20.9

Let (MP) denote the following mixed-integer linear programming problem:

$$\begin{array}{lllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 & + & x_2 & + x_3 \geq 1 \\ & -x_1 & - & x_2 & + 2x_3 \geq 0 \\ & -x_1 & + & 5x_2 & - x_3 = 3 \\ & x_1, & x_2, & x_3 & \geq 0 \\ & & & & x_3 \in \mathbb{Z}. \end{array}$$

The linear programming relaxation of (MP) is:

$$\begin{array}{lllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 & + & x_2 & + x_3 \geq 1 \\ & -x_1 & - & x_2 & + 2x_3 \geq 0 \\ & -x_1 & + & 5x_2 & - x_3 = 3 \\ & x_1, & x_2, & x_3 & \geq 0. \end{array}$$

Let (P1) denote the linear programming relaxation of (MP). Observe that the optimal value of (P1) is a lower bound for the optimal value of (MP) since the feasible region of (P1) contains all the feasible solutions to (MP), thus making it possible to find a feasible solution to (P1) with objective function value better than the optimal value of (MP). Hence, if an optimal solution to the linear programming relaxation happens to be a feasible solution to the original problem, then it is also an optimal solution to the original problem. Otherwise, there is an integer variable having a nonintegral value  $v$ . What we then do is to create two new subproblems as follows: one requiring the variable to be at most the greatest integer less than  $v$ , the other requiring the variable to be at least the smallest integer greater than  $v$ . This is the basic idea behind the **branch-and-bound method**. We now illustrate these ideas on (MP).

Solving the linear programming relaxation (P1), we find that  $\mathbf{x}' = \begin{bmatrix} 0 \\ \frac{2}{3} \\ \frac{1}{3} \end{bmatrix}$  is an optimal solution to (P1). Note that  $\mathbf{x}'$  is not a feasible solution to (MP) because  $x'_3$  is not an integer. We now create two subproblems (P2) and (P3) such that (P2) is obtained from (P1) by adding the constraint  $x_3 \leq \lfloor x'_3 \rfloor$  and (P3) is obtained from (P1) by adding the constraint  $x_3 \geq \lceil x'_3 \rceil$ . (For a number  $a$ ,  $\lfloor a \rfloor$  denotes the

greatest integer at most  $a$  and  $\lceil a \rceil$  denotes the smallest integer at least  $a$ .) Hence, (P2) is the problem

$$\begin{array}{llllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 & + & x_2 & + & x_3 \geq 1 \\ & -x_1 & - & x_2 & + & 2x_3 \geq 0 \\ & -x_1 & + & 5x_2 & - & x_3 = 3 \\ & & & & x_3 \leq 0 \\ & x_1, & x_2, & x_3 \geq 0, \end{array}$$

and (P3) is the problem

$$\begin{array}{llllll} \min & x_1 & + & x_3 \\ \text{s.t.} & -x_1 & + & x_2 & + & x_3 \geq 1 \\ & -x_1 & - & x_2 & + & 2x_3 \geq 0 \\ & -x_1 & + & 5x_2 & - & x_3 = 3 \\ & & & & x_3 \geq 1 \\ & x_1, & x_2, & x_3 \geq 0. \end{array}$$

Note that any feasible solution to (MP) must be a feasible solution to either (P2) or (P3). Using the help of a solver, one sees that (P2) is infeasible. The problem (P3) has an optimal solution at

$$\mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{4}{5} \\ 1 \end{bmatrix}, \text{ which is also feasible to (MP). Hence, } \mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{4}{5} \\ 1 \end{bmatrix} \text{ is an optimal solution to (MP).}$$

We now give a description of the method for a general mixed-integer linear programming problem (MIP). Suppose that (MIP) is a minimization problem and has  $n$  variables  $x_1, \dots, x_n$ . Let  $\mathcal{I} \subseteq \{1, \dots, n\}$  denote the set of indices  $i$  such that  $x_i$  is required to be an integer in (MIP).

### Branch-and-bound method

**Input:** The problem (MIP).

**Steps:**

- Set  $\text{bestbound} := \infty$ ,  $\mathbf{x}_{\text{best}}^* := \text{N/A}$ ,  $\text{activeproblems} := \{(LP)\}$  where  $(LP)$  denotes the linear programming relaxation of (MIP).
- If there is no problem in  $\text{activeproblems}$ , then stop; if  $\mathbf{x}_{\text{best}}^* \neq \text{N/A}$ , then  $\mathbf{x}_{\text{best}}^*$  is an optimal solution; otherwise, (MIP) has no optimal solution.
- Select a problem  $P$  from  $\text{activeproblems}$  and remove it from  $\text{activeproblems}$ .
- Solve  $P$ .
  - If  $P$  is unbounded, then stop and conclude that (MIP) does not have an optimal solution.
  - If  $P$  is infeasible, go to step 2.
  - If  $P$  has an optimal solution  $\mathbf{x}^*$ , then let  $z$  denote the objective function value of  $\mathbf{x}^*$ .
    - If  $z \geq \text{bestbound}$ , go to step 2.
    - If  $x_i^*$  is not an integer for some  $i \in \mathcal{I}$ , then create two subproblems  $P_1$  and  $P_2$  such that  $P_1$  is the problem obtained from  $P$  by adding the constraint  $x_i \leq \lfloor x_i^* \rfloor$  and  $P_2$  is the problem obtained from  $P$  by adding the constraint  $x_i \geq \lceil x_i^* \rceil$ . Add the problems  $P_1$  and  $P_2$  to  $\text{activeproblems}$  and go to step 2.
    - Set  $\mathbf{x}_{\text{best}}^* = \mathbf{x}^*$ ,  $\text{bestbound} = z$  and go to step 2.

**Remarks.**

- Throughout the algorithm, `activeproblems` is a set of subproblems remained to be solved. Note that for each problem  $P$  in `activeproblems`,  $P$  is a linear programming problem and that any feasible solution to  $P$  satisfying the integrality requirements is a feasible solution to (MIP).
- $x_{\text{best}}^*$  is the feasible solution to (MIP) that has the best objective function value found so far and `bestbound` is its objective function value. It is often called an **incumbent**.
- In practice, how a problem from `activeproblems` is selected in step 3 has an impact on the overall performance. However, there is no general rule for selection that guarantees good performance all the time.
- In step 5, the problem  $P$  is discarded since it cannot contain any feasible solution to (MIP) having a better objective function value than  $x_{\text{best}}^*$ .
- If step 7 is reached, then  $x^*$  is a feasible solution to (MIP) having objective function value better than `bestbound`. So it becomes the current best solution.
- It is possible for the algorithm to never terminate. Below is an example for which the algorithm will never stop:

$$\begin{aligned} \min \quad & x_1 \\ \text{s.t.} \quad & x_1 + 2x_2 - 2x_3 = 1 \\ & x_1, x_2, x_3 \geq 0 \\ & x_1, x_2, x_3 \in \mathbb{Z}. \end{aligned}$$

However, it is easy to see that  $\mathbf{x}^* = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$  is an optimal solution because there is no feasible solution with  $x_1 = 0$ .

One way to keep track of the progress of the computations is to set up a progress chart with the following headings:

Iter	solved	status	branching	activeproblems	$\mathbf{x}_{\text{best}}^*$	bestbound
------	--------	--------	-----------	----------------	------------------------------	-----------

In a given iteration, the entry in the **solved** column denotes the subproblem that has been solved with the result in the **status** column. The **branching** column indicates the subproblems created from the solved subproblem with an optimal solution not feasible to (MIP). The entries in the remaining columns contain the latest information in the given iteration. For the example (MP) above, the chart could look like the following:

Iter	solved	status	branching	activeproblems	$\mathbf{x}_{\text{best}}^*$	bestbound
1	(P1)	optimal	(P2): $x_3 \leq 0$ , $\mathbf{x}^* = \begin{bmatrix} 0 \\ 0 \\ \frac{2}{3} \end{bmatrix}$ (P3): $x_3 \geq 1$	(P2), (P3)	N/A	$\infty$

Iter	solved	status	branching	active problems	$\mathbf{x}^*$ <sub>best</sub>	bestbound
2	(P2)	infeasible	—	(P3)	N/A	$\infty$
3	(P3)	optimal $\mathbf{x}^* = \begin{bmatrix} 0 \\ \frac{4}{5} \\ \frac{5}{5} \\ 1 \end{bmatrix}$	—	—	$\begin{bmatrix} 0 \\ \frac{4}{5} \\ \frac{5}{5} \\ 1 \end{bmatrix}$	1

## Exercises

---

- (a) Suppose that (MP) in Example 20.8 above has  $x_2$  required to be an integer as well. Continue with the computations and determine an optimal solution to the modified problem.  
(b) With the help of a solver, determine the optimal value of

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & x + 3y \leq 6 \\ & 2x + y \leq 4 \\ & x, y \geq 0 \\ & x, y \in \mathbb{Z}. \end{aligned}$$

- (c) Let  $\mathbf{A} \in \mathbb{Q}^{m \times n}$  and  $\mathbf{b} \in \mathbb{Q}^m$ . Let  $S$  denote the system

$$\begin{aligned} \mathbf{Ax} &\geq \mathbf{b} \\ \mathbf{x} &\in \mathbb{Z}^n \end{aligned}$$

- i. Suppose that  $\mathbf{d} \in \mathbb{Q}^m$  satisfies  $\mathbf{d} \geq 0$  and  $\mathbf{d}^\top \mathbf{A} \in \mathbb{Z}^n$ . Prove that every  $\mathbf{x}$  satisfying  $S$  also satisfies  $\mathbf{d}^\top \mathbf{Ax} \geq \lceil \mathbf{d}^\top \mathbf{b} \rceil$ . (This inequality is known as a **Chvátal-Gomory cutting plane**.)  
ii. Suppose that  $\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 5 & 3 \\ 7 & 6 \end{bmatrix}$  and  $\mathbf{b} = \begin{bmatrix} 2 \\ 1 \\ 8 \end{bmatrix}$ . Show that every  $\mathbf{x}$  satisfying  $S$  also satisfies  $x_1 + x_2 \geq 2$ .

## Solutions

---

- (a) An optimal solution to the modified problem is given by  $x^* = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ .  
(b) An optimal solution is  $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ . Thus, the optimal value is 6.  
(c) i. Since  $\mathbf{d} \geq 0$  and  $\mathbf{Ax} \geq \mathbf{b}$ , we have  $\mathbf{d}^\top \mathbf{Ax} \geq \mathbf{d}^\top \mathbf{b}$ . If  $\mathbf{d}^\top \mathbf{b}$  is an integer, the result follows immediately. Otherwise, note that  $\mathbf{d}^\top \mathbf{A} \in \mathbb{Z}^n$  and  $\mathbf{x} \in \mathbb{Z}^n$  imply that  $\mathbf{d}^\top \mathbf{Ax}$  is an integer. Thus,  $\mathbf{d}^\top \mathbf{Ax}$  must be greater than or equal to the least integer greater than  $\mathbf{d}^\top \mathbf{b}$ .

ii. Take  $\mathbf{d} = \begin{bmatrix} \frac{1}{9} \\ 0 \\ \frac{1}{9} \end{bmatrix}$  and apply the result in the previous part.

### 20.8.1. Other discrete problems

---

### 20.8.2. Assignment Problem and the Hungarian Algorithm

---

#### Assignment Problem:

*Polynomial time (P)*

$$\begin{aligned}
 & \min \langle C, X \rangle \\
 \text{s.t. } & \sum_i X_{ij} = 1 \text{ for all } j \\
 & \sum_j X_{ij} = 1 \text{ for all } i \\
 & X_{ij} \in \{0, 1\} \text{ for all } i = 1, \dots, n, j = 1, \dots, m
 \end{aligned} \tag{20.1}$$

This problem is efficiently solved by the Hungarian Algorithm.

### 20.8.3. History of Computation in Combinatorial Optimization

---

Book: Computing in Combinatorial Optimization by William Cook, 2019

Model	LP Solution
$\max \quad x_1 + x_2$ subject to $-2x_1 + x_2 \leq 0.5$ $x_1 + 2x_2 \leq 10.5$ $x_1 - x_2 \leq 0.5$ $-2x_1 - x_2 \leq -2$	<p>A 2D plot with x-axis labeled <math>x_1</math> and y-axis labeled <math>x_2</math>, both ranging from 0 to 5. A shaded blue polygon represents the feasible region for the LP relaxation. The vertices of this polygon are approximately at (0, 0), (0.5, 1), (1, 2), (2, 3), (3, 4), and (4, 3). Several yellow dots are scattered within the shaded region. A blue star marks the corner point (3, 3) where <math>x_1 = 3</math> and <math>x_2 = 3</math>.</p>
$\max \quad x_1 + x_2$ subject to $-2x_1 + x_2 \leq 0.5$ $x_1 + 2x_2 \leq 10.5$ $x_1 - x_2 \leq 0.5$ $-2x_1 - x_2 \leq -2$ $x_1 \leq 3$	<p>A 2D plot with x-axis labeled <math>x_1</math> and y-axis labeled <math>x_2</math>, both ranging from 0 to 5. A shaded blue polygon represents the feasible region for the LP relaxation. The vertices of this polygon are approximately at (0, 0), (0.5, 1), (1, 2), (2, 3), (3, 4), and (3, 3). Several yellow dots are scattered within the shaded region. A blue star marks the corner point (3, 3) where <math>x_1 = 3</math> and <math>x_2 = 3</math>.</p>
$\max \quad x_1 + x_2$ subject to $-2x_1 + x_2 \leq 0.5$ $x_1 + 2x_2 \leq 10.5$ $x_1 - x_2 \leq 0.5$ $-2x_1 - x_2 \leq -2$ $x_1 \leq 3$ $x_1 + x_2 \leq 6$	<p>A 2D plot with x-axis labeled <math>x_1</math> and y-axis labeled <math>x_2</math>, both ranging from 0 to 5. A shaded blue polygon represents the feasible region for the LP relaxation. The vertices of this polygon are approximately at (0, 0), (0.5, 1), (1, 2), (2, 3), (3, 4), and (3, 3). Several yellow dots are scattered within the shaded region. A blue star marks the corner point (3, 3) where <math>x_1 = 3</math> and <math>x_2 = 3</math>.</p>

# 21. Heuristics for TSP

---

**Resources**



- VRP Heuristic Approach Lecture by Flipkart Delivery Hub

In this section we will show how different heuristics can find good solutions to TSP. For convenience, we will focus on the *symmetric TSP* problem. That is, the distance  $d_{ij}$  from node  $i$  to node  $j$  is the same as the distance  $d_{ji}$  from node  $j$  to node  $i$ .

There are two general types of heuristics: construction heuristics and improvement heuristics. We will first discuss a few construction heuristics for TSP.

Then we will demonstrate three types of metaheuristics for improvement- Hill Climbing, Tabu Search, and Simulated Annealing. These are called *metaheuristics* because they are a general framework for a heuristic that can be applied to many different types of settings. Furthermore, Tabu Search and Simulated Annealing have parameters that can be adjusted to try to find better solutions.

## 21.1 Construction Heuristics

---

### 21.1.1. Random Solution

---

TSP is convenient in that choosing a random ordering of the nodes creates a feasible solution. It may not be a very good one, but it is at least a solution.

**Random Construction:**Complexity:  $O(n)$ For  $i = 1, \dots, n$ , randomly choose a node not yet in the tour and place it at the end of the tour.

### 21.1.2. Nearest Neighbor

---

Starting from any node, add the edge to the next closest node. Continue this process.

**Nearest Neighbor:**Complexity:  $O(n^2)$ 

- Start from any node (lets call this node 1) and label this as your current node.
- Pick the next current node as the one that is closest to the current node that has not yet been visited.
- Repeat step 2 until all nodes are in the tour.

### 21.1.3. Insertion Method

---

**Insertion Method:**Complexity:  $O(n^2)$ 

- Start from any 3 nodes (lets call this node 1) and label this as your current node.
- Pick the next current node as the one that is closest to the current node that has not yet been visited.
- Repeat step 2 until all nodes are in the tour.

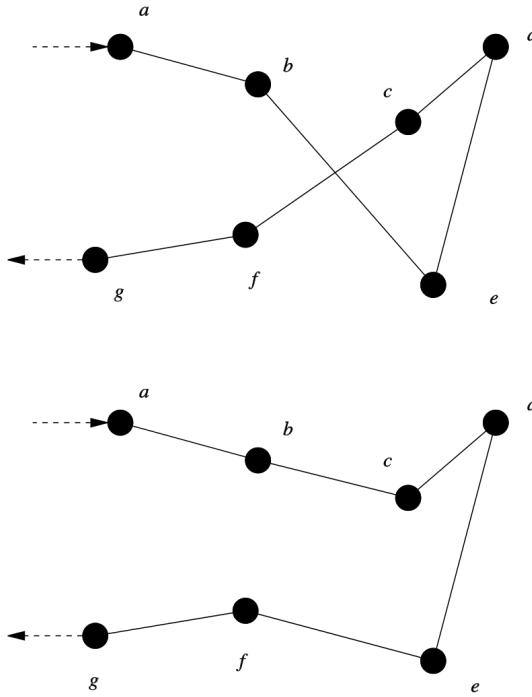
## 21.2 Improvement Heuristics

---

There are many ways to generate improving steps. The key features of improving step to consider are

- What is the complexity of computing this improving step?
- How good this this improving step?

We will mention ways to find neighbors of a current solution for TSP. If the neighbor has a better objective value, the moving to this neighbor will be an improving step.



© wiki/File/2-opt\_wiki.png<sup>1</sup>  
**Figure 21.1:** wiki/File/2-opt\_wiki.png

### 21.2.1. 2-Opt (Subtour Reversal)

---

We will assume that all tours start and end with then node 1.

**2-Opt (Subtour reversal):**

Input a tour  $1 \rightarrow \dots \rightarrow 1$ .

- (a) Pick distinct nodes  $i, j \neq 1$ .
- (b) Let  $s, t$  and  $x_1, \dots, x_k$  be nodes in the tour such that it can be written as

$$1 \rightarrow \dots \rightarrow s \rightarrow i \rightarrow x_1 \rightarrow \dots \rightarrow x_k \rightarrow j \rightarrow t \rightarrow \dots \rightarrow 1.$$

- (c) Consider the subtour reversal

$$1 \rightarrow \dots \rightarrow s \rightarrow j \rightarrow x_k \rightarrow \dots \rightarrow x_1 \rightarrow i \rightarrow t \rightarrow \dots \rightarrow 1.$$

Thus, we reverse the order of  $i \rightarrow x_1 \rightarrow \dots \rightarrow x_k \rightarrow j$ .

- (d) In this process, we

- deleted the edges  $(s, i)$  and  $(j, t)$
- added the edges  $(s, j)$  and  $(i, t)$

Pictorially, this looks like the following

Figure 21.1

Computationally, we need to consider the costs on the edges of a graph.....

See [Englert2014] for an analysis of performance of this improvement.

## 21.2.2. 3-Opt

---

## 21.2.3. *k*-Opt

---

This is a generalization of 2-Opt and 3-Opt.

# 21.3 Meta-Heuristics

---

## 21.3.1. Hill Climbing (2-Opt for TSP)

---

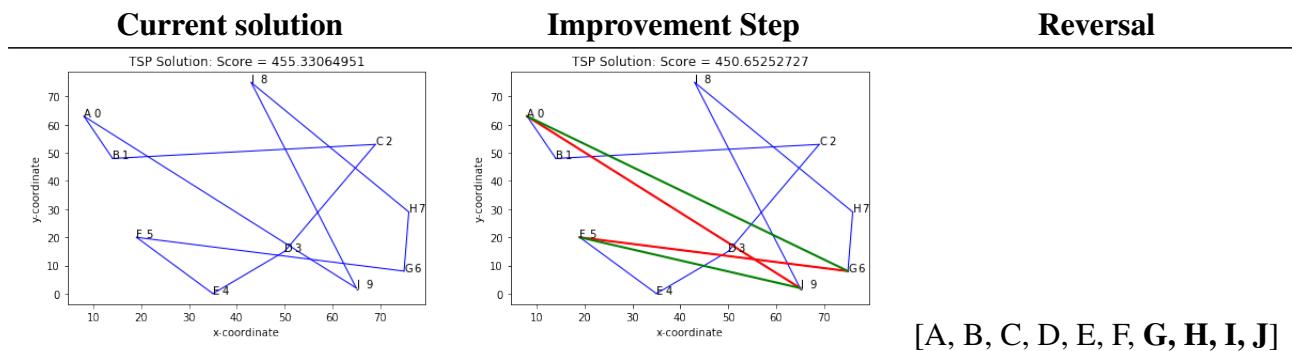
The *Hill Climbing* algorithm finds an improving neighboring solution and climbs in that direction. It continues this process until there is no other neighbor that is improving.

In the context of TSP, we will consider 2-Opt improving moves and the Hill Climbing algorithm for TSP in this case is referred to as the 2-Opt algorithm (also known as the Subtour Reversal Algorithm).

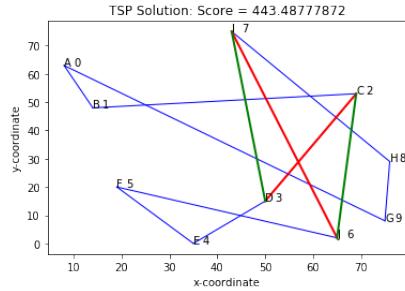
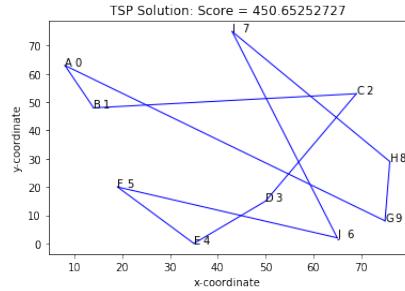
### Hill Climbing:

- Start with an initial feasible solution, label it as the current solution.
- List all neighbors of the current solution.
- If no neighbor has a better solution, then stop.
- Otherwise, move to the best neighbor and go to Step 2.

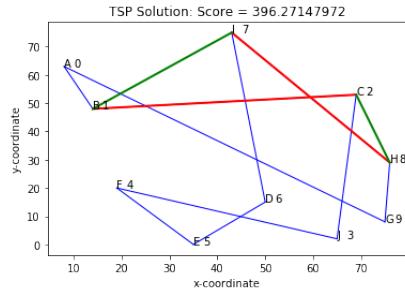
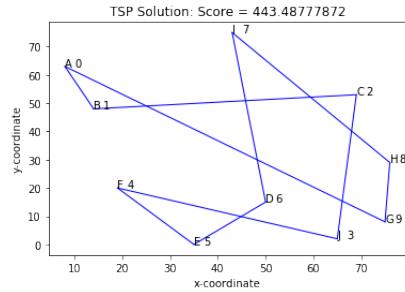
Here is an example on the TSP problem with 2-Opt swaps:



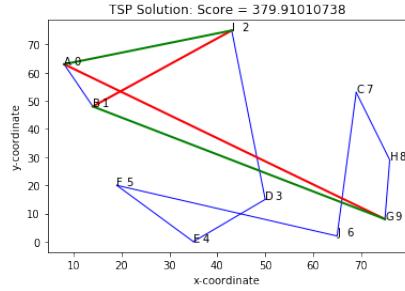
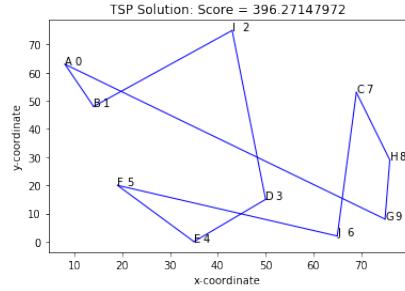
<sup>1</sup>wiki/File/2-opt\_wiki.png, from wiki/File/2-opt\_wiki.png. wiki/File/2-opt\_wiki.png, wiki/File/2-opt\_wiki.png.



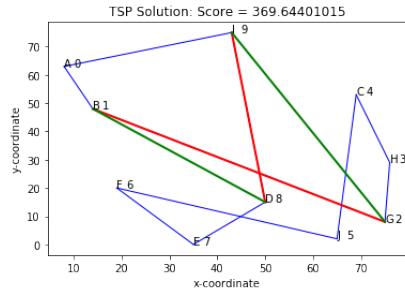
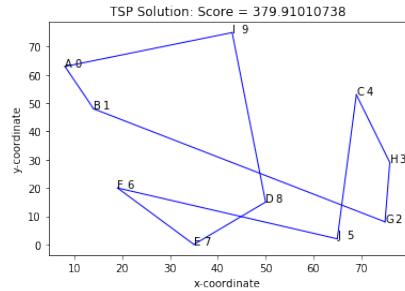
[A, B, C, D, E, F, J, I, H, G]



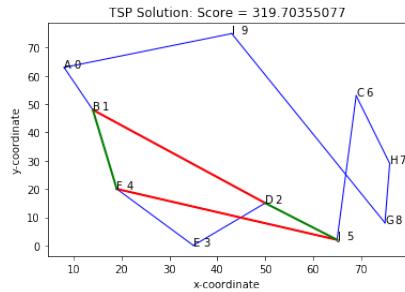
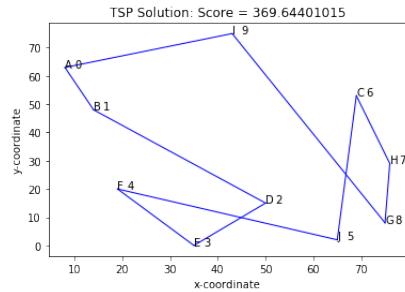
[A, B, C, J, F, E, D, I, H, G]



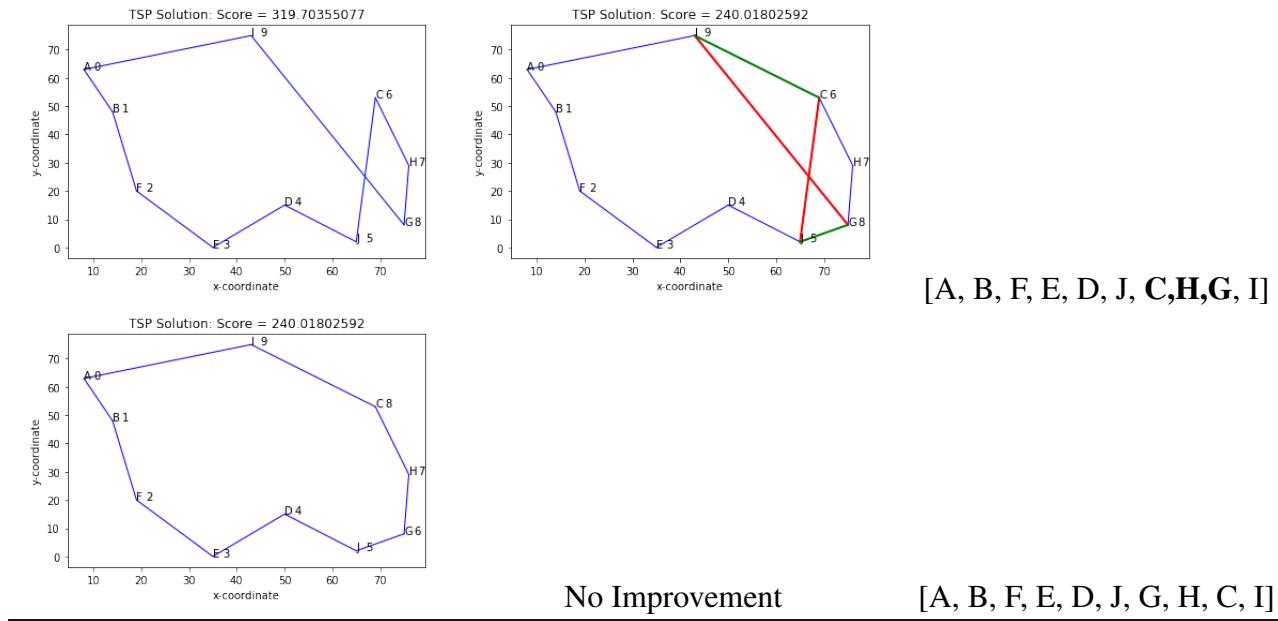
[A, B, I, D, E, F, J, C, H, G]



[A, B, G, H, C, J, F, E, D, I]



[A, B, D, E, F, J, C, H, G, I]



### 21.3.2. Simulated Annealing

Here is a great python package for TSP Simulated annealing: <https://pypi.org/project/satsp/>.

Simulated annealing is a randomized heuristic that randomly decides when to accept non-improving moves. For this, we use what is referred to as a *temperature schedule*. The temperature schedule guides a parameter  $T$  that goes into deciding the probability of accepting a non-improving move.

A typical temperature schedule starts at a value  $T = 0.2 \times Z_c$ , where  $Z_c$  is the objective value of an initial feasible solution. Then the temperature is decreased over time to smaller values.

#### Temperature schedule example:

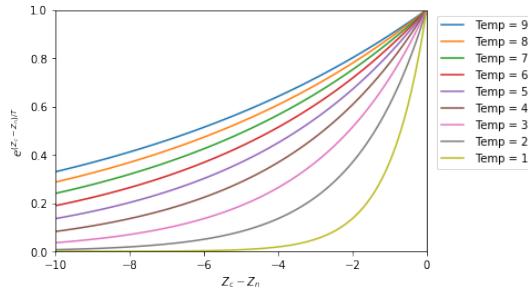
- $T_1 = 0.2Z_c$
- $T_2 = 0.8T_1$
- $T_3 = 0.8T_2$
- $T_4 = 0.8T_3$
- $T_5 = 0.8T_4$

For instance, we could choose to run the algorithm for 10 iterations at each temperature value. The Simulated Annealing algorithm is the following:

#### Simulated Annealing Outline:

[minimization version]

- (a) Start with an initial feasible solution, label it as the current solution, and compute its objective value  $Z_c$ .
- (b) Select a neighbor of the current solution and compute its objective value  $Z_n$ .



© simulated\_annealing\_temperatures<sup>2</sup>

**Figure 21.2: simulated\_annealing\_temperatures**

- (c) Decide whether or not to move to the neighbor:
- If the neighbor is an improvement ( $Z_n < Z_c$ ), **accept the move** and set the neighbor as the current solution.
  - Otherwise,  $Z_c < Z_n$  and thus  $Z_c - Z_n < 0$ . Now with probability  $e^{\frac{Z_c - Z_n}{T}}$  accept the move. In detail:
    - Compute the number  $p = e^{\frac{Z_c - Z_n}{T}}$
    - Generate a random number  $x \in [0, 1]$  from the computer.
    - If  $x < p$ , then **accept the move**.
    - Otherwise, if  $x \geq p$ , **reject the move** and stay at the current solution.
- (d) While still iterations left in the schedule, update the temperature  $T$  and go to Step 2.
- (e) Return the best found solution during the algorithm.

Figure 21.2

### 21.3.3. Tabu Search

---

#### Tabu Search Outline:

[minimization version]

- Initialize a *Tabu List* as an empty set:  $\text{Tabu} = \{\}$ .
- Start with an initial feasible solution, label it as the current solution.
- List all neighbors of the current solution.
- Choose the best neighbor that is not tabu to move too (the move should not be restricted by the set Tabu.)

<sup>2</sup>simulated\_annealing\_temperatures, from simulated\_annealing\_temperatures. simulated\_annealing\_temperatures, simulated\_annealing\_temperatures.

- (e) Add moves to the Tabu List.
- (f) If the Tabu List is longer than its designated maximal size  $S$ , then remove old moves in the list until it reaches the size  $S$ .
- (g) If no object improvement has been seen for  $K$  steps, then Stop.
- (h) Otherwise, Go to Step 3 and continue.

### 21.3.4. Genetic Algorithms

---

Genetic algorithms start with a set of possible solutions, and then slowly mutate them to better solutions. See Scikit-opt for an implementation for the TSP.

[Video explaining a genetic algorithm for TSP](#)

### 21.3.5. Greedy randomized adaptive search procedure (GRASP)

---

We currently do not cover this topic.

[Wikipedia - GRASP](#)

For an in depth (and recent) book, check out Optimization by GRASP Greedy Randomized Adaptive Search Procedures Authors: Resende, Mauricio, Ribeiro, Celso C..

### 21.3.6. Ant Colony Optimization

---

[Wikipedia - Ant Colony Optimization](#)

## 21.4 Computational Comparisons

---

Notice how the heuristics are generally faster and provide reasonable solutions, but the solvers provide the best solutions. This is a trade off to consider when deciding how fast you need a solution and how good of a solution it is that you actually need.

On an instance with 5 nodes:

Nearest Neighbor

494

0.000065 seconds (58 allocations: 2.172 KiB)

Farthest Insertion

494

0.000057 seconds (49 allocations: 1.781 KiB)

Simulated Annealing

494  
0.000600 seconds (7.81 k allocations: 162.156 KiB)

Math Programming Cbc  
494.0  
0.091290 seconds (26.29 k allocations: 1.460 MiB)

Math Programming Gurobi  
Academic license - for non-commercial use only  
494.0  
0.006610 seconds (780 allocations: 78.797 KiB)

One instance on 20 nodes.

Nearest Neighbor  
790  
0.000162 seconds (103 allocations: 6.406 KiB)

Farthest Insertion  
791  
0.000128 seconds (58 allocations: 2.734 KiB)

Simulated Annealing  
777  
0.007818 seconds (130.31 k allocations: 2.601 MiB)

Math Programming Cbc  
773.0  
2.738521 seconds (5.76 k allocations: 607.961 KiB)

Math Programming Gurobi  
Academic license - for non-commercial use only  
773.0  
0.238488 seconds (5.68 k allocations: 717.133 KiB)

Nearest Neighbor  
1216  
0.000288 seconds (142 allocations: 15.141 KiB)

Farthest Insertion  
1281  
0.000286 seconds (60 allocations: 3.969 KiB)

Simulated Annealing

1227

0.047512 seconds (520.51 k allocations: 10.387 MiB, 19.12% gc time)

Math Programming Cbc

1088.0

6.292632 seconds (20.30 k allocations: 2.111 MiB)

Math Programming Gurobi

Academic license - for non-commercial use only

1088.0

1.349253 seconds (20.16 k allocations: 2.520 MiB)

# **Part V**

# **Nonlinear Programming**



## 22. Non-linear Programming (NLP)

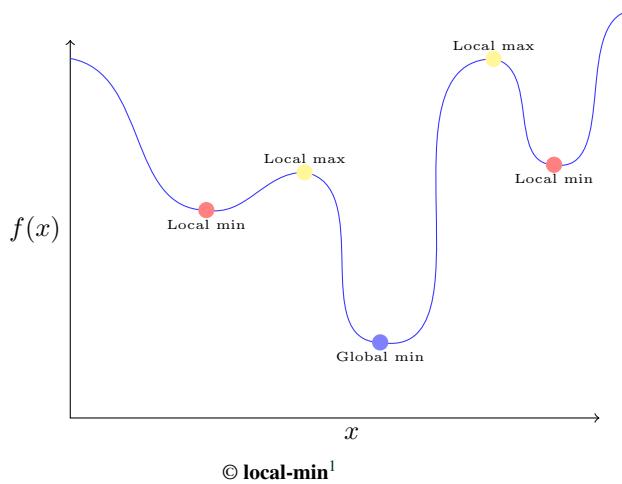
$\min_{x \in \mathbb{R}^n} f(x)$ Unconstrained Minimization	$\min_{x \in \mathbb{R}^n} f(x)$ $f_i(x) \leq 0 \text{ for } i = 1, \dots, m$ Constrained Minimization
--	--

- **objective function**  $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- may **maximize**  $f$  by minimizing the function  $g(x) := -f(x)$

### Definition 22.1

The vector  $x^*$  is a

- global minimizer if  $f(x^*) \leq f(x)$  for all  $x \in \mathbb{R}^n$ .
- local minimizer if  $f(x^*) \leq f(x)$  for all  $x$  satisfying  $\|x - x^*\| \leq \varepsilon$  for some  $\varepsilon > 0$ .
- strict local minimizer if  $f(x^*) < f(x)$  for all  $x \neq x^*$  satisfying  $\|x - x^*\| \leq \varepsilon$  for some  $\varepsilon > 0$ .



### Theorem 22.2: Attaining a minimum

Let  $S$  be a nonempty set that is closed and bounded. Suppose that  $f: S \rightarrow \mathbb{R}$  is continuous. Then the problem  $\min\{f(x) : x \in S\}$  attains its minimum.

### Definition 22.3: Critical Point

A critical point is a point  $\bar{x}$  where  $\nabla f(\bar{x}) = 0$ .

<sup>1</sup>local-min, from local-min. local-min, local-min.

**Theorem 22.4**

Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is differentiable. If  $\min\{f(x) : x \in \mathbb{R}^n\}$  has an optimizer  $x^*$ , then  $x^*$  is a critical point of  $f$  (i.e.,  $\nabla f(x^*) = 0$ ).

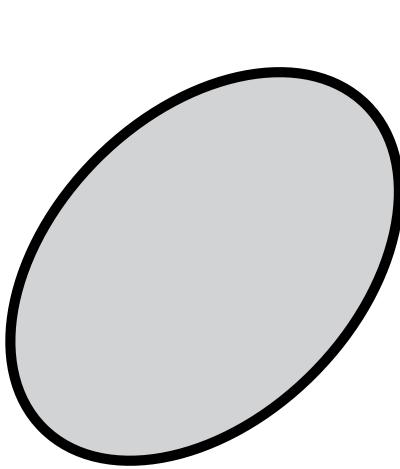
## 22.1 Convex Sets

**Definition 22.5: Convex Combination**

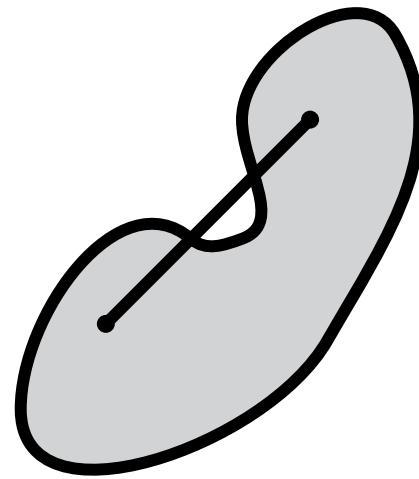
Given two points  $x, y$ , a convex combination is any point  $z$  that lies on the line between  $x$  and  $y$ . Algebraically, a convex combination is any point  $z$  that can be represented as  $z = \lambda x + (1 - \lambda)y$  for some multiplier  $\lambda \in [0, 1]$ .

**Definition 22.6: Convex Set**

A set  $C$  is convex if it contains all convex combinations of points in  $C$ . That is, for any  $x, y \in C$ , it holds that  $\lambda x + (1 - \lambda)y \in C$  for all  $\lambda \in [0, 1]$ .



(a) Convex Set



(b) Non-convex set

**Figure 22.1: Examples of convex and non-convex sets.****Definition 22.7: Convex Sets**

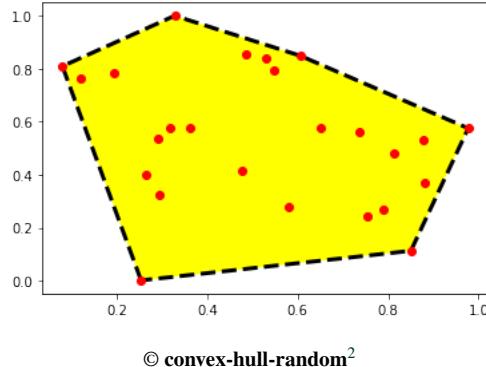
A set  $S$  is convex if for any two points in  $S$ , the entire line segment between them is also contained in  $S$ . That is, for any  $x, y \in S$

$$\lambda x + (1 - \lambda)y \in S \quad \text{for all } \lambda \in [0, 1].$$

- (a) Hyperplane  $H = \{x \in \mathbb{R}^n : a^\top x = b\}$
- (b) Halfspace  $H = \{x \in \mathbb{R}^n : a^\top x \leq b\}$
- (c) Polyhedron  $P = \{x \in \mathbb{R}^n : Ax \leq b\}$
- (d) Sphere  $S = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i^2 \leq 1\}$
- (e) Second Order Cone  $S = \{(x, t) \in \mathbb{R}^n \times \mathbb{R} : \sum_{i=1}^n x_i^2 \leq t^2\}$

### Definition 22.8: Convex Hull

Let  $S \subseteq \mathbb{R}^n$ . The convex hull  $\text{conv}(S)$  is the smallest convex set containing  $S$ .

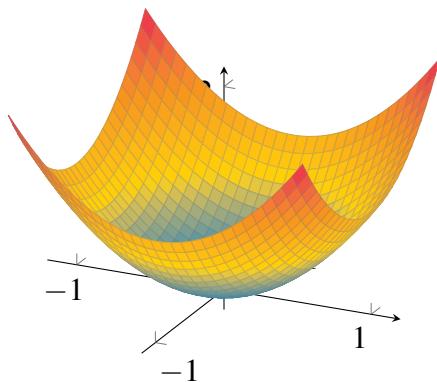


### Theorem 22.9: Caratheodory's Theorem

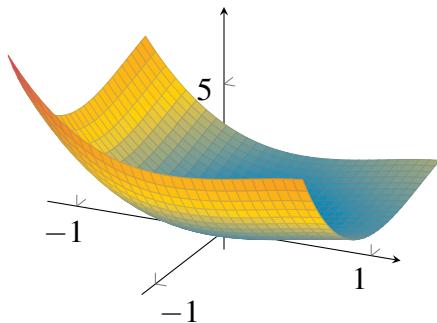
Let  $x \in \text{conv}(S)$  and  $S \subseteq \mathbb{R}^n$ . Then there exist  $x^1, \dots, x^k \in S$  such that  $x \in \text{conv}(\{x^1, \dots, x^k\})$  and  $k \leq n + 1$ .

## 22.2 Convex Functions

Convex function are "nice" functions that "open up". They represent an extremely important class of functions in optimization and typically can be optimized over efficiently.



**Figure 22.2: Convex Function  $f(x,y) = x^2 + y^2$ .**



**Figure 22.3: Non-Convex Function  $f(x,y) = x^2 + y^2 - (x - 0.3)^2 - (y - 0.4)^2$ .**

#### Definition 22.10: Convex Functions

A function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is convex if for all  $x, y \in \mathbb{R}^n$  and  $\lambda \in [0, 1]$  we have

$$\lambda f(x) + (1 - \lambda)f(y) \geq f(\lambda x + (1 - \lambda)y). \quad (22.1)$$

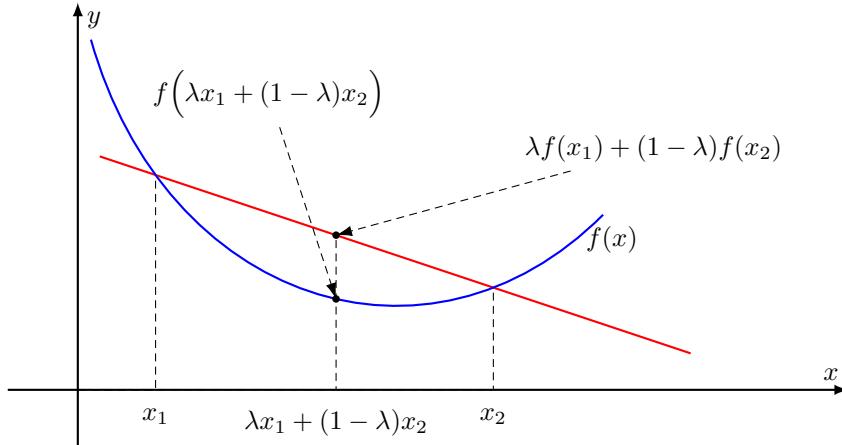
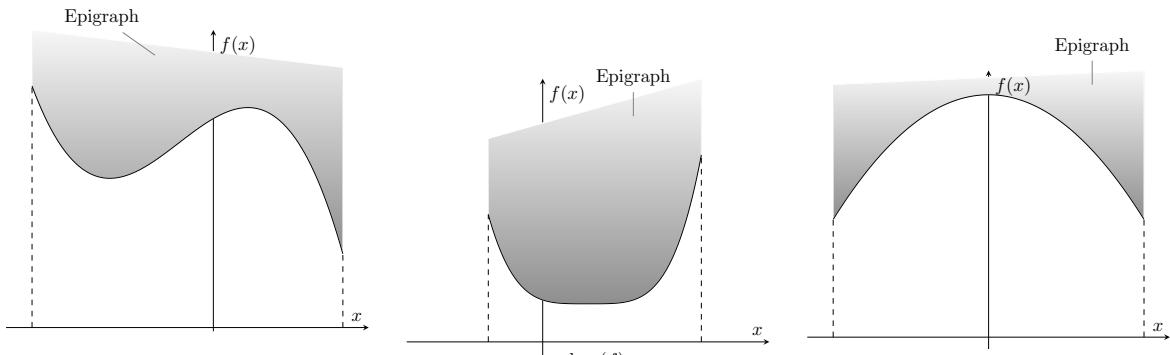
An equivalent definition of convex function are through the epigraph.

#### Definition 22.11: Epigraph

The epigraph of  $f$  is the set  $\{(x, y) : y \geq f(x)\}$ . This is the set of all points "above" the function.

<sup>3</sup>tikz/convexity-definition.pdf, from tikz/convexity-definition.pdf. [tikz/convexity-definition.pdf](https://tex.stackexchange.com/questions/394923/how-one-can-draw-a-convex-function), [tikz/convexity-definition.pdf](https://tex.stackexchange.com/questions/394923/how-one-can-draw-a-convex-function).

<sup>3</sup><https://tex.stackexchange.com/questions/394923/how-one-can-draw-a-convex-function>

© tikz/convexity-definition.pdf<sup>3</sup>**Figure 22.4: Illustration explaining the definition of a convex function.****Theorem 22.12** $f(x)$  is a convex function if and only if the epigraph of  $f$  is a convex set.© epigraph.pdf<sup>4</sup>

5

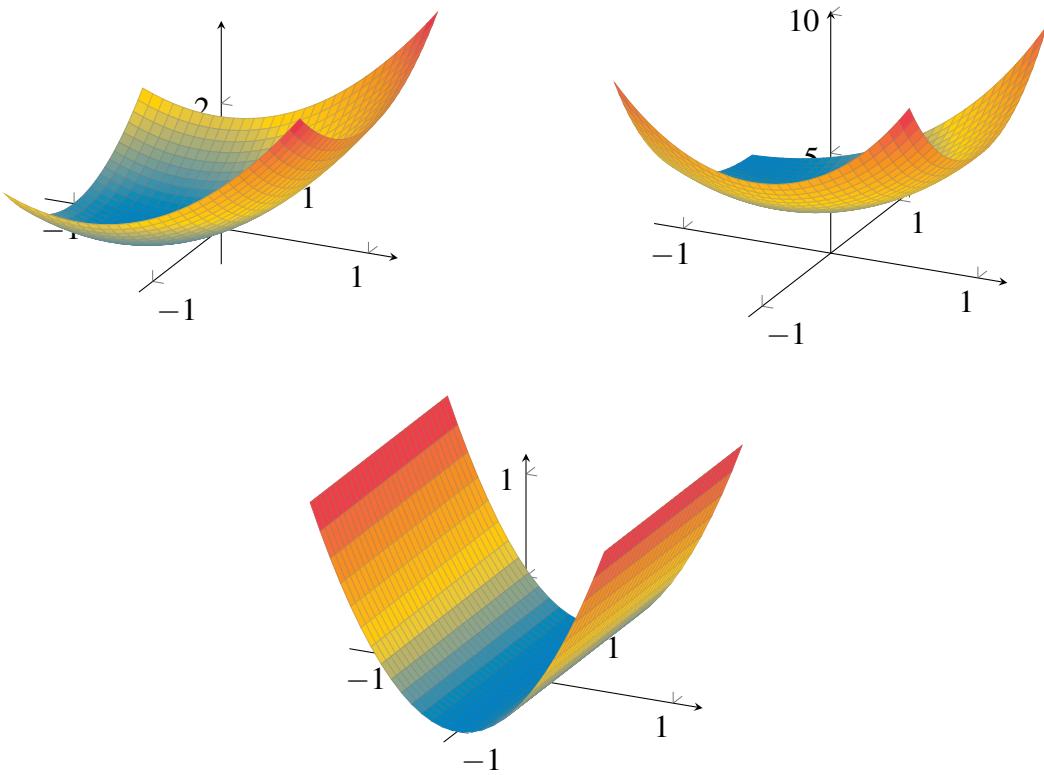
**Example 22.13: Examples of Convex functions**

Some examples are

- $f(x) = ax + b$
- $f(x) = x^2$

<sup>4</sup>epigraph.pdf, from epigraph.pdf, epigraph.pdf, epigraph.pdf.<sup>5</sup><https://tex.stackexchange.com/questions/261501/function-epigraph-possibly-using-fillbetween>

- $f(x) = x^4$
- $f(x) = |x|$
- $f(x) = e^x$
- $f(x) = -\sqrt{x}$  on the domain  $[0, \infty)$ .
- $f(x) = x^3$  on the domain  $[0, \infty)$ .
- $f(x, y) = \sqrt{x^2 + y^2}$
- $f(x, y) = x^2 + y^2 + x$
- $f(x, y) = e^{x+y}$
- $f(x, y) = e^x + e^y + x^2 + (3x + 4y)^6$



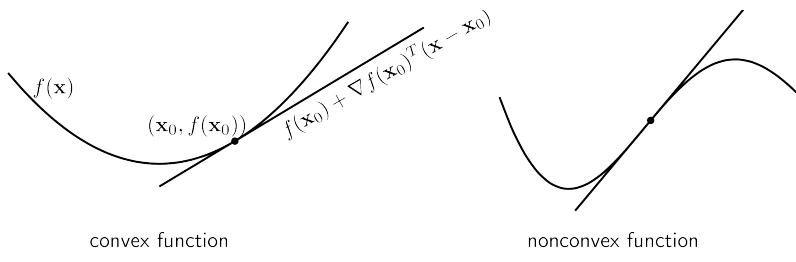
**Figure 22.5: Convex Functions**  $f(x, y) = x^2 + y^2 + x$ ,  $f(x, y) = e^{x+y} + e^{x-y} + e^{-x-y}$ , and  $f(x, y) = x^2$ .

### 22.2.1. Proving Convexity - Characterizations

#### Theorem 22.14: Convexity: First order characterization - linear underestimates

Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is differentiable. Then  $f$  is convex if and only if for all  $\bar{x} \in \mathbb{R}^n$ , the linear tangent is an underestimator to the function, that is,

$$f(\bar{x}) + (x - \bar{x})^\top \nabla f(\bar{x}) \leq f(x).$$

© first-order-convexity<sup>6</sup>

7

**Theorem 22.15: Convexity: Second order characterization - positive curvature**

We give statements for uni-variate functions and multi-variate functions.

- Suppose  $f: \mathbb{R} \rightarrow \mathbb{R}$  is twice differentiable. Then  $f$  is convex if and only if  $f''(x) \geq 0$  for all  $x \in \mathbb{R}$ .
- Suppose  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is twice differentiable. Then  $f$  is convex if and only if  $\nabla^2 f(x) \succcurlyeq 0$  for all  $x \in \mathbb{R}^n$ .

**22.2.2. Proving Convexity - Composition Tricks****Positive Scaling of Convex Function is Convex:**

If  $f$  is convex and  $\alpha > 0$ , then  $\alpha f$  is convex.

**Example:**  $f(x) = e^x$  is convex. Therefore,  $25e^x$  is also convex.

**Sum of Convex Functions is Convex:**

If  $f$  and  $g$  are both convex, then  $f + g$  is also convex.

**Example:**  $f(x) = e^x, g(x) = x^4$  are convex. Therefore,  $e^x + x^4$  is also convex.

**Composition with affine function:**

If  $f(x)$  is convex, then  $f(a^\top x + b)$  is also convex.

**Example:**  $f(x) = x^4$  are convex. Therefore,  $(3x + 5y + 10z)^4$  is also convex.

**Pointwise maximum:**

<sup>6</sup><https://machinelearningcoban.com/2017/03/12/convexity/>

If  $f_i$  are convex for  $i = 1, \dots, t$ , then  $f(x) = \max_{i=1, \dots, t} f_i(x)$  is convex.

**Example:**  $f_1(x) = e^{-x}$ ,  $f_2(x) = e^x$  are convex. Therefore,  $f(x) = \max(e^x, e^{-x})$  is also convex.

### Other compositions:

Suppose

$$f(x) = h(g(x)).$$

- (a) If  $g$  is convex,  $h$  is convex and **non-decreasing**, then  $f$  is convex.
- (b) If  $g$  is concave,  $h$  is convex and **non-increasing**, then  $f$  is convex.

**Example 1:**  $g(x) = x^4$  is convex,  $h(x) = e^x$  is convex and non-decreasing. Therefore,  $f(x) = e^{x^4}$  is also convex.

**Example 2:**  $g(x) = \sqrt{x}$  is concave (on  $[0, \infty)$ ),  $h(x) = e^{-x}$  is convex and non-increasing. Therefore,  $f(x) = e^{-\sqrt{x}}$  is convex on  $x \in [0, \infty)$ .

## 22.3 Convex Optimization Examples

---

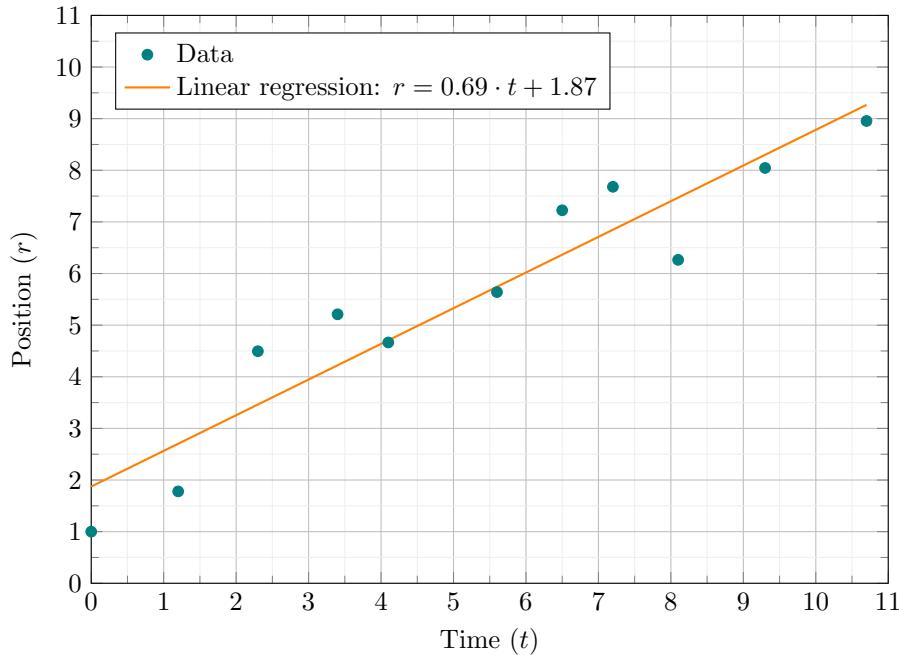
### 22.3.1. Unconstrained Optimization: Linear Regression

---

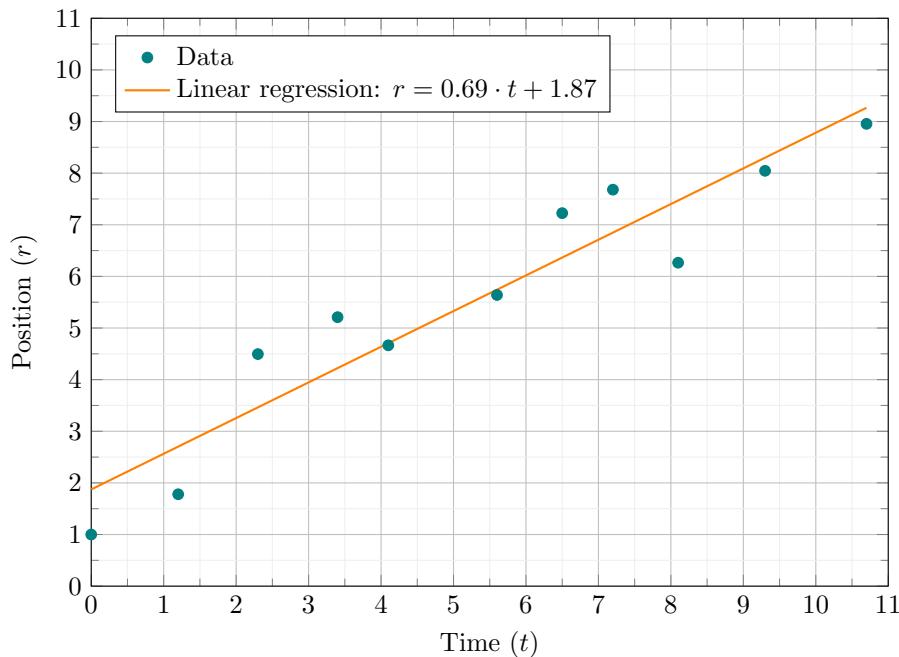
Given data points  $x^1, \dots, x^N \in \mathbb{R}^d$  and out values  $y^i \in \mathbb{R}$ , we want to find a linear function  $y = \beta \cdot x$  that best approximates  $x^i \cdot \beta \approx y^i$ . For example, the data could  $x = (\text{time})$  and the output could be  $y = \text{position}$ .

---

<sup>8</sup>[tikz/linear-regression.pdf](#), from [tikz/linear-regression.pdf](#). [tikz/linear-regression.pdf](#), [tikz/linear-regression.pdf](#).



© tikz/linear-regression.pdf<sup>8</sup>  
**Figure 22.6: Line derived through linear regression.**



© LinearRegression.pdf<sup>9</sup>  
10

As is standard, we choose the error (or "loss") from each data point as the squared error. Hence, we

<sup>10</sup><https://latexdraw.com/linear-regression-in-latex-using-tikz/>

can model this as the optimization problem:

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^N (x^i \cdot \beta - y^i)^2 \quad (22.1)$$

This problem has a nice closed form solution. We will derive this solution, and then in a later section discuss why using this solution might be too slow to compute on large data sets. In particular, the solution comes as a system of linear equations. But when  $N$  is really large, we may not have time to solve this system, so an alternative is to use decent methods, discussed later in this chapter.

### Theorem 22.16: Linear Regression Solution

The solution to (22.1) is

$$\beta = (X^\top X)^{-1} X^\top Y, \quad (22.2)$$

where

$$X = \begin{bmatrix} x^1 \\ \vdots \\ x^N \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_d^1 \\ \vdots & \vdots & & \vdots \\ x_1^N & x_2^N & \dots & x_d^N \end{bmatrix}$$

**Proof.** Solve for  $\nabla f(\beta) = 0$ .

To be completed....



### Resources

<https://www.youtube.com/watch?v=E5RjzSK0fvY>

## 22.4 Machine Learning - SVM

*Support Vector Machine* (SVM) is a tool used in machine learning for classifying data points. For instance, if there are red and black data points, how can we find a good line that separates them? The input data that you are given is as follows:

INPUT:

- $d$ -dimensional data points  $x^1, \dots, x^N$
- 1-dimensional labels  $z^1, \dots, z^N$  (typically we will use  $z_i$  is either 1 or  $-1$ )

The output to the problem should be a hyperplane  $w^\top x + b = 0$  that separates the two data types (either exact separation or approximate separation).

OUTPUT:

- A  $d$ -dimensional vector  $w$
- A 1-dimensional value  $b$

Given this output, we can construct a classification function  $f(x)$  as

$$f(x) = \begin{cases} 1 & \text{if } w^\top x + b \geq 0, \\ -1 & \text{if } w^\top x + b < 0. \end{cases} \quad (22.1)$$

There are three versions to consider:

#### 22.4.0.1. Feasible separation

---

If we only want to a line that separates the data points, we can use the following optimization model.

$$\begin{aligned} \min \quad & 0 \\ \text{such that} \quad & z^i(w^\top x^i + b) \geq 1 \quad \text{for all } i = 1, \dots, N \\ & w \in \mathbb{R}^d \\ & b \in \mathbb{R} \end{aligned}$$

#### 22.4.0.2. Distance between hyperplanes

---

We want to know a formula for the distance between two parallel hyperplanes. In particular, we will look at the hyperplane  $H_0 = \{x : h^\top x = 0\}$  and  $H_1 = \{x : h^\top x = 1\}$ .

We choose a point  $x^0 = 0 \in H_0$ , and then find the point in  $H_1$  that minimizes the distance between these two points.

This can be phrased as the optimization problem

$$\begin{aligned} \min \quad & \|x\|_2 \\ \text{s.t.} \quad & h^\top x = 1 \end{aligned}$$

For convenience, we will solve this problem instead:

$$\begin{aligned} \min \quad & \|x\|_2^2 \\ \text{s.t.} \quad & h^\top x = 1 \end{aligned}$$

We will rewrite this problem by removing the equation. We assume, without loss of generality, that  $h_n \neq 0$ . Otherwise, reorder the variables.

So, since  $h_n \neq 0$ , we have

$$1 = h_1x_1 + \cdots + h_nx_n$$

and hence

$$x_n = \frac{1 - h_1x_1 - \cdots - h_{n-1}x_{n-1}}{h_n}.$$

Now we can re-write the optimization problem as the unconstrained optimization problem

$$\min f(x) := x_1^2 + \cdots + x_{n-1}^2 + \left( \frac{1 - h_1x_1 - \cdots - h_{n-1}x_{n-1}}{h_n} \right)^2$$

First, note that  $f(x)$  is a strictly convex function. Thus, we can find the optimal solution by computing where the gradient vanishes.

$$\nabla f(x) = \begin{bmatrix} \vdots \\ 2x_i + 2\frac{h_i}{h_n} \left( \frac{1 - h_1x_1 - \cdots - h_{n-1}x_{n-1}}{h_n} \right) \\ \vdots \end{bmatrix} = 0$$

But notice what happens when we substitute back in  $x_n$ . We obtain

$$\nabla f(x) = \begin{bmatrix} \vdots \\ 2x_i + 2\frac{h_i}{h_n}x_n \\ \vdots \end{bmatrix} = 0$$

Taking the  $i$ th equation, we have

$$\frac{x_i}{h_i} = \frac{x_n}{h_n} \quad \text{for all } i = 1, \dots, n-1.$$

Let  $\lambda = \frac{x_n}{h_n}$ . Then for all  $i = 1, \dots, n$ , we have  $x_i = \lambda h_i$ , or in vector form, we have

$$x = \lambda h.$$

Thus, we can go back the original optimization problem and look only at solutions that are of the form  $x = \lambda h$ .

Plugging this into the original model, we have

$$\begin{aligned} & \min \|\lambda h\|_2 \\ & \text{s.t. } h^\top(\lambda h) = 1 \end{aligned}$$

which is equivalent to

$$\begin{aligned} & \min \lambda \|h\|_2 \\ \text{s.t. } & \lambda = \frac{1}{h^\top h}. \end{aligned}$$

Recall that  $h^\top h = \|h\|_2^2$ .

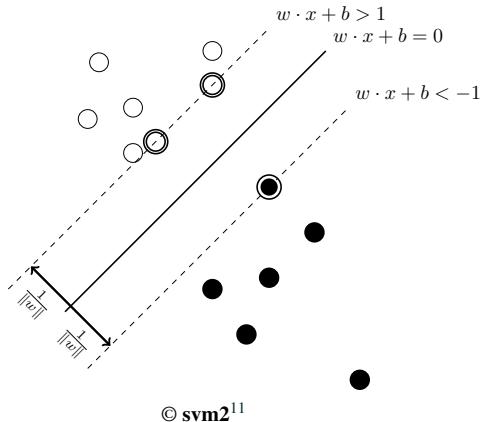
Thus, the minimum distance is exactly

$$\frac{1}{\|h\|_2^2} \|h\|_2 = \frac{1}{\|h\|_2}$$

### 22.4.0.3. SVM

---

We can modify the objective function to find a best separation between the points. This can be done in the following way



$$\begin{aligned} & \min \|w\|_2^2 \\ \text{such that } & z^i(w^\top x^i + b) \geq 1 \quad \text{for all } i = 1, \dots, N \\ & w \in \mathbb{R}^d \\ & b \in \mathbb{R} \end{aligned}$$

Here,  $\|w\|_2^2 = \sum_{i=1}^d w_i^2 = w_1^2 + \dots + w_d^2$ .

---

<sup>11</sup>svm2, from svm2. svm2, svm2.

#### 22.4.0.4. Approximate SVM

---

We can modify the objective function and the constraints to allow for approximate separation. This would be the case when you want to ignore outliers that don't fit well with the data set, or when exact SVM is not possible. This is done by changing the constraints to be

$$z^i(w^\top x^i + b) \geq 1 - \delta_i$$

where  $\delta_i \geq 0$  is the error in the constraint for datapoint  $i$ . In order to reduce these errors, we add a penalty term in the objective function that encourages these errors to be small. For this, we can pick some number  $C$  and write the objective as

$$\min \|w\|_2^2 + C \sum_{i=1}^N \delta_i.$$

This creates the following optimization problem

$$\begin{aligned} \min \quad & \|w\|_2^2 + C \sum_{i=1}^N \delta_i \\ \text{such that} \quad & z^i(w^\top x^i + b) \geq 1 - \delta_i && \text{for all } i = 1, \dots, N \\ & w \in \mathbb{R}^d \\ & b \in \mathbb{R} \\ & \delta_i \geq 0 \text{ for all } i = 1, \dots, N \end{aligned}$$

See information about the scikit-learn module for svm here: <https://scikit-learn.org/stable/modules/svm.html>.

#### 22.4.1. SVM with non-linear separators

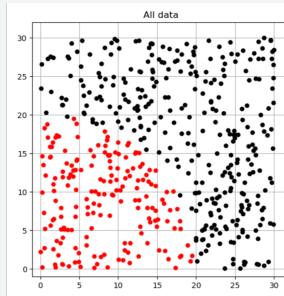
---

##### Resources

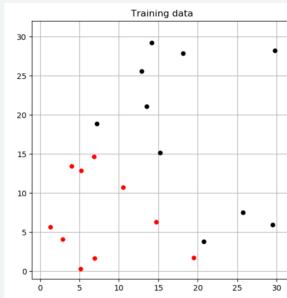
[https://www.youtube.com/watch?time\\_continue=6&v=N1v0golbjSc](https://www.youtube.com/watch?time_continue=6&v=N1v0golbjSc)

Suppose for instance you are given data  $x^1, \dots, x^N \in \mathbb{R}^2$  (2-dimensional data) and given labels are dependent on the distance from the origin, that is, all data points  $x$  with  $x_1^2 + x_2^2 > r$  are given a label  $+1$  and all data points with  $x_1^2 + x_2^2 \leq r$  are given a label  $-1$ . That is, we want to learn the function

$$f(x) = \begin{cases} 1 & \text{if } x_1^2 + x_2^2 > r, \\ -1 & \text{if } x_1^2 + x_2^2 \leq r. \end{cases} \quad (22.2)$$

**Example 22.17**

© svm-nonlinear<sup>12</sup>



© svm-nonlinear-training<sup>13</sup>

Here we have a *classification problem where the data cannot be separated by a hyperplane*. On the left, we have all of the data given to use. On the right, we have a subset of the data that we could try using for training and then test our learned function on the remaining data. As we saw in class, this amount of data was not sufficient to properly classify most of the data.

---

**svm-nonlinear**, from **svm-nonlinear**. **svm-nonlinear**, **svm-nonlinear**.

**svm-nonlinear-training**, from **svm-nonlinear-training**.

**svm-nonlinear-training**,

**svm-nonlinear-training**.

We cannot learn this classifier from the data directly using the hyperplane separation with SVM in the last section. But, if we modify the data set, then we can do this.

For each data point  $x$ , we transform it into a data point  $X$  by adding a third coordinate equal to  $x_1^2 + x_2^2$ . That is

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow X = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{pmatrix}. \quad (22.3)$$

In this way, we convert the data  $x^1, \dots, x^N$  into data  $X^1, \dots, X^N$  that lives in a higher-dimensional space. But with this new dataset, we can apply the hyperplane separation technique in the last section to properly classify the data.

This can be done with other nonlinear classification functions.

## 22.4.2. Support Vector Machines

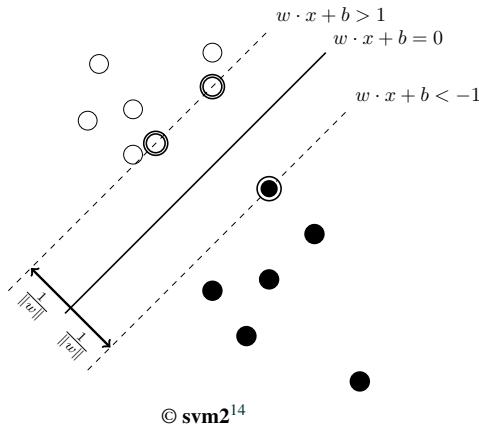
### Support Vector Machine - Exact Classification:

Given labeled data  $(x^i, y_i)$  for  $i = 1, \dots, N$ , where  $x^i \in \mathbb{R}^d$  and  $y^i \in \{-1, 1\}$ , find a vector  $w \in \mathbb{R}^d$  and a number  $b \in \mathbb{R}$  such that

$$x^i \cdot w + b > 0 \quad \text{if } y^i = 1 \quad (22.4)$$

$$x^i \cdot w + b < 0 \quad \text{if } y^i = -1 \quad (22.5)$$

There may exist many solutions to this problem. Thus, we are interested in the "best" solution. Such a solution will maximize the separation between the two sets of points. To consider an equal margin on either side, we set the right hand sides to 1 and -1 and then compute the margin from the hyperplane. Notice that it is sufficient to use 1 and -1 on the right hand sides since any scaling can happen in  $w$  and  $b$ .



We will show that the margin under this model can be computed as  $\frac{2}{\|w\|}$  where  $\|w\| = \sqrt{w_1^2 + \dots + w_d^2}$ . Hence, maximizing the margin is equivalent to minimizing  $w_1^2 + \dots + w_d^2$ . We arrive at the model

$$\min \sum_{i=1}^d w_i^2 \quad (22.6)$$

$$x^i \cdot w + b \geq 1 \quad \text{if } y^i = 1 \quad (22.7)$$

$$x^i \cdot w + b \leq -1 \quad \text{if } y^i = -1 \quad (22.8)$$

<sup>14</sup>svm2, from **svm2**. **svm2**, **svm2**.

Or even more compactly written as

$$\min \sum_{i=1}^d w_i^2 \quad (22.9)$$

$$y^i(x^i \cdot w + b) \geq 1 \quad \text{for } i = 1, \dots, N \quad (22.10)$$

## 22.5 Classification

---

### 22.5.1. Machine Learning

---

#### Resources

[https://www.youtube.com/watch?v=bwZ3Qiuj3i8&list=PL9ooVrP1hQOHUfd-g8GUpKI3hH0wM\\_9Dn&index=13](https://www.youtube.com/watch?v=bwZ3Qiuj3i8&list=PL9ooVrP1hQOHUfd-g8GUpKI3hH0wM_9Dn&index=13)

<https://towardsdatascience.com/solving-a-simple-classification-problem-with-python>

### 22.5.2. Neural Networks

---

#### Resources

<https://www.youtube.com/watch?v=bVQUSndD11U>

<https://www.youtube.com/watch?v=8bNIkfRJZpo>

<https://www.youtube.com/watch?v=Dws9Zveu9ug>

## 22.6 Box Volume Optimization in Scipy.Minimize

---

<https://www.youtube.com/watch?v=iSnTtV6b0Gw>

## 22.7 Modeling

---

We will discuss a few models and mention important changes to the models that will make them solvable.

## IMPORTANT TIPS

- (a) **Find a convex formulation.** It may be that the most obvious model for your problem is actually non-convex. Try to reformulate your model into one that is convex and hence easier for solvers to handle.
- (b) **Intelligent formulation.** Understanding the problem structure may help reduce the complexity of the problem. Try to deduce something about the solution to the problem that might make the problem easier to solve. This may work for special cases of the problem.
- (c) **Identify problem type and select solver.** Based on your formulation, identify which type of problem it is and which solver is best to use for that type of problem. For instance, Gurobi can handle some convex quadratic problems, but not all. Ipopt is a more general solver, but may be slower due to the types of algorithms that it uses.
- (d) **Add bounds on the variables.** Many solvers perform much better if they are provided bounds to the variables. This is because it reduces the search region where the variables live. Adding good bounds could be the difference in the solver finding an optimal solution and not finding any solution at all.
- (e) **Warm start.** If you know good possible solutions to the problem (or even just a feasible solution), you can help the solver by telling it this solution. This will reduce the amount of work the solver needs to do. In JUMP this can be done by using the command `setvalue(x,[2 4 6])`, where here it sets the value of vector  $x$  to [2 4 6]. It may be necessary to specify values for all variables in the problem for it to start at.
- (f) **Rescaling variables.** It sometimes is useful to have all variables on the same rough scale. For instance, if minimizing  $x^2 + 100^2y^2$ , it may be useful to define a new variable  $\bar{y} = 100y$  and instead minimize  $x^2 + \bar{y}^2$ .
- (g) **Provide derivatives.** Working out gradient and hessian information by hand can save the solver time. Particularly when these are sparse (many zeros). These can often be provided directly to the solver.

See <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=4982C26EC5F25564BCC239FD3785E2D3?doi=10.1.1.210.3547&rep=rep1&type=pdf> for many other helpful tips on using Ipopt.

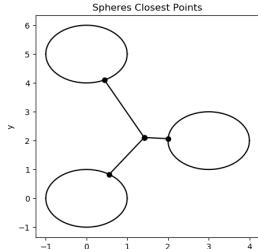
### 22.7.1. Minimum distance to circles

---

The problem we will consider here is: Given  $n$  circles, find a center point that minimizes the sum of the distances to all of the circles.

**Minimize distance to circles:**

Given circles described by center points  $(a_i, b_i)$  and radius  $r_i$  for  $i = 1, \dots, n$ , find a point  $c = (c_x, c_y)$  that minimizes the sum of the distances to the circles.



© circles-figure<sup>15</sup>

### Minimize distance to circles - Model attempt #1:

Non-convex

Let  $(x_i, y_i)$  be a point in circle  $i$ . Let  $w_i$  be the distance from  $(x_i, y_i)$  to  $c$ . Then we can model the problem as follows:

$$\begin{array}{ll} \min & \sum_{i=1}^3 w_i & \text{Sum of distances} \\ \text{s.t.} & \sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} = r, \quad i = 1, \dots, n & (x_i, y_i) \text{ is in circle } i \\ & \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} = w_i, \quad i = 1, \dots, n & w_i \text{ is distance from } (x_i, y_i) \text{ to } c \end{array} \quad (22.1)$$

### This model has several issues:

- (a) If the center  $c$  lies inside one of the circles, then the constraint  $\sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} = r$  may not be valid. This is because the optimal choice for  $(x_i, y_i)$  in this case would be inside the circle, that is, satisfying  $\sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} \leq r$ .
- (b) This model is **nonconvex**. In particular the equality constraints make the problem nonconvex. Fortunately, we can relax the problem to make it convex and still model the correct solution. In particular, consider the constraint

$$\sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \leq w_i.$$

Since we are minimizing  $\sum w_i$ , it is equivalent to have the constraint

$$\sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \leq w_i.$$

This is equivalent because any optimal solution makes  $w_i$  the smallest it can, and hence will meet that constraint at equality.

What is great about this change, is that it makes the constraint **convex!**. To see this we can write  $f(z) = \|z\|_2^2$ ,  $z = (x_i - c_x, y_i - c_y)$ . Since  $f(z)$  is convex and the transformation into variables  $x_i, c_x, y_i, c_y$  is linear, we have that  $f(x_i - c_x, y_i - c_y)$  is convex. Then since  $-w_i$  is linear, we have that

$$f(x_i - c_x, y_i - c_y) - w_i$$

is a convex function. Thus, the constraint

$$f(x_i - c_x, y_i - c_y) - w_i \leq 0$$

<sup>15</sup>circles-figure, from circles-figure. circles-figure, circles-figure.

is a convex constraint.

This brings us to our second model.

### Minimize distance to circles - Model attempt #2:

Convex

Let  $(x_i, y_i)$  be a point in circle  $i$ . Let  $w_i$  be the distance from  $(x_i, y_i)$  to  $c$ . Then we can model the problem as follows:

$$\begin{array}{ll} \min & \sum_{i=1}^3 w_i & \text{Sum of distances} \\ \text{s.t.} & \sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} \leq r, \quad i = 1, \dots, n & (x_i, y_i) \text{ is in circle } i \\ & \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \leq w_i, \quad i = 1, \dots, n & w_i \text{ is distance from } (x_i, y_i) \text{ to } c \end{array} \quad (22.2)$$

Lastly, we would like to make this model better for a solver. For this we will

- (a) Add bounds on all the variables
- (b) Change format of non-linear inequalities

### Minimize distance to circles - Model attempt #3:

Convex

Let  $(x_i, y_i)$  be a point in circle  $i$ . Let  $w_i$  be the distance from  $(x_i, y_i)$  to  $c$ . Then we can model the problem as follows:

$$\begin{array}{ll} \min & \sum_{i=1}^3 w_i & \text{Sum of distances} \\ \text{s.t.} & (x_i - a_i)^2 + (y_i - b_i)^2 \leq r^2, \quad i = 1, \dots, n & (x_i, y_i) \text{ is in circle } i \\ & (x_i - c_x)^2 + (y_i - c_y)^2 \leq w_i^2, \quad i = 1, \dots, n & w_i \text{ is distance from } (x_i, y_i) \text{ to } c \\ & 0 \leq w_i \leq u_i \\ & a_i - r \leq x_i \leq a_i + r \\ & b_i - r \leq y_i \leq b_i + r \end{array} \quad (22.3)$$

### Example: Minimize distance to circles

Code

Here we minimize the distance of three circles of radius 1 centered at  $(0,0)$ ,  $(3,2)$ , and  $(0,5)$ .

Note: The bounds on the variables here are not chosen optimally.

$$\begin{aligned}
 \min \quad & w_1 + w_2 + w_3 \\
 \text{Subject to} \quad & (x_1 - 0)^2 + (y_1 - 0)^2 \leq 1 \\
 & (x_2 - 3)^2 + (y_2 - 2)^2 \leq 1 \\
 & (x_3 - 0)^2 + (y_3 - 5)^2 \leq 1 \\
 & (x_1 - c_x)^2 + (y_1 - c_y)^2 \leq w_1^2 \\
 & (x_2 - c_x)^2 + (y_2 - c_y)^2 \leq w_2^2 \\
 & (x_3 - c_x)^2 + (y_3 - c_y)^2 \leq w_3^2 \\
 & -1 \leq x_i \leq 10 \quad \forall i \in \{1, 2, 3\} \\
 & -1 \leq y_i \leq 10 \quad \forall i \in \{1, 2, 3\} \\
 & 0 \leq w_i \leq 40 \quad \forall i \in \{1, 2, 3\} \\
 & -1 \leq c_x \leq 10 \\
 & -1 \leq c_y \leq 10
 \end{aligned}$$

## 22.8 Machine Learning

---

There are two main fields of machine learning:

- Supervised Machine Learning,
- Unsupervised Machine Learning.

Supervised machine learning is composed of *Regression* and *Classification*. This area is thought of as being given labeled data that you are then trying to understand the trends of this labeled data.

Unsupervised machine learning is where you are given unlabeled data and then need to decide how to label this data. For instance, how can you optimally partition the people in a room into 5 groups that share the most commonalities?

## 22.9 Machine Learning - Supervised Learning - Regression

---

See the video lecture information.

## 22.10 Machine learning - Supervised Learning - Classification

---

The problem of data *classification* begins with *data* and *labels*. The goal is *classification* of future data based on sample data that you have by constructing a function to understand future data.

*Goal:* Classification - create a function  $f(x)$  that takes in a data point  $x$  and outputs the correct label.

These functions can take many forms. In binary classification, the label set is  $\{+1, -1\}$ , and we want to correctly (as often as we can) determine the correct label for a future data point.

There are many ways to determine such a function  $f(x)$ . In the next section, we will learn about SVM that determines the function by computing a hyperplane that separates the data labeled  $+1$  from the data labeled  $-1$ .

Later, we will learn about *neural networks* that describe much more complicated functions.

Another method is to create a *decision tree*. These are typically more interpretable functions (neural networks are often a bit mysterious) and thus sometimes preferred in settings where the classification should be easily understood, such as a medical diagnosis. We will not discuss this method here since it fits less well with the theme of nonlinear programming.

### 22.10.1. Python SGD implementation and video

---

[https://github.com/l1Sourcell/Classifying\\_Data\\_Using\\_a\\_Support\\_Vector\\_Machine/blob/master/support\\_vector\\_machine\\_lesson.ipynb](https://github.com/l1Sourcell/Classifying_Data_Using_a_Support_Vector_Machine/blob/master/support_vector_machine_lesson.ipynb)

# 23. NLP Algorithms

---

## 23.1 Algorithms Introduction

---

We will begin with unconstrained optimization and consider several different algorithms based on what is known about the objective function. In particular, we will consider the cases where we use

- Only function evaluations (also known as *derivative free optimization*),
- Function and gradient evaluations,
- Function, gradient, and hessian evaluations.

We will first look at these algorithms and their convergence rates in the 1-dimensional setting and then extend these results to higher dimensions.

## 23.2 1-Dimensional Algorithms

---

We suppose that we solve the problem

$$\min f(x) \tag{23.1}$$

$$x \in [a, b]. \tag{23.2}$$

That is, we minimize the univariate function  $f(x)$  on the interval  $[l, u]$ .

For example,

$$\min(x^2 - 2)^2 \tag{23.3}$$

$$0 \leq x \leq 10. \tag{23.4}$$

Note, the optimal solution lies at  $x^* = \sqrt{2}$ , which is an irrational number. Since we will consider algorithms using floating point precision, we will look to return a solution  $\bar{x}$  such that  $\|x^* - \bar{x}\| < \varepsilon$  for some small  $\varepsilon > 0$ , for instance,  $\varepsilon = 10^{-6}$ .

### 23.2.1. Golden Search Method - Derivative Free Algorithm

#### Resources

- *Youtube! - Golden Section Search Method*

Suppose that  $f(x)$  is unimodal on the interval  $[a, b]$ , that is, it is a continuous function that has a single minimizer on the interval.

Without any extra information, our best guess for the optimizer is  $\bar{x} = \frac{a+b}{2}$  with a maximum error of  $\epsilon = \frac{b-a}{2}$ . Our goal is to reduce the size of the interval where we know  $x^*$  to be, and hence improve our best guess and the maximum error of our guess.

Now we want to choose points in the interior of the interval to help us decide where the minimizer is. Let  $x_1, x_2$  such that

$$a < x_2 < x_1 < b.$$

Next, we evaluate the function at these four points. Using this information, we would like to argue a smaller interval in which  $x^*$  is contained. In particular, since  $f$  is unimodal, it must hold that

- (a)  $x^* \in [a, x_2]$  if  $f(x_1) \leq f(x_2)$ ,
- (b)  $x^* \in [x_1, b]$  if  $f(x_2) < f(x_1)$ ,

After comparing these function values, we can reduce the size of the interval and hence reduce the region where we think  $x^*$  is.

We will now discuss how to chose  $x_1, x_2$  in a way that we can

- (a) Reuse function evaluations,
- (b) Have a constant multiplicative reduction in the size of the interval.

We consider the picture:

To determine the best  $d$ , we want to decrease by a constant factor. Hence, we decrease be a factor  $\gamma$ , which we will see is the golden ration (GR). To see this, we assume that  $(b - a) = 1$ , and ask that  $d = \gamma$ . Thus,  $x_1 - a = \gamma$  and  $b - x_2 = \gamma$ . If we are in case 1, then we cut off  $b - x_1 = 1 - \gamma$ . Now, if we iterate and do this again, we will have an initial length of  $\gamma$  and we want to cut off the interval  $x_2 - x_1$  with this being a proportion of  $(1 - \gamma)$  of the remaining length. Hence, the second time we will cut of  $(1 - \gamma)\gamma$ , which we set as the length between  $x_1$  and  $x_2$ .

Considering the geometry, we have

$$\text{length } a \text{ to } x_1 + \text{length } x_2 \text{ to } b = \text{total length} + \text{length } x_2 \text{ to } x_1$$

hence

$$\gamma + \gamma = 1 + (1 - \gamma)\gamma.$$

Simplifying, we have

$$\gamma^2 + \gamma - 1 = 0.$$

Applying the quadratic formula, we see

$$\gamma = \frac{-1 \pm \sqrt{5}}{2}.$$

Since we want  $\gamma > 0$ , we take

$$\gamma = \frac{-1 + \sqrt{5}}{2} \approx 0.618$$

This is exactly the Golden Ratio (or, depending on the definition, the golden ratio minus 1).

### 23.2.1.1. Example:

---

We can conclude that the optimal solution is in  $[1.4, 3.8]$ , so we would guess the midpoint  $\bar{x} = 2.6$  as our approximate solution with a maximum error of  $\epsilon = 1.2$ .

#### Convergence Analysis of Golden Search Method:

After  $t$  steps of the Golden Search Method, the interval in question will be of length

$$(b - a)(GR)^t \approx (b - a)(0.618)^t$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b - a)(0.618)^t.$$

## 23.2.2. Bisection Method - 1st Order Method (using Derivative)

---

### 23.2.2.1. Minimization Interpretation

---

**Assumptions:**  $f$  is convex, differentiable

We can look for a minimizer of the function  $f(x)$  on the interval  $[a, b]$ .

### 23.2.2.2. Root finding Interpretation

---

Instead of minimizing, we can look for a root of  $f'(x)$ . That is, find  $x$  such that  $f'(x) = 0$ .

**Assumptions:**  $f'(a) < 0 < f'(b)$ , OR,  $f'(b) < 0 < f'(a)$ .  $f'$  is continuous

The goal is to find a root of the function  $f'(x)$  on the interval  $[a, b]$ . If  $f$  is convex, then we know that this root is indeed a global minimizer.

Note that if  $f$  is convex, it only makes sense to have the assumption  $f'(a) < 0 < f'(b)$ .

**Convergence Analysis of Bisection Method:**

After  $t$  steps of the Bisection Method, the interval in question will be of length

$$(b-a) \left(\frac{1}{2}\right)^t.$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b-a) \left(\frac{1}{2}\right)^t.$$

### 23.2.3. Gradient Descent - 1st Order Method (using Derivative)

---

**Input:**  $f(x)$ ,  $\nabla f(x)$ , initial guess  $x^0$ , learning rate  $\alpha$ , tolerance  $\varepsilon$

**Output:** An approximate solution  $x$

- (a) Set  $t = 0$
- (b) While  $\|f(x^t)\|_2 > \varepsilon$ :
  - i. Set  $x^{t+1} \leftarrow x^t - \alpha \nabla f(x^t)$ .
  - ii. Set  $t \leftarrow t + 1$ .
- (c) Return  $x^t$ .

### 23.2.4. Newton's Method - 2nd Order Method (using Derivative and Hessian)

---

**Input:**  $f(x)$ ,  $\nabla f(x)$ ,  $\nabla^2 f(x)$ , initial guess  $x^0$ , learning rate  $\alpha$ , tolerance  $\varepsilon$

**Output:** An approximate solution  $x$

- (a) Set  $t = 0$
- (b) While  $\|f(x^t)\|_2 > \varepsilon$ :
  - i. Set  $x^{t+1} \leftarrow x^t - \alpha [\nabla^2 f(x^t)]^{-1} \nabla f(x^t)$ .
  - ii. Set  $t \leftarrow t + 1$ .
- (c) Return  $x^t$ .

## 23.3 Multi-Variate Unconstrained Optimizaiton

---

We will now use the techniques for 1-Dimensional optimization and extend them to multi-variate case. We will begin with unconstrained versions (or at least, constrained to a large box) and then show how we can apply these techniques to constrained optimization.

### 23.3.1. Descent Methods - Unconstrained Optimization - Gradient, Newton

---

#### Outline for Descent Method for Unconstrained Optimization:

##### Input:

- A function  $f(x)$
- Initial solution  $x^0$
- Method for computing step direction  $d_t$
- Method for computing length  $t$  of step
- Number of iterations  $T$

##### Output:

- A point  $x_T$  (hopefully an approximate minimizer)

##### Algorithm

- (a) For  $t = 1, \dots, T$ ,

$$\text{set } x_{t+1} = x_t + \alpha_t d_t$$

#### 23.3.1.1. Choice of $\alpha_t$

---

There are many different ways to choose the step length  $\alpha_t$ . Some choices have proofs that the algorithm will converge quickly. An easy choice is to have a constant step length  $\alpha_t = \alpha$ , but this may depend on the specific problem.

#### 23.3.1.2. Choice of $d_t$ using $\nabla f(x)$

---

Choice of descent methods using  $\nabla f(x)$  are known as *first order methods*. Here are some choices:

- (a) **Gradient Descent:**  $d_t = -\nabla f(x_t)$
- (b) **Nesterov Accelerated Descent:**  $d_t = \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1}))$

Here,  $\mu, \gamma$  are some numbers. The number  $\mu$  is called the momentum.

### 23.3.2. Stochastic Gradient Descent - The mother of all algorithms.

A popular method is called *stochastic gradient descent* (SGD). This has been described as "The mother of all algorithms". This is a method to **approximate the gradient** typically used in machine learning or stochastic programming settings.

#### Stochastic Gradient Descent:

Suppose we want to solve

$$\min_{x \in \mathbb{R}^n} F(x) = \sum_{i=1}^N f_i(x). \quad (23.1)$$

We could use *gradient descent* and have to compute the gradient  $\nabla F(x)$  at each iteration. But! We see that in the **cost to compute the gradient** is roughly  $O(nN)$ , that is, it is very dependent on the number of function  $N$ , and hence each iteration will take time dependent on  $N$ .

**Instead!** Let  $i$  be a uniformly random sample from  $\{1, \dots, N\}$ . Then we will use  $\nabla f_i(x)$  as an approximation of  $\nabla F(x)$ . Although we lose a bit by using a guess of the gradient, this approximation only takes  $O(n)$  time to compute. And in fact, in expectation, we are doing the same thing. That is,

$$N \cdot \mathbb{E}(\nabla f_i(x)) = N \sum_{i=1}^N \frac{1}{N} \nabla f_i(x) = \sum_{i=1}^N \nabla f_i(x) = \nabla \left( \sum_{i=1}^N f_i(x) \right) = \nabla F(x).$$

Hence, the SGD algorithm is:

- (a) Set  $t = 0$
- (b) While ... (some stopping criterion)
  - i. Choose  $i$  uniformly at random in  $\{1, \dots, N\}$ .
  - ii. Set  $d_t = \nabla f_i(x_t)$
  - iii. Set  $x_{t+1} = x_t - \alpha d_t$

There can be many variations on how to decide which functions  $f_i$  to evaluate gradient information on. Above is just one example.

Linear regression is an excellent example of this.

#### Example 23.1: Linear Regression with SGD

Given data points  $x^1, \dots, x^N \in \mathbb{R}^d$  and output  $y^1, \dots, y^N \in \mathbb{R}$ , find  $a \in \mathbb{R}^d$  and  $b \in \mathbb{R}$  such that  $a^\top x^i + b \approx y^i$ . This can be written as the optimization problem

$$\min_{a,b} \sum_{i=1}^N g_i(a,b) \quad (23.2)$$

where  $g_i(a,b) = (a^\top x^i + b)^2$ .

Notice that the objective function  $G(a,b) = \sum_{i=1}^N g_i(a,b)$  is a convex quadratic function. The

gradient of the objective function is

$$\nabla G(a, b) = \sum_{i=1}^N \nabla g_i(a, b) = \sum_{i=1}^N 2x^i(a^\top x^i + b)$$

Hence, if we want to use gradient descent, we must compute this large sum (think of  $N \approx 10,000$ ).

Instead, we can **approximate the gradient!**. Let  $\tilde{\nabla}G(a, b)$  be our approximate gradient. We will compute this by randomly choosing a value  $r \in \{1, \dots, N\}$  (with uniform probability). Then set

$$\tilde{\nabla}G(a, b) = \nabla g_r(a, b).$$

It holds that the expected value is the same as the gradient, that is,

$$\mathbb{E}(\tilde{\nabla}G(a, b)) = G(a, b).$$

Hence, we can make probabilistic arguments that these two will have the same (or similar) convergence properties (in expectation).

### 23.3.3. Neural Networks

---

#### Resources

- ML Zero to Hero - Part 1: Intro to machine learning
- ML Zero to Hero - Part 2: Basic Computer Vision with ML

### 23.3.4. Choice of $\Delta_k$ using the hessian $\nabla^2 f(x)$

---

These choices are called *second order methods*

- (a) **Newton's Method:**  $\Delta_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- (b) **BFGS (Quasi-Newton):**  $\Delta_k = -(B_k)^{-1} \nabla f(x_k)$

Here

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \end{aligned}$$

and

$$B_{k+1} = B_k - \frac{(B_k s_k)(B_k s_k)^\top}{s_k^\top B_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}.$$

This serves as an approximation of the hessian and can be efficiently computed. Furthermore, the inverse can be easily computed using certain updating rules. This makes for a fast way to approximate the hessian.

## 23.4 Constrained Convex Nonlinear Programming

---

Given a convex function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{23.1}$$

### 23.4.1. Barrier Method

---

#### Constrained Convex Programming via Barrier Method:

We convert 23.1 into the unconstrained minimization problem:

$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^m \log(-f_i(x)) \\ & x \in \mathbb{R}^d \end{aligned} \tag{23.2}$$

Here  $\phi > 0$  is some number that we choose. As  $\phi \rightarrow 0$ , the optimal solution  $x(\phi)$  to (23.2) tends to the optimal solution of (23.1). That is  $x(\phi) \rightarrow x^*$  as  $\phi \rightarrow 0$ .

#### Constrained Convex Programming via Barrier Method - Initial solution:

Define a variable  $s \in \mathbb{R}$  and add that to the right hand side of the inequalities and then minimize it in the objective function.

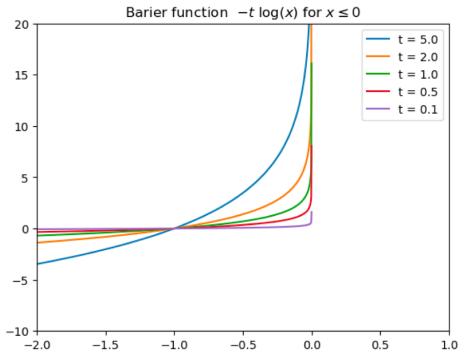
$$\begin{aligned} \min \quad & s \\ \text{s.t.} \quad & f_i(x) \leq s \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \tag{23.3}$$

Note that this problem is feasible for all  $x$  values since  $s$  can always be made larger. If there exists a solution with  $s \leq 0$ , then we can use the corresponding  $x$  solution as an initial feasible solution. Otherwise, the problem is infeasible.

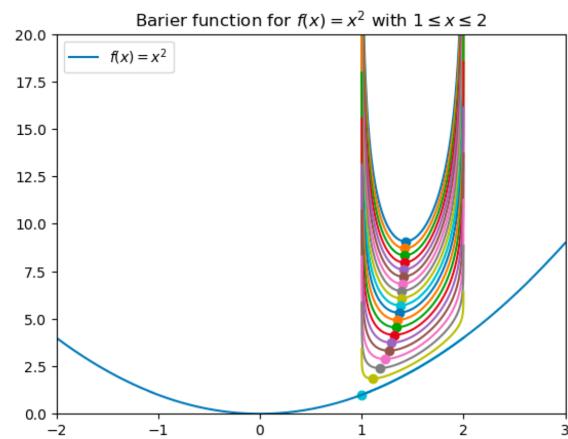
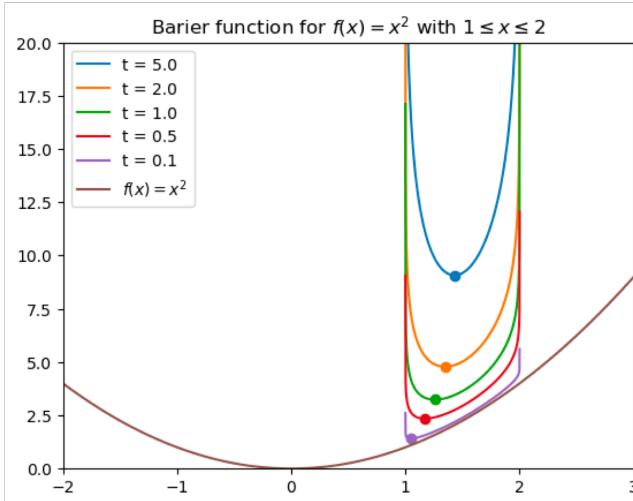
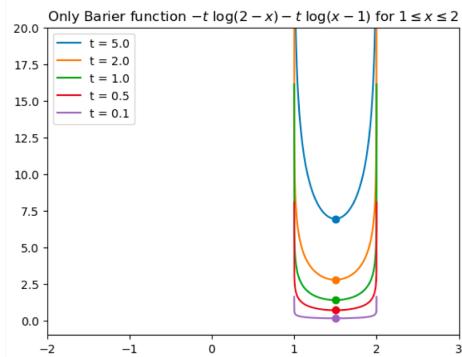
Now, convert this problem into the unconstrained minimization problem:

$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^m \log(-(f_i(x) - s)) \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \tag{23.4}$$

This problem has an easy time of finding an initial feasible solution. For instance, let  $x = 0$ , and then  $s = \max_i f_i(x) + 1$ .

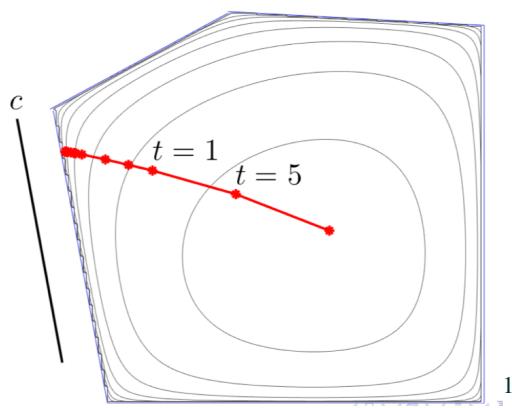


**Images below: the value  $t$  is the value  $\phi$  discussed above**



---

Minimizing  $c^T x$  subject to  $Ax \leq b$ .



---

<sup>1</sup>Image taken from unknown source.

# 24. Computational Issues with NLP

---

We mention a few computational issues to consider with nonlinear programs.

## 24.1 Irrational Solutions

---

Consider nonlinear problem (this is even convex)

$$\begin{array}{ll} \min & -x \\ \text{s.t.} & x^2 \leq 2. \end{array} \quad (24.1)$$

The optimal solution is  $x^* = \sqrt{2}$ , which cannot be easily represented. Hence, we would settle for an **approximate solution** such as  $\bar{x} = 1.41421$ , which is feasible since  $\bar{x}^2 \leq 2$ , and it is close to optimal.

## 24.2 Discrete Solutions

---

Consider nonlinear problem (not convex)

$$\begin{array}{ll} \min & -x \\ \text{s.t.} & x^2 = 2. \end{array} \quad (24.1)$$

Just as before, the optimal solution is  $x^* = \sqrt{2}$ , which cannot be easily represented. Furthermore, the only two feasible solutions are  $\sqrt{2}$  and  $-\sqrt{2}$ . Thus, there is no chance to write down a feasible rational approximation.

## 24.3 Convex NLP Harder than LP

---

Convex NLP is typically polynomially solvable. It is a generalization of linear programming.

### Convex Programming:

*Polynomial time (P)* (typically)

Given a convex function  $f(x) : \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $f_i(x) : \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{array} \quad (24.1)$$

**Example 24.1: C**

*nvex programming is a generalization of linear programming. This can be seen by letting  $f(x) = c^\top x$  and  $f_i(x) = A_i x - b_i$ .*

## 24.4 NLP is harder than IP

---

As seen above, quadratic constraints can be used to create a feasible region with discrete solutions. For example

$$x(1-x) = 0$$

has exactly two solutions:  $x = 0, x = 1$ . Thus, quadratic constraints can be used to model binary constraints.

### Binary Integer programming (BIP) as a NLP:

#### *NP-Hard*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0,1\}^n \\ & x_i(1-x_i) = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \tag{24.1}$$

## 24.5 Karush-Huhn-Tucker (KKT) Conditions

---

The KKT conditions use the augmented Lagrangian problem to describe sufficient conditions for optimality of a convex program.

### KKT Conditions for Optimality:

Given a convex function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $g_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{aligned} & \min f(x) \\ \text{s.t. } & g_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{24.1}$$

Given  $(\bar{x}, \bar{\lambda})$  with  $\bar{x} \in \mathbb{R}^d$  and  $\bar{\lambda} \in \mathbb{R}^m$ , if the KKT conditions hold, then  $\bar{x}$  is optimal for the convex programming problem.

The KKT conditions are

(a) (Stationary).

$$-\nabla f(\bar{x}) = \sum_{i=1}^m \bar{\lambda}_i \nabla g_i(\bar{x}) \tag{24.2}$$

(b) (Complimentary Slackness).

$$\bar{\lambda}_i g_i(\bar{x}) = 0 \text{ for } i = 1, \dots, m \tag{24.3}$$

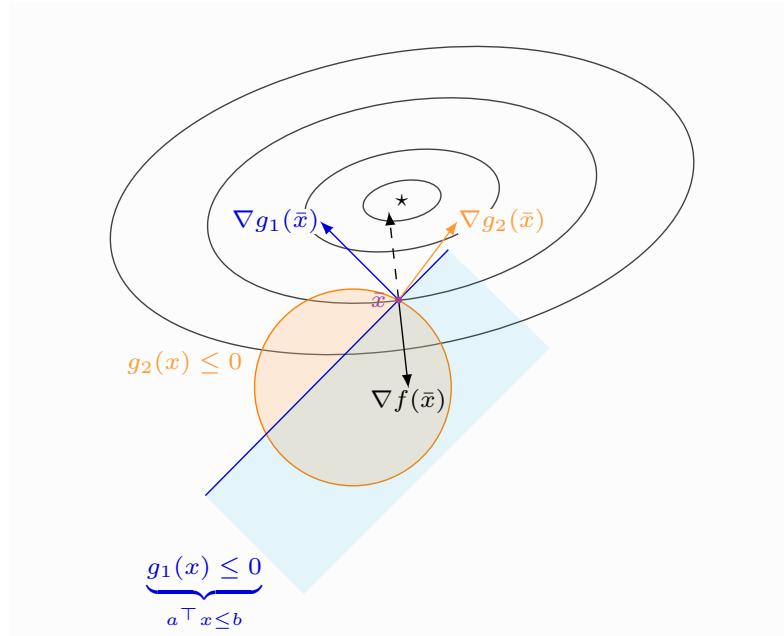
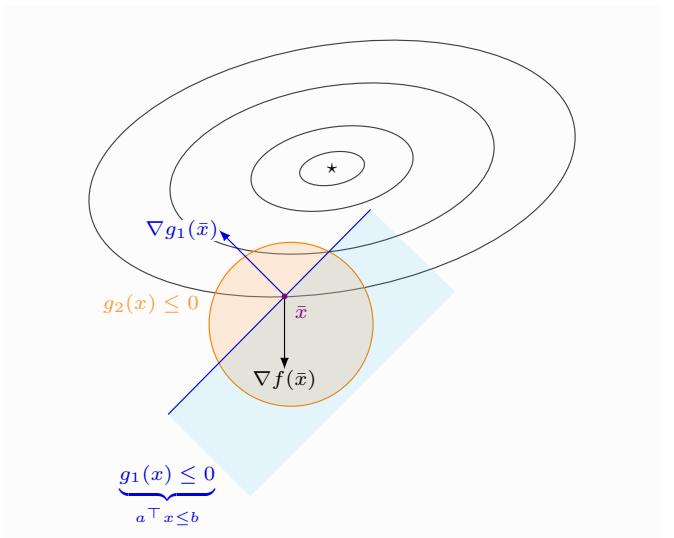
(c) (Primal Feasibility).

$$g_i(\bar{x}) \leq 0 \text{ for } i = 1, \dots, m \tag{24.4}$$

(d) (Dual Feasibility).

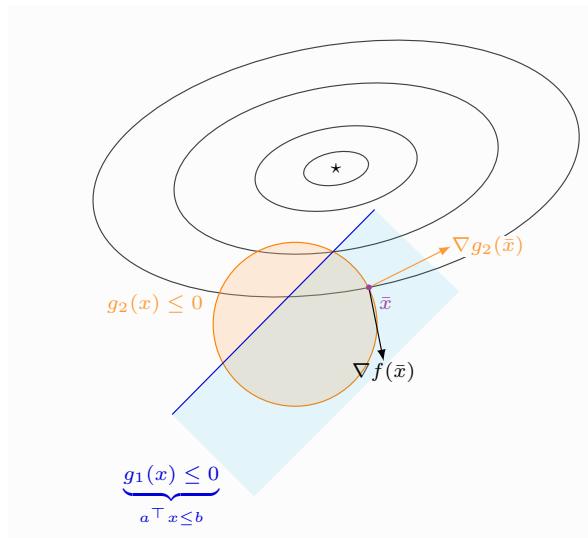
$$\bar{\lambda}_i \geq 0 \text{ for } i = 1, \dots, m \tag{24.5}$$

If certain properties are true of the convex program, then every optimizer has these properties. In particular, this holds for Linear Programming.

© tikz/kkt-optimal<sup>1</sup>© tikz/kkt-non-optimal1<sup>2</sup>


---

tikz/kkt-non-optimal1, from tikz/kkt-non-optimal1.  
tikz/kkt-non-optimal1, tikz/kkt-non-optimal1.

© tikz/kkt-non-optimal2<sup>3</sup>


---

tikz/kkt-non-optimal2, from tikz/kkt-non-optimal2.  
tikz/kkt-non-optimal2, tikz/kkt-non-optimal2.

---

<sup>1</sup>tikz/kkt-optimal, from tikz/kkt-optimal. tikz/kkt-optimal, tikz/kkt-optimal.

## 24.6 Gradient Free Algorithms

---

### 24.6.1. Nelder-Mead

---

#### Resources

- *Wikipedia*
- *Youtube*



# 25. Material to add...

---

## 25.0.1. Bisection Method and Newton's Method

---

### Resources

- See section 4 of the following nodes: <http://www.seas.ucla.edu/~vandenbe/133A/133A-notes.pdf>

# 25.1 Gradient Descent

---

### Resources

Recap Gradient and Directional Derivatives:

- <https://www.youtube.com/watch?v=tIpKfDc295M>
- [https://www.youtube.com/watch?v=\\_-02ze7tf08](https://www.youtube.com/watch?v=_-02ze7tf08)
- [https://www.youtube.com/watch?v=N\\_ZRcLheNv0](https://www.youtube.com/watch?v=N_ZRcLheNv0)
- <https://www.youtube.com/watch?v=4RBkIJPG6Yo>

Idea of Gradient descent:

- <https://youtu.be/IHZwWFHwa-w?t=323>

Vectors:

- [https://www.youtube.com/watch?v=fNk\\_zzaMoSs&list=PLZHQB0WTQDPD3MizzM2xVFitgF8hE\\_ab&index=2&t=0s](https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQB0WTQDPD3MizzM2xVFitgF8hE_ab&index=2&t=0s)



## 26. Fairness in Algorithms

---

### Resources

- *Simons Institute - Michael Kearns (University of Pennsylvania) - "The Ethical Algorithm"*



## **Part VI**

### **Appendix - Linear Algebra Background**



# A. Linear Transformations

---

**Lab Objective:** *Linear transformations are the most basic and essential operators in vector space theory. In this lab we visually explore how linear transformations alter points in the Cartesian plane. We also empirically explore the computational cost of applying linear transformations via matrix multiplication.*

## Linear Transformations

---

A *linear transformation* is a mapping between vector spaces that preserves addition and scalar multiplication. More precisely, let  $V$  and  $W$  be vector spaces over a common field  $\mathbb{F}$ . A map  $L : V \rightarrow W$  is a linear transformation from  $V$  into  $W$  if

$$L(a\mathbf{x}_1 + b\mathbf{x}_2) = aL\mathbf{x}_1 + bL\mathbf{x}_2$$

for all vectors  $\mathbf{x}_1, \mathbf{x}_2 \in V$  and scalars  $a, b \in \mathbb{F}$ .

Every linear transformation  $L$  from an  $m$ -dimensional vector space into an  $n$ -dimensional vector space can be represented by an  $m \times n$  matrix  $A$ , called the *matrix representation* of  $L$ . To apply  $L$  to a vector  $\mathbf{x}$ , left multiply by its matrix representation. This results in a new vector  $\mathbf{x}'$ , where each component is some linear combination of the elements of  $\mathbf{x}$ . For linear transformations from  $\mathbb{R}^2$  to  $\mathbb{R}^2$ , this process has the form

$$A\mathbf{x} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{x}'.$$

Linear transformations can be interpreted geometrically. To demonstrate this, consider the array of points  $H$  that collectively form a picture of a horse, stored in the file `horse.npy`. The coordinate pairs  $\mathbf{x}_i$  are organized by column, so the array has two rows: one for  $x$ -coordinates, and one for  $y$ -coordinates. Matrix multiplication on the left transforms each coordinate pair, resulting in another matrix  $H'$  whose columns are the transformed coordinate pairs:

$$\begin{aligned} AH &= A \begin{bmatrix} x_1 & x_2 & x_3 & \dots \\ y_1 & y_2 & y_3 & \dots \end{bmatrix} = A \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \dots \end{bmatrix} = \begin{bmatrix} A\mathbf{x}_1 & A\mathbf{x}_2 & A\mathbf{x}_3 & \dots \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{x}'_1 & \mathbf{x}'_2 & \mathbf{x}'_3 & \dots \end{bmatrix} = \begin{bmatrix} x'_1 & x'_2 & x'_3 & \dots \\ y'_1 & y'_2 & y'_3 & \dots \end{bmatrix} = H'. \end{aligned}$$

To begin, use `np.load()` to extract the array from the `npy` file, then plot the unaltered points as individual pixels. See Figure A.1 for the result.

```

>>> import numpy as np
>>> from matplotlib import pyplot as plt

# Load the array from the .npy file.
>>> data = np.load("horse.npy")

# Plot the x row against the y row with black pixels.
>>> plt.plot(data[0], data[1], 'k,')

# Set the window limits to [-1, 1] by [-1, 1] and make the window square.
>>> plt.axis([-1,1,-1,1])
>>> plt.gca().set_aspect("equal")
>>> plt.show()

```

## Types of Linear Transformations

---

Linear transformations from  $\mathbb{R}^2$  into  $\mathbb{R}^2$  can be classified in a few ways.

- **Stretch:** Stretches or compresses the vector along each axis. The matrix representation is diagonal:

$$\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}.$$

If  $a = b$ , the transformation is called a *dilation*. The stretch in Figure A.1 uses  $a = \frac{1}{2}$  and  $b = \frac{6}{5}$  to compress the  $x$ -axis and stretch the  $y$ -axis.

- **Shear:** Slants the vector by a scalar factor horizontally or vertically (or both simultaneously). The matrix representation is

$$\begin{bmatrix} 1 & a \\ b & 1 \end{bmatrix}.$$

Pure horizontal shears ( $b = 0$ ) skew the  $x$ -coordinate of the vector while pure vertical shears ( $a = 0$ ) skew the  $y$ -coordinate. Figure A.1 has a horizontal shear with  $a = \frac{1}{2}, b = 0$ .

- **Reflection:** Reflects the vector about a line that passes through the origin. The reflection about the line spanned by the vector  $[a, b]^T$  has the matrix representation

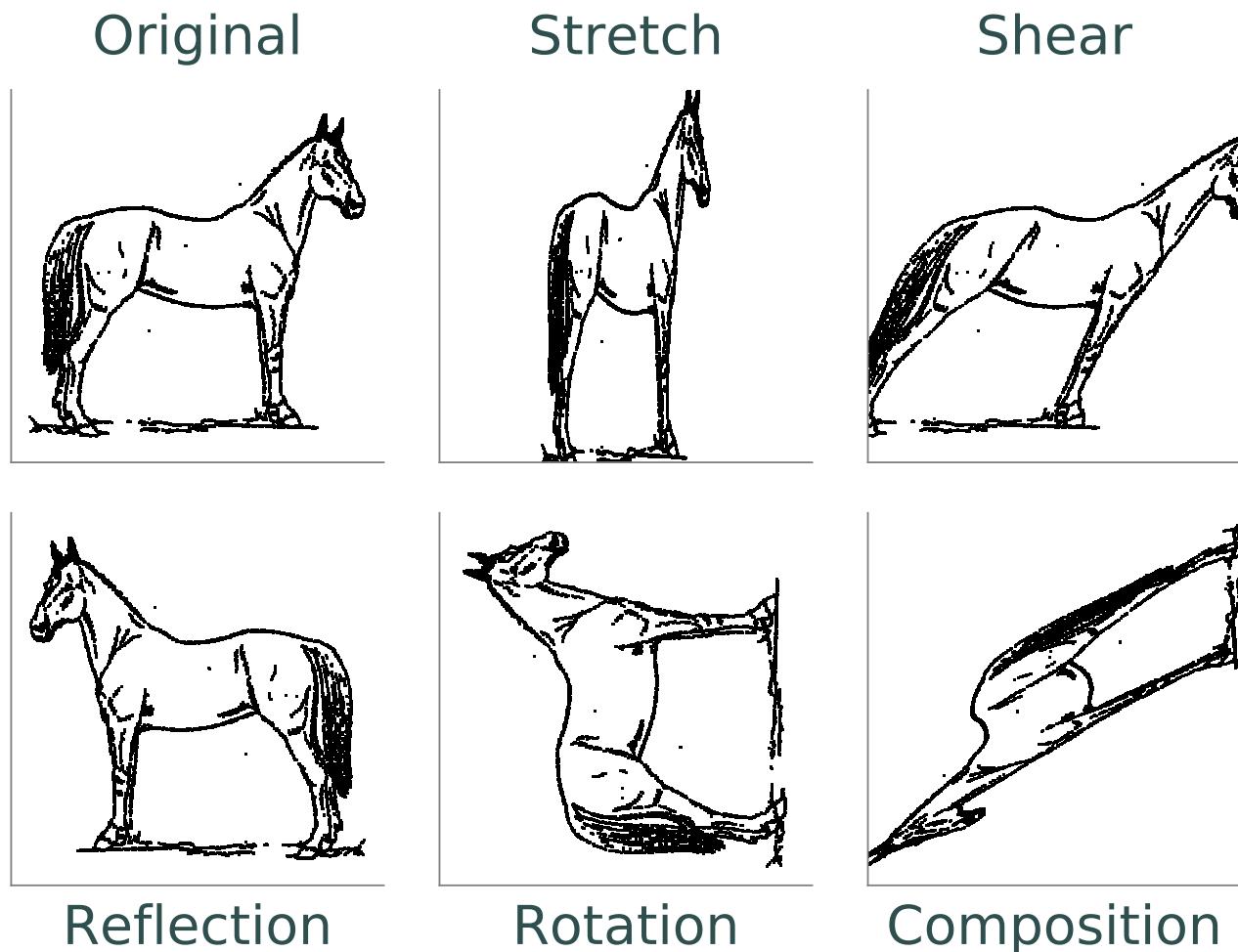
$$\frac{1}{a^2 + b^2} \begin{bmatrix} a^2 - b^2 & 2ab \\ 2ab & b^2 - a^2 \end{bmatrix}.$$

The reflection in Figure A.1 reflects the image about the  $y$ -axis ( $a = 0, b = 1$ ).

- **Rotation:** Rotates the vector around the origin. A counterclockwise rotation of  $\theta$  radians has the following matrix representation:

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

A negative value of  $\theta$  performs a clockwise rotation. Choosing  $\theta = \frac{\pi}{2}$  produces the rotation in Figure A.1.



**Figure A.1:** The points stored in `horse.npy` under various linear transformations.

#### Problem A.1: Implement linear transformations.

Write a function for each type of linear transformation. Each function should accept an array to transform and the scalars that define the transformation ( $a$  and  $b$  for stretch, shear, and reflection, and  $\theta$  for rotation). Construct the matrix representation, left multiply it with the input array, and return the transformed array.

To test these functions, write a function to plot the original points in `horse.npy` together with the transformed points in subplots for a side-by-side comparison. Compare your results to Figure A.1.

## Compositions of Linear Transformations

---

Let  $V$ ,  $W$ , and  $Z$  be finite-dimensional vector spaces. If  $L : V \rightarrow W$  and  $K : W \rightarrow Z$  are linear transformations with matrix representations  $A$  and  $B$ , respectively, then the *composition* function  $KL : V \rightarrow Z$  is also a linear transformation, and its matrix representation is the matrix product  $BA$ .

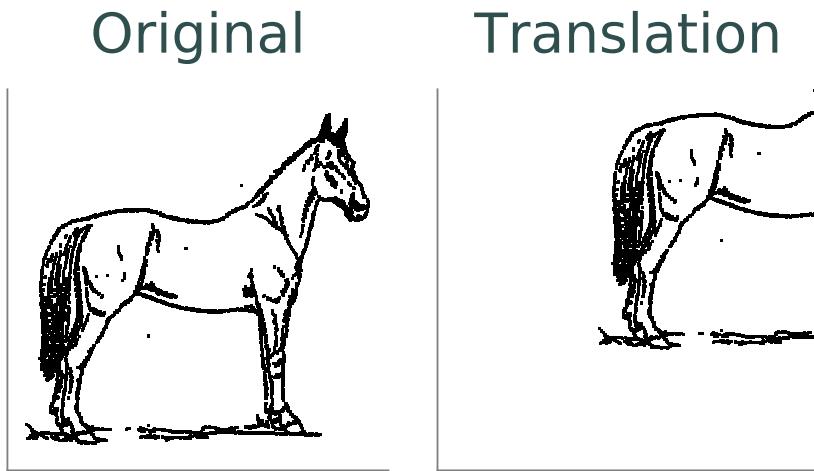
For example, if  $S$  is a matrix representing a shear and  $R$  is a matrix representing a rotation, then  $RS$  represents a shear followed by a rotation. In fact, any linear transformation  $L : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  is a composition of the four transformations discussed above. Figure A.1 displays the composition of all four previous transformations, applied in order (stretch, shear, reflection, then rotation).

## Affine Transformations

---

All linear transformations map the origin to itself. An *affine transformation* is a mapping between vector spaces that preserves the relationships between points and lines, but that may not preserve the origin. Every affine transformation  $T$  can be represented by a matrix  $A$  and a vector  $b$ . To apply  $T$  to a vector  $x$ , calculate  $Ax + b$ . If  $b = 0$  then the transformation is linear, and if  $A = I$  but  $b \neq 0$  then it is called a *translation*.

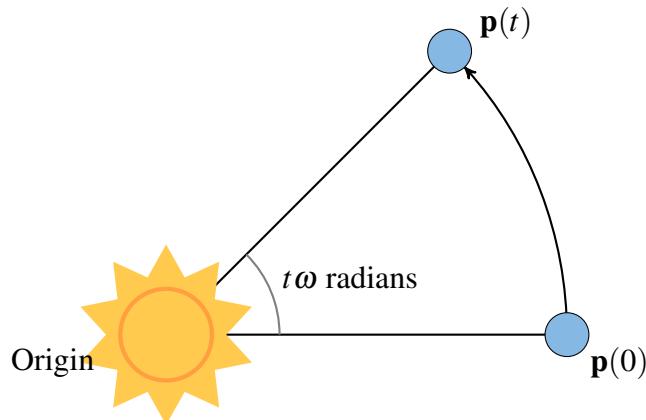
For example, if  $T$  is the translation with  $\mathbf{b} = \left[ \frac{3}{4}, \frac{1}{2} \right]^T$ , then applying  $T$  to an image will shift it right by  $\frac{3}{4}$  and up by  $\frac{1}{2}$ . This translation is illustrated below.



Affine transformations include all compositions of stretches, shears, rotations, reflections, and translations. For example, if  $S$  represents a shear and  $R$  a rotation, and if  $b$  is a vector, then  $RS\mathbf{x} + b$  shears, then rotates, then translates  $\mathbf{x}$ .

## Modeling Motion with Affine Transformations

Affine transformations can be used to model particle motion, such as a planet rotating around the sun. Let the sun be the origin, the planet's location at time  $t$  be given by the vector  $\mathbf{p}(t)$ , and suppose the planet has angular velocity  $\omega$  (a measure of how fast the planet goes around the sun). To find the planet's position at time  $t$  given the planet's initial position  $\mathbf{p}(0)$ , rotate the vector  $\mathbf{p}(0)$  around the origin by  $t\omega$  radians. Thus if  $R(\theta)$  is the matrix representation of the linear transformation that rotates a vector around the origin by  $\theta$  radians, then  $\mathbf{p}(t) = R(t\omega)\mathbf{p}(0)$ .



Composing the rotation with a translation shifts the center of rotation away from the origin, yielding more complicated motion.

**Problem A.2: Moon orbiting the earth orbiting the sun.**

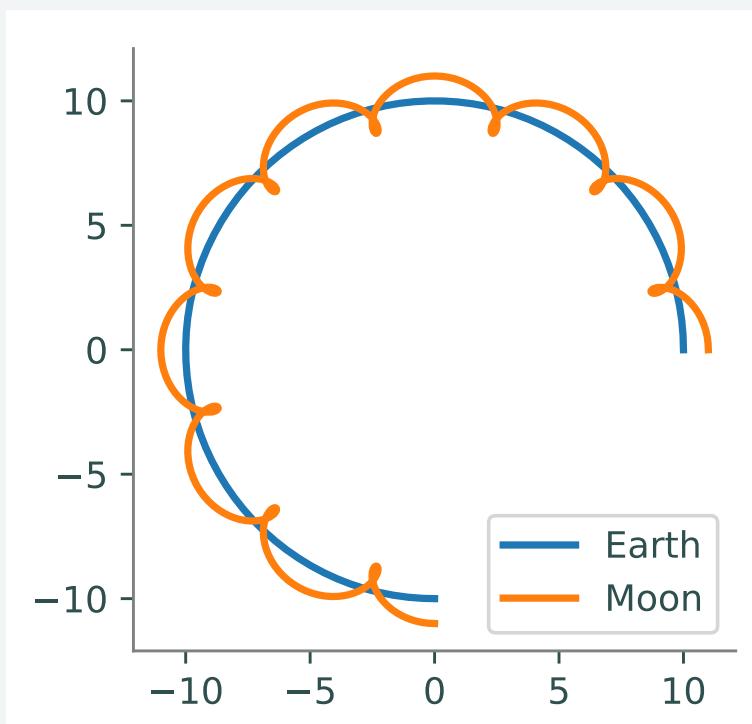
he moon orbits the earth while the earth orbits the sun. Assuming circular orbits, we can compute the trajectories of both the earth and the moon using only linear and affine transformations.

Assume an orientation where both the earth and moon travel counterclockwise, with the sun at the origin. Let  $\mathbf{p}_e(t)$  and  $\mathbf{p}_m(t)$  be the positions of the earth and the moon at time  $t$ , respectively, and let  $\omega_e$  and  $\omega_m$  be each celestial body's angular velocity. For a particular time  $t$ , we calculate  $\mathbf{p}_e(t)$  and  $\mathbf{p}_m(t)$  with the following steps.

- Compute  $\mathbf{p}_e(t)$  by rotating the initial vector  $\mathbf{p}_e(0)$  counterclockwise about the origin by  $t\omega_e$  radians.
- Calculate the position of the moon relative to the earth at time  $t$  by rotating the vector  $\mathbf{p}_m(0) - \mathbf{p}_e(0)$  counterclockwise about the origin by  $t\omega_m$  radians.
- To compute  $\mathbf{p}_m(t)$ , translate the vector resulting from the previous step by  $\mathbf{p}_e(t)$ .

Write a function that accepts a final time  $T$ , initial positions  $x_e$  and  $x_m$ , and the angular momenta  $\omega_e$  and  $\omega_m$ . Assuming initial positions  $\mathbf{p}_e(0) = (x_e, 0)$  and  $\mathbf{p}_m(0) = (x_m, 0)$ , plot  $\mathbf{p}_e(t)$  and  $\mathbf{p}_m(t)$  over the time interval  $t \in [0, T]$ .

Setting  $T = \frac{3\pi}{2}$ ,  $x_e = 10$ ,  $x_m = 11$ ,  $\omega_e = 1$ , and  $\omega_m = 13$ , your plot should resemble the following figure (fix the aspect ratio with `ax.set_aspect("equal")`). Note that a more celestially accurate figure would use  $x_e = 400$ ,  $x_m = 401$  (the interested reader should see <http://www.math.nus.edu.sg/aslaksen/teaching/convex.html>).



# Timing Matrix Operations

---

Linear transformations are easy to perform via matrix multiplication. However, performing matrix multiplication with very large matrices can strain a machine's time and memory constraints. For the remainder of this lab we take an empirical approach in exploring how much time and memory different matrix operations require.

## Timing Code

---

Recall that the `time` module's `time()` function measures the number of seconds since the Epoch. To measure how long it takes for code to run, record the time just before and just after the code in question, then subtract the first measurement from the second to get the number of seconds that have passed. Additionally, in IPython, the quick command `%timeit` uses the `timeit` module to quickly time a single line of code.

```
In [1]: import time

In [2]: def for_loop():
...:     """Go through ten million iterations of nothing."""
...:     for _ in range(int(1e7)):
...:         pass

In [3]: def time_for_loop():
...:     """Time for_loop() with time.time()."""
...:     start = time.time()           # Clock the starting time.
...:     for_loop()
...:     return time.time() - start   # Return the elapsed time.

In [4]: time_for_loop()
0.24458789825439453

In [5]: %timeit for_loop()
248 ms +- 5.35 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

## Timing an Algorithm

Most algorithms have at least one input that dictates the size of the problem to be solved. For example, the following functions take in a single integer  $n$  and produce a random vector of length  $n$  as a list or a random  $n \times n$  matrix as a list of lists.

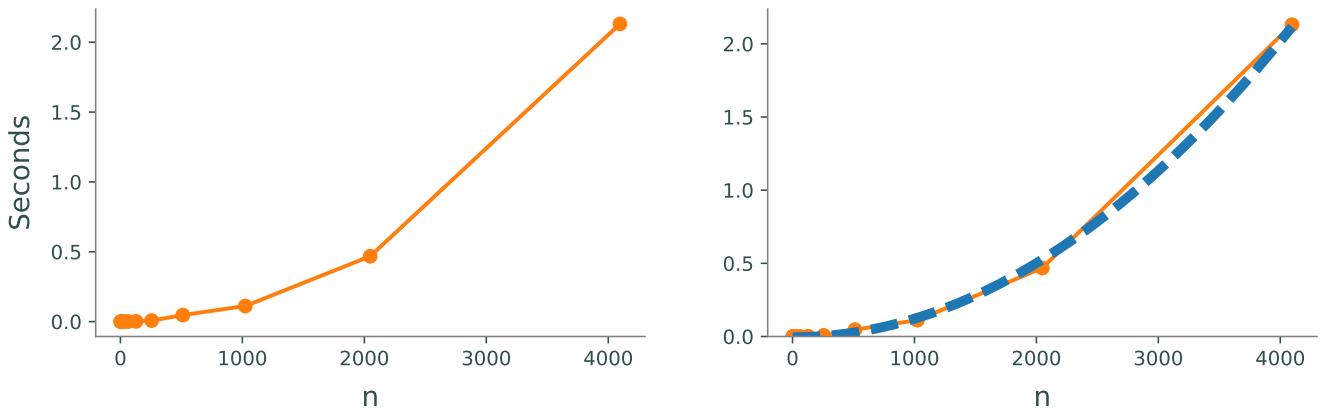
```
from random import random
def random_vector(n):      # Equivalent to np.random.random(n).tolist()
    """Generate a random vector of length n as a list."""
    return [random() for i in range(n)]

def random_matrix(n):      # Equivalent to np.random.random((n,n)).←
    tolist()
    """Generate a random nxn matrix as a list of lists."""
    return [[random() for j in range(n)] for i in range(n)]
```

Executing `random_vector(n)` calls `random()`  $n$  times, so doubling  $n$  should about double the amount of time `random_vector(n)` takes to execute. By contrast, executing `random_matrix(n)` calls `random()`  $n^2$  times ( $n$  times per row with  $n$  rows). Therefore doubling  $n$  will likely more than double the amount of time `random_matrix(n)` takes to execute, especially if  $n$  is large.

To visualize this phenomenon, we time `random_matrix()` for  $n = 2^1, 2^2, \dots, 2^{12}$  and plot  $n$  against the execution time. The result is displayed below on the left.

```
>>> domain = 2**np.arange(1,13)
>>> times = []
>>> for n in domain:
...     start = time.time()
...     random_matrix(n)
...     times.append(time.time() - start)
...
>>> plt.plot(domain, times, 'g.-', linewidth=2, markersize=15)
>>> plt.xlabel("n", fontsize=14)
>>> plt.ylabel("Seconds", fontsize=14)
>>> plt.show()
```



The figure on the left shows that the execution time for `random_matrix(n)` increases quadratically in  $n$ . In fact, the blue dotted line in the figure on the right is the parabola  $y = an^2$ , which fits nicely over the timed observations. Here  $a$  is a small constant, but it is much less significant than the exponent on the  $n$ . To represent this algorithm's growth, we ignore  $a$  altogether and write  $\text{random\_matrix}(n) \sim n^2$ .

### NOTE

An algorithm like `random_matrix(n)` whose execution time increases quadratically with  $n$  is called  $O(n^2)$ , notated by  $\text{random\_matrix}(n) \in O(n^2)$ . Big-oh notation is common for indicating both the *temporal complexity* of an algorithm (how the execution time grows with  $n$ ) and the *spatial complexity* (how the memory usage grows with  $n$ ).

### Problem A.3: Time Matrix-Vector and Matrix-Matrix Multiplication

Let  $A$  be an  $m \times n$  matrix with entries  $a_{ij}$ ,  $\mathbf{x}$  be an  $n \times 1$  vector with entries  $x_k$ , and  $B$  be an  $n \times p$  matrix with entries  $b_{ij}$ . The matrix-vector product  $A\mathbf{x} = \mathbf{y}$  is a new  $m \times 1$  vector and the matrix-matrix product  $AB = C$  is a new  $m \times p$  matrix. The entries  $y_i$  of  $\mathbf{y}$  and  $c_{ij}$  of  $C$  are determined by the following formulas:

$$y_i = \sum_{k=1}^n a_{ik}x_k \quad c_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$$

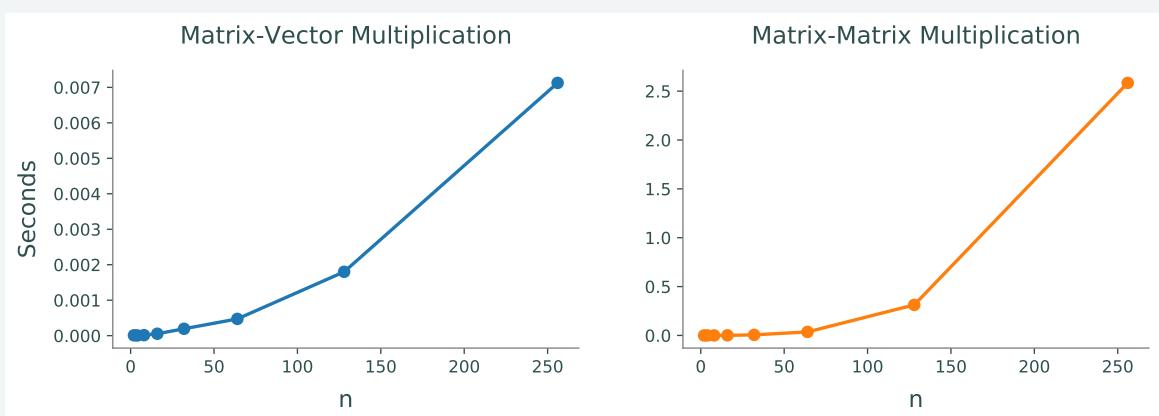
These formulas are implemented below **without** using NumPy arrays or operations.

```
def matrix_vector_product(A, x):      # Equivalent to np.dot(A,x).tolist()
    """Compute the matrix-vector product Ax as a list."""
    m, n = len(A), len(x)
    return [sum([A[i][k] * x[k] for k in range(n)]) for i in range(m)]
}

def matrix_matrix_product(A, B):      # Equivalent to np.dot(A,B).tolist()
    """Compute the matrix-matrix product AB as a list of lists."""
    m, n, p = len(A), len(B), len(B[0])
    return [[sum([A[i][k] * B[k][j] for k in range(n)])
            for j in range(p)] for i in range(m)]
```

Time each of these functions with increasingly large inputs. Generate the inputs  $A$ ,  $\mathbf{x}$ , and  $B$  with `random_matrix()` and `random_vector()` (so each input will be  $n \times n$  or  $n \times 1$ ). Only time the multiplication functions, not the generating functions.

Report your findings in a single figure with two subplots: one with matrix-vector times, and one with matrix-matrix times. Choose a domain for  $n$  so that your figure accurately describes the growth, but avoid values of  $n$  that lead to execution times of more than 1 minute. Your figure should resemble the following plots.



## Logarithmic Plots

Though the two plots from Problem A look similar, the scales on the  $y$ -axes show that the actual execution times differ greatly. To be compared correctly, the results need to be viewed differently.

A *logarithmic plot* uses a logarithmic scale—with values that increase exponentially, such as  $10^1, 10^2, 10^3, \dots$ —on one or both of its axes. The three kinds of log plots are listed below.

- **log-lin:** the  $x$ -axis uses a logarithmic scale but the  $y$ -axis uses a linear scale.  
Use `plt.semilogx()` instead of `plt.plot()`.
- **lin-log:** the  $x$ -axis is uses a linear scale but the  $y$ -axis uses a log scale.  
Use `plt.semilogy()` instead of `plt.plot()`.
- **log-log:** both the  $x$  and  $y$ -axis use a logarithmic scale.  
Use `plt.loglog()` instead of `plt.plot()`.

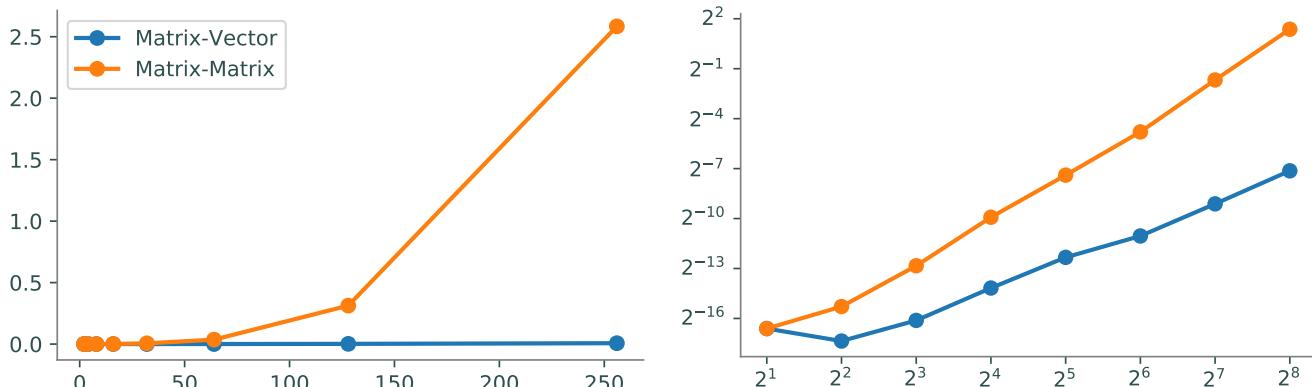
Since the domain  $n = 2^1, 2^2, \dots$  is a logarithmic scale and the execution times increase quadratically, we visualize the results of the previous problem with a log-log plot. The default base for the logarithmic scales on logarithmic plots in Matplotlib is 10. To change the base to 2 on each axis, specify the keyword arguments `baseX=2` and `baseY=2`.

Suppose the domain of  $n$  values are stored in `domain` and the corresponding execution times for `matrix_vector_product()` and `matrix_matrix_product()` are stored in `vector_times` and `matrix_times`, respectively. Then the following code produces Figure A.5.

```
>>> ax1 = plt.subplot(121) # Plot both curves on a regular lin-lin plot.
>>> ax1.plot(domain, vector_times, 'b.-', lw=2, ms=15, label="Matrix-<→
    Vector")
>>> ax1.plot(domain, matrix_times, 'g.-', lw=2, ms=15, label="Matrix-<→
    Matrix")
>>> ax1.legend(loc="upper left")

>>> ax2 = plt.subplot(122) # Plot both curves on a base 2 log-log plot.
>>> ax2.loglog(domain, vector_times, 'b.-', basex=2, basey=2, lw=2)
>>> ax2.loglog(domain, matrix_times, 'g.-', basex=2, basey=2, lw=2)

>>> plt.show()
```

**Figure A.5**

In the log-log plot, the slope of the `matrix_matrix_product()` line is about 3 and the slope of the `matrix_vector_product()` line is about 2. This reflects the fact that matrix-matrix multiplication (which uses 3 loops) is  $O(n^3)$ , while matrix-vector multiplication (which only has 2 loops) is only  $O(n^2)$ .

#### Exercise A.4: N

*mPy is built specifically for fast numerical computations. Repeat the experiment of Problem A, timing the following operations:*

- matrix-vector multiplication with `matrix_vector_product()`.
- matrix-matrix multiplication with `matrix_matrix_product()`.
- matrix-vector multiplication with `np.dot()` or `@`.
- matrix-matrix multiplication with `np.dot()` or `@`.

*Create a single figure with two subplots: one with all four sets of execution times on a regular linear scale, and one with all four sets of execution times on a log-log scale. Compare your results to Figure A.5.*

#### NOTE

Problem A shows that **matrix operations are significantly faster in NumPy than in plain Python**. Matrix-matrix multiplication grows cubically regardless of the implementation; however, with lists the times grows at a rate of  $an^3$  while with NumPy the times grow at a rate of  $bn^3$ , where  $a$  is much larger than  $b$ . NumPy is more efficient for several reasons:

- Iterating through loops is very expensive. Many of NumPy's operations are implemented in C, which are much faster than Python loops.
- Arrays are designed specifically for matrix operations, while Python lists are general purpose.
- NumPy carefully takes advantage of computer hardware, efficiently using different levels of computer memory.

However, in Problem A, the execution times for matrix multiplication with NumPy seem to increase somewhat inconsistently. This is because the fastest layer of computer memory can only handle so much information before the computer has to begin using a larger, slower layer of memory.



## B. Linear Systems

---

**Lab Objective:** *The fundamental problem of linear algebra is solving the linear system  $\mathbf{Ax} = \mathbf{b}$ , given that a solution exists. There are many approaches to solving this problem, each with different pros and cons. In this lab we implement the LU decomposition and use it to solve square linear systems. We also introduce SciPy, together with its libraries for linear algebra and working with sparse matrices.*

### Gaussian Elimination

---

The standard approach for solving the linear system  $\mathbf{Ax} = \mathbf{b}$  on paper is reducing the augmented matrix  $[\mathbf{A} | \mathbf{b}]$  to row-echelon form (REF) via *Gaussian elimination*, then using back substitution. The matrix is in REF when the leading non-zero term in each row is the diagonal term, so the matrix is upper triangular.

At each step of Gaussian elimination, there are three possible operations: swapping two rows, multiplying one row by a scalar value, or adding a scalar multiple of one row to another. Many systems, like the one displayed below, can be reduced to REF using only the third type of operation. First, use multiples of the first row to get zeros below the diagonal in the first column, then use a multiple of the second row to get zeros below the diagonal in the second column.

$$\left[ \begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 4 & 2 & 3 \\ 4 & 7 & 8 & 9 \end{array} \right] \longrightarrow \left[ \begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ \textcolor{red}{0} & 3 & 1 & 2 \\ 4 & 7 & 8 & 9 \end{array} \right] \longrightarrow \left[ \begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ \textcolor{red}{0} & 3 & 4 & 5 \end{array} \right] \longrightarrow \left[ \begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & \textcolor{red}{0} & 3 & 3 \end{array} \right]$$

Each of these operations is equivalent to left-multiplying by a *type III elementary matrix*, the identity with a single non-zero non-diagonal term. If row operation  $k$  corresponds to matrix  $E_k$ , the following equation is  $E_3E_2E_1\mathbf{A} = \mathbf{U}$ .

$$\left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{array} \right] \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{array} \right] \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right] \left[ \begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 4 & 2 & 3 \\ 4 & 7 & 8 & 9 \end{array} \right] = \left[ \begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 0 & 3 & 3 \end{array} \right]$$

However, matrix multiplication is an inefficient way to implement row reduction. Instead, modify the matrix in place (without making a copy), changing only those entries that are affected by each row operation.

```
>>> import numpy as np
```

```

>>> A = np.array([[1, 1, 1, 1],
...                 [1, 4, 2, 3],
...                 [4, 7, 8, 9]], dtype=np.float)

# Reduce the 0th column to zeros below the diagonal.
>>> A[1,0:] -= (A[1,0] / A[0,0]) * A[0]
>>> A[2,0:] -= (A[2,0] / A[0,0]) * A[0]

# Reduce the 1st column to zeros below the diagonal.
>>> A[2,1:] -= (A[2,1] / A[1,1]) * A[1,1:]
>>> print(A)
[[ 1.  1.  1.  1.]
 [ 0.  3.  1.  2.]
 [ 0.  0.  3.  3.]]

```

Note that the final row operation modifies only part of the third row to avoid spending the computation time of adding 0 to 0.

If a 0 appears on the main diagonal during any part of row reduction, the approach given above tries to divide by 0. Swapping the current row with one below it that does not have a 0 in the same column solves this problem. This is equivalent to left-multiplying by a type II elementary matrix, also called a *permutation matrix*.

### ACHTUNG!

Gaussian elimination is not always numerically stable. In other words, it is susceptible to rounding error that may result in an incorrect final matrix. Suppose that, due to roundoff error, the matrix  $A$  has a very small entry on the diagonal.

$$A = \begin{bmatrix} 10^{-15} & 1 \\ -1 & 0 \end{bmatrix}$$

Though  $10^{-15}$  is essentially zero, instead of swapping the first and second rows to put  $A$  in REF, a computer might multiply the first row by  $10^{15}$  and add it to the second row to eliminate the  $-1$ . The resulting matrix is far from what it would be if the  $10^{-15}$  were actually 0.

$$\begin{bmatrix} 10^{-15} & 1 \\ -1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 10^{-15} & 1 \\ 0 & 10^{15} \end{bmatrix}$$

Round-off error can propagate through many steps in a calculation. The NumPy routines that employ row reduction use several tricks to minimize the impact of round-off error, but these tricks cannot fix every matrix.

### Problem B.1: Program simple row reduction to REF.

rite a function that reduces an arbitrary square matrix  $A$  to REF. You may assume that  $A$  is invertible and that a 0 will never appear on the main diagonal (so only use type III row reductions, not type II). Avoid operating on entries that you know will be 0 before and after a row operation. Use at most two nested loops.

Test your function with small test cases that you can check by hand. Consider using `np.random.randint()` to generate a few manageable tests cases.

## The LU Decomposition

The *LU decomposition* of a square matrix  $A$  is a factorization  $A = LU$  where  $U$  is the **upper** triangular REF of  $A$  and  $L$  is the **lower** triangular product of the type III elementary matrices whose inverses reduce  $A$  to  $U$ . The LU decomposition of  $A$  exists when  $A$  can be reduced to REF using only type III elementary matrices (without any row swaps). However, the rows of  $A$  can always be permuted in a way such that the decomposition exists. If  $P$  is a permutation matrix encoding the appropriate row swaps, then the decomposition  $PA = LU$  always exists.

Suppose  $A$  has an LU decomposition (not requiring row swaps). Then  $A$  can be reduced to REF with  $k$  row operations, corresponding to left-multiplying the type III elementary matrices  $E_1, \dots, E_k$ . Because there were no row swaps, each  $E_i$  is lower triangular, so each inverse  $E_i^{-1}$  is also lower triangular. Furthermore, since the product of lower triangular matrices is lower triangular,  $L$  is lower triangular:

$$\begin{aligned} E_k \dots E_2 E_1 A &= U \quad \longrightarrow \quad A = (E_k \dots E_2 E_1)^{-1} U \\ &= E_1^{-1} E_2^{-1} \dots E_k^{-1} U \\ &= LU. \end{aligned}$$

Thus,  $L$  can be computed by right-multiplying the identity by the matrices used to reduce  $U$ . However, in this special situation, each right-multiplication only changes one entry of  $L$ , matrix multiplication can be avoided altogether. The entire process, only slightly different than row reduction, is summarized below.

---

#### Algorithm 7

```

1: procedure LU DECOMPOSITION( $A$ )
2:    $m, n \leftarrow \text{shape}(A)$                                  $\triangleright$  Store the dimensions of  $A$ .
3:    $U \leftarrow \text{copy}(A)$                                   $\triangleright$  Make a copy of  $A$  with np.copy().
4:    $L \leftarrow I_m$                                           $\triangleright$  The  $m \times m$  identity matrix.
5:   for  $j = 0 \dots n - 1$  do
6:     for  $i = j + 1 \dots m - 1$  do
7:        $L_{i,j} \leftarrow U_{i,j} / U_{j,j}$ 
8:        $U_{i,j:} \leftarrow U_{i,j:} - L_{i,j} U_{j,j:}$ 
9:   return  $L, U$ 

```

---

**Problem B.2: LU Decomposition**

Write a function that finds the LU decomposition of a square matrix. You may assume that the decomposition exists and requires no row swaps.

**Forward and Backward Substitution**

If  $PA = LU$  and  $A\mathbf{x} = b$ , then  $LUX = PA\mathbf{x} = Pb$ . This system can be solved by first solving  $L\mathbf{y} = Pb$ , then  $U\mathbf{x} = \mathbf{y}$ . Since  $L$  and  $U$  are both triangular, these systems can be solved with backward and forward substitution. We can thus compute the LU factorization of  $A$  once, then use substitution to efficiently solve  $A\mathbf{x} = b$  for various values of  $b$ .

Since the diagonal entries of  $L$  are all 1, the triangular system  $L\mathbf{y} = b$  has the form

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}.$$

Matrix multiplication yields the equations

$$\begin{aligned} y_1 &= b_1, & y_1 &= b_1, \\ l_{21}y_1 + y_2 &= b_2, & y_2 &= b_2 - l_{21}y_1, \\ &\vdots &&\vdots \\ \sum_{j=1}^{k-1} l_{kj}y_j + y_k &= b_k, & y_k &= b_k - \sum_{j=1}^{k-1} l_{kj}y_j. \end{aligned} \tag{2.1}$$

The triangular system  $U\mathbf{x} = \mathbf{y}$  yields similar equations, but in reverse order:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix},$$

$$\begin{aligned} u_{nn}x_n &= y_n, & x_n &= \frac{1}{u_{nn}}y_n, \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1}, & x_{n-1} &= \frac{1}{u_{n-1,n-1}}(y_{n-1} - u_{n-1,n}x_n), \\ &\vdots &&\vdots \\ \sum_{j=k}^n u_{kj}x_j &= y_k, & x_k &= \frac{1}{u_{kk}} \left( y_k - \sum_{j=k+1}^n u_{kj}x_j \right). \end{aligned} \tag{2.2}$$

**Problem B.3: Program back and forward substitution.**

rite a function that, given  $A$  and  $b$ , solves the square linear system  $Ax = b$ . Use the function from Problem B to compute  $L$  and  $U$ , then use (2.1) and (2.2) to solve for  $y$ , then  $x$ . You may again assume that no row swaps are required ( $P = I$  in this case).

## SciPy

---

SciPy [scipy] is a powerful scientific computing library built upon NumPy. It includes high-level tools for linear algebra, statistics, signal processing, integration, optimization, machine learning, and more.

SciPy is typically imported with the convention `import scipy as sp`. However, SciPy is set up in a way that requires its submodules to be imported individually.<sup>1</sup>

```
>>> import scipy as sp
>>> hasattr(sp, "stats")           # The stats module isn't loaded yet.
False

>>> from scipy import stats        # Import stats explicitly. Access it
>>> hasattr(sp, "stats")          # with 'stats' or 'sp.stats'.
True
```

## Linear Algebra

---

NumPy and SciPy both have a linear algebra module, each called `linalg`, but SciPy's module is the larger of the two. Some of SciPy's common `linalg` functions are listed below.

Function	Returns
<code>det()</code>	The determinant of a square matrix.
<code>eig()</code>	The eigenvalues and eigenvectors of a square matrix.
<code>inv()</code>	The inverse of an invertible matrix.
<code>norm()</code>	The norm of a vector or matrix norm of a matrix.
<code>solve()</code>	The solution to $Ax = b$ (the system need not be square).

This library also includes routines for computing matrix decompositions.

```
>>> from scipy import linalg as la
```

<sup>1</sup>SciPy modules like `linalg` are really *packages*, which are not initialized when SciPy is imported alone.

```
# Make a random matrix and a random vector.
>>> A = np.random.random((1000,1000))
>>> b = np.random.random(1000)

# Compute the LU decomposition of A, including pivots.
>>> L, P = la.lu_factor(A)

# Use the LU decomposition to solve Ax = b.
>>> x = la.lu_solve((L,P), b)

# Check that the solution is legitimate.
>>> np.allclose(A @ x, b)
True
```

As with NumPy, SciPy's routines are all highly optimized. However, some algorithms are, by nature, faster than others.

#### Problem B.4: Time ways to solve $Ax = b$ with `scipy.linalg`.

Write a function that times different `scipy.linalg` functions for solving square linear systems. For various values of  $n$ , generate a random  $n \times n$  matrix  $A$  and a random  $n$ -vector  $b$  using `np.random.random()`. Time how long it takes to solve the system  $Ax = b$  with each of the following approaches:

- Invert  $A$  with `la.inv()` and left-multiply the inverse to  $b$ .
- Use `la.solve()`.
- Use `la.lu_factor()` and `la.lu_solve()` to solve the system with the LU decomposition.
- Use `la.lu_factor()` and `la.lu_solve()`, but only time `la.lu_solve()` (not the time it takes to do the factorization with `la.lu_factor()`).

Plot the system size  $n$  versus the execution times. Use log scales if needed.

#### ACHTUNG!

Problem B demonstrates that computing a matrix inverse is computationally expensive. In fact, numerically inverting matrices is so costly that there is hardly ever a good reason to do it. Use a specific solver like `la.lu_solve()` whenever possible instead of using `la.inv()`.

## Sparse Matrices

---

Large linear systems can have tens of thousands of entries. Storing the corresponding matrices in memory can be difficult: a  $10^5 \times 10^5$  system requires around 40 GB to store in a NumPy array (4 bytes per entry  $\times 10^{10}$  entries). This is well beyond the amount of RAM in a normal laptop.

In applications where systems of this size arise, it is often the case that the system is *sparse*, meaning that most of the entries of the matrix are 0. SciPy's `sparse` module provides tools for efficiently constructing and manipulating 1- and 2-D sparse matrices. A sparse matrix only stores the nonzero values and the positions of these values. For sufficiently sparse matrices, storing the matrix as a sparse matrix may only take megabytes, rather than gigabytes.

For example, diagonal matrices are sparse. Storing an  $n \times n$  diagonal matrix in the naïve way means storing  $n^2$  values in memory. It is more efficient to instead store the diagonal entries in a 1-D array of  $n$  values. In addition to using less storage space, this allows for much faster matrix operations: the standard algorithm to multiply a matrix by a diagonal matrix involves  $n^3$  steps, but most of these are multiplying by or adding 0. A smarter algorithm can accomplish the same task much faster.

SciPy has seven sparse matrix types. Each type is optimized either for storing sparse matrices whose nonzero entries follow certain patterns, or for performing certain computations.

Name	Description	Advantages
<code>bsr_matrix</code>	Block Sparse Row	Specialized structure.
<code>coo_matrix</code>	Coordinate Format	Conversion among sparse formats.
<code>csc_matrix</code>	Compressed Sparse Column	Column-based operations and slicing.
<code>csr_matrix</code>	Compressed Sparse Row	Row-based operations and slicing.
<code>dia_matrix</code>	Diagonal Storage	Specialized structure.
<code>dok_matrix</code>	Dictionary of Keys	Element access, incremental construction.
<code>lil_matrix</code>	Row-based Linked List	Incremental construction.

### Creating Sparse Matrices

---

A regular, non-sparse matrix is called *full* or *dense*. Full matrices can be converted to each of the sparse matrix formats listed above. However, it is more memory efficient to never create the full matrix in the first place. There are three main approaches for creating sparse matrices from scratch.

- **Coordinate Format:** When all of the nonzero values and their positions are known, create the entire sparse matrix at once as a `coo_matrix`. All nonzero values are stored as a coordinate and a value. This format also converts quickly to other sparse matrix types.

```
>>> from scipy import sparse

# Define the rows, columns, and values separately.
>>> rows = np.array([0, 1, 0])
>>> cols = np.array([0, 1, 1])
>>> vals = np.array([3, 5, 2])
```

```

>>> A = sparse.coo_matrix((vals, (rows,cols)), shape=(3,3))
>>> print(A)
 (0, 0)    3
 (1, 1)    5
 (0, 1)    2

# The toarray() method casts the sparse matrix as a NumPy array.
>>> print(A.toarray())                      # Note that this method forfeits
[[3 2 0]                         # all sparsity-related ↵
     optimizations.
 [0 5 0]
 [0 0 0]]

```

- **DOK and LIL Formats:** If the matrix values and their locations are not known beforehand, construct the matrix incrementally with a `dok_matrix` or a `lil_matrix`. Indicate the size of the matrix, then change individual values with regular slicing syntax.

```

>>> B = sparse.lil_matrix((2,6))
>>> B[0,2] = 4
>>> B[1,3:] = 9

>>> print(B.toarray())
[[ 0.  0.  4.  0.  0.  0.]
 [ 0.  0.  0.  9.  9.  9.]]

```

- **DIA Format:** Use a `dia_matrix` to store matrices that have nonzero entries on only certain diagonals. The function `sparse.diags()` is one convenient way to create a `dia_matrix` from scratch. Additionally, every sparse matrix has a `setdiags()` method for modifying specified diagonals.

```

# Use sparse.diags() to create a matrix with diagonal entries.
>>> diagonals = [[1,2],[3,4,5],[6]]      # List the diagonal entries.
>>> offsets = [-1,0,3]                   # Specify the diagonal they ↵
                                         go on.
>>> print(sparse.diags(diagonals, offsets, shape=(3,4)).toarray())
[[ 3.  0.  0.  6.]
 [ 1.  4.  0.  0.]
 [ 0.  2.  5.  0.]]

# If all of the diagonals have the same entry, specify the entry ↵
                                         alone.
>>> A = sparse.diags([1,3,6], offsets, shape=(3,4))
>>> print(A.toarray())
[[ 3.  0.  0.  6.]]

```

```
[ 1.  3.  0.  0.]
[ 0.  1.  3.  0.]]
```

```
# Modify a diagonal with the setdiag() method.
>>> A.setdiag([4,4,4], 0)
>>> print(A.toarray())
[[ 4.  0.  0.  6.]
 [ 1.  4.  0.  0.]
 [ 0.  1.  4.  0.]]
```

- **BSR Format:** Many sparse matrices can be formulated as block matrices, and a block matrix can be stored efficiently as a `bsr_matrix`. Use `sparse.bmat()` or `sparse.block_diag()` to create a block matrix quickly.

```
# Use sparse.bmat() to create a block matrix. Use 'None' for zero ←
blocks.
>>> A = sparse.coo_matrix(np.ones((2,2)))
>>> B = sparse.coo_matrix(np.full((2,2), 2.))
>>> print(sparse.bmat([[ A , None,  A ],
                      [None,  B , None]], format='bsr').toarray())
[[ 1.  1.  0.  0.  1.  1.]
 [ 1.  1.  0.  0.  1.  1.]
 [ 0.  0.  2.  2.  0.  0.]
 [ 0.  0.  2.  2.  0.  0.]]
```

```
# Use sparse.block_diag() to construct a block diagonal matrix.
>>> print(sparse.block_diag((A,B)).toarray())
[[ 1.  1.  0.  0.]
 [ 1.  1.  0.  0.]
 [ 0.  0.  2.  2.]
 [ 0.  0.  2.  2.]]
```

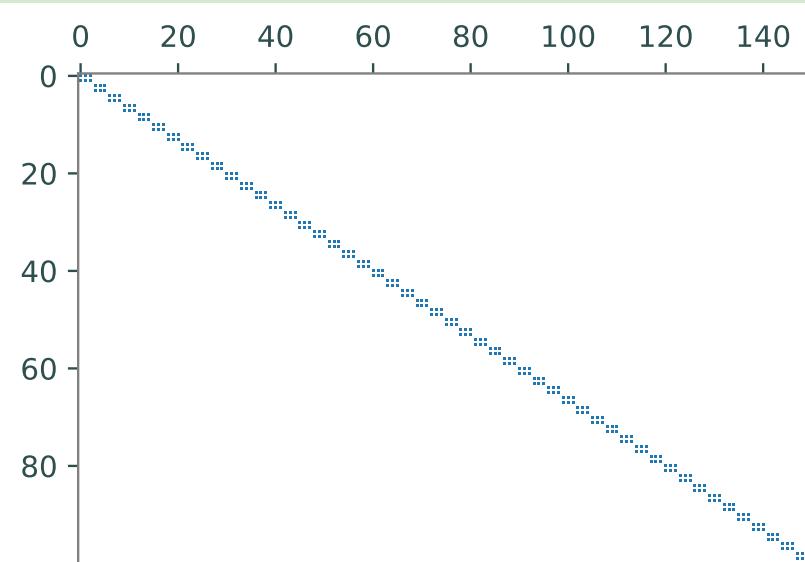
### NOTE

If a sparse matrix is too large to fit in memory as an array, it can still be visualized with Matplotlib's `plt.spy()`, which colors in the locations of the non-zero entries of the matrix.

```
>>> from matplotlib import pyplot as plt

# Construct and show a matrix with 50 2x3 diagonal blocks.
>>> B = sparse.coo_matrix([[1,3,5],[7,9,11]])
>>> A = sparse.block_diag([B]*50)
```

```
>>> plt.spy(A, markersize=1)
>>> plt.show()
```



**Problem B.5: Construct a large sparse matrix.**

Let  $I$  be the  $n \times n$  identity matrix, and define

$$A = \begin{bmatrix} B & I & & \\ I & B & I & \\ & I & \ddots & \ddots & \\ & & \ddots & \ddots & I \\ & & & I & B \end{bmatrix}, \quad B = \begin{bmatrix} -4 & 1 & & \\ 1 & -4 & 1 & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -4 \end{bmatrix},$$

where  $A$  is  $n^2 \times n^2$  and each block  $B$  is  $n \times n$ . The large matrix  $A$  is used in finite difference methods for solving Laplace's equation in two dimensions,  $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$ .

Write a function that accepts an integer  $n$  and constructs and returns  $A$  as a sparse matrix. Use `plt.spy()` to check that your matrix has nonzero values in the correct places.

## Sparse Matrix Operations

Once a sparse matrix has been constructed, it should be converted to a `csr_matrix` or a `csc_matrix` with the matrix's `tocsr()` or `tocsc()` method. The CSR and CSC formats are optimized for row or column operations, respectively. To choose the correct format to use, determine what direction the matrix will be traversed.

For example, in the matrix-matrix multiplication  $AB$ ,  $A$  is traversed row-wise, but  $B$  is traversed column-wise. Thus  $A$  should be converted to a `csr_matrix` and  $B$  should be converted to a `csc_matrix`.

```
# Initialize a sparse matrix incrementally as a lil_matrix.
>>> A = sparse.lil_matrix((10000,10000))
>>> for k in range(10000):
...     A[np.random.randint(0,9999), np.random.randint(0,9999)] = k
...
>>> A
<10000x10000 sparse matrix of type '<type 'numpy.float64'>' with 9999 stored elements in LInked List format>

# Convert A to CSR and CSC formats to compute the matrix product AA.
>>> Acsr = A.tocsr()
>>> Acsc = A.tocsc()
>>> Acsr.dot(Acsc)
<10000x10000 sparse matrix of type '<type 'numpy.float64'>' with 10142 stored elements in Compressed Sparse Row format>
```

Beware that row-based operations on a `csc_matrix` are very slow, and similarly, column-based operations on a `csr_matrix` are very slow.

### ACHTUNG!

Many familiar NumPy operations have analogous routines in the `sparse` module. These methods take advantage of the sparse structure of the matrices and are, therefore, usually significantly faster. However, SciPy's sparse matrices behave a little differently than NumPy arrays.

Operation	<code>numpy</code>	<code>scipy.sparse</code>
Component-wise Addition	<code>A + B</code>	<code>A + B</code>
Scalar Multiplication	<code>2 * A</code>	<code>2 * A</code>
Component-wise Multiplication	<code>A * B</code>	<code>A.multiply(B)</code>
Matrix Multiplication	<code>A.dot(B), A @ B</code>	<code>A * B, A.dot(B), A @ B</code>

Note in particular the difference between `A * B` for NumPy arrays and SciPy sparse matrices.

**Do not** use `np.dot()` to try to multiply sparse matrices, as it may treat the inputs incorrectly. The syntax `A.dot(B)` is safest in most cases.

SciPy's sparse module has its own linear algebra library, `scipy.sparse.linalg`, designed for operating on sparse matrices. Like other SciPy modules, it must be imported explicitly.

```
>>> from scipy.sparse import linalg as spla
```

### Problem B.6: Time `scipy.sparse.linalg.spsolve()` against `sp.linalg.solve()`.

Write a function that times regular and sparse linear system solvers.

For various values of  $n$ , generate the  $n^2 \times n^2$  matrix  $A$  described in Problem B and a random vector  $b$  with  $n^2$  entries. Time how long it takes to solve the system  $Ax = b$  with each of the following approaches:

- (a) Convert  $A$  to CSR format and use `scipy.sparse.linalg.spsolve()` (`spla.spsolve()`).
- (b) Convert  $A$  to a NumPy array and use `scipy.linalg.solve()` (`la.solve()`).

In each experiment, only time how long it takes to solve the system (not how long it takes to convert  $A$  to the appropriate format).

Plot the system size  $n^2$  versus the execution times. As always, use log scales where appropriate and use a legend to label each line.

### ACHTUNG!

Even though there are fast algorithms for solving certain sparse linear systems, it is still very computationally difficult to invert sparse matrices. In fact, the inverse of a sparse matrix is usually not sparse. There is rarely a good reason to invert a matrix, sparse or dense.

See <http://docs.scipy.org/doc/scipy/reference/sparse.html> for additional details on SciPy's sparse module.

# Additional Material

---

## Improvements on the LU Decomposition

---

### Vectorization

---

Algorithm 7 uses two loops to compute the LU decomposition. With a little vectorization, the process can be reduced to a single loop.

---

#### Algorithm 8

---

```

1: procedure FAST LU DECOMPOSITION( $A$ )
2:    $m, n \leftarrow \text{shape}(A)$ 
3:    $U \leftarrow \text{copy}(A)$ 
4:    $L \leftarrow I_m$ 
5:   for  $k = 0 \dots n - 1$  do
6:      $L_{k+1:,k} \leftarrow U_{k+1:,k} / U_{k,k}$ 
7:      $U_{k+1:,k:} \leftarrow U_{k+1:,k:} - L_{k+1:,k} U_{k,k}^T$ 
8:   return  $L, U$ 
```

---

Note that step 7 is an *outer product*, not the regular dot product ( $\mathbf{x}\mathbf{y}^T$  instead of the usual  $\mathbf{x}^T\mathbf{y}$ ). Use `np.outer()` instead of `np.dot()` or `@` to get the desired result.

### Pivoting

---

Gaussian elimination iterates through the rows of a matrix, using the diagonal entry  $x_{k,k}$  of the matrix at the  $k$ th iteration to zero out all of the entries in the column below  $x_{k,k}$  ( $x_{i,k}$  for  $i \geq k$ ). This diagonal entry is called the *pivot*. Unfortunately, Gaussian elimination, and hence the LU decomposition, can be very numerically unstable if at any step the pivot is a very small number. Most professional row reduction algorithms avoid this problem via *partial pivoting*.

The idea is to choose the largest number (in magnitude) possible to be the pivot by swapping the pivot row<sup>2</sup> with another row before operating on the matrix. For example, the second and fourth rows of the following matrix are exchanged so that the pivot is  $-6$  instead of  $2$ .

$$\begin{bmatrix} \times & \times & \times & \times \\ 0 & 2 & \times & \times \\ 0 & 4 & \times & \times \\ 0 & -6 & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & \times & \times & \times \\ 0 & -6 & \times & \times \\ 0 & 4 & \times & \times \\ 0 & 2 & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & \times & \times & \times \\ 0 & -6 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}$$

A row swap is equivalent to left-multiplying by a type II elementary matrix, also called a *permutation matrix*.

---

<sup>2</sup>Complete pivoting involves row and column swaps, but doing both operations is usually considered overkill.

*tion matrix.*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times \\ 0 & 2 & \times & \times \\ 0 & 4 & \times & \times \\ 0 & -6 & \times & \times \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \times \\ 0 & -6 & \times & \times \\ 0 & 4 & \times & \times \\ 0 & 2 & \times & \times \end{bmatrix}$$

For the LU decomposition, if the permutation matrix at step  $k$  is  $P_k$ , then  $P = P_k \dots P_2 P_1$  yields  $PA = LU$ . The complete algorithm is given below.

---

### Algorithm 9

---

```

1: procedure LU DECOMPOSITION WITH PARTIAL PIVOTING( $A$ )
2:    $m, n \leftarrow \text{shape}(A)$ 
3:    $U \leftarrow \text{copy}(A)$ 
4:    $L \leftarrow I_m$ 
5:    $P \leftarrow [0, 1, \dots, n-1]$                                  $\triangleright$  See tip 4b below.
6:   for  $k = 0 \dots n-1$  do
7:     Select  $i \geq k$  that maximizes  $|U_{i,k}|$ 
8:      $U_{k,k} : \leftrightarrow U_{i,k}$                                  $\triangleright$  Swap the two rows.
9:      $L_{k,:k} \leftrightarrow L_{i,:k}$                                  $\triangleright$  Swap the two rows.
10:     $P_k \leftrightarrow P_i$                                  $\triangleright$  Swap the two entries.
11:     $L_{k+1:,k} \leftarrow U_{k+1:,k}/U_{k,k}$ 
12:     $U_{k+1:,k} \leftarrow U_{k+1:,k} - L_{k+1:,k}U_{k,k}^T$ 
13:   return  $L, U, P$ 

```

---

The following tips may be helpful for implementing this algorithm:

- (a) Since NumPy arrays are mutable, use `np.copy()` to reassign the rows of an array simultaneously.
- (b) Instead of storing  $P$  as an  $n \times n$  array, fancy indexing allows us to encode row swaps in a 1-D array of length  $n$ . Initialize  $P$  as the array  $[0, 1, \dots, n]$ . After performing a row swap on  $A$ , perform the same operations on  $P$ . Then the matrix product  $PA$  will be the same as  $A[P]$ .

```

>>> A = np.zeros(3) + np.vstack(np.arange(3))
>>> P = np.arange(3)
>>> print(A)
[[ 0.  0.  0.]
 [ 1.  1.  1.]
 [ 2.  2.  2.]]

# Swap rows 1 and 2.
>>> A[1], A[2] = np.copy(A[2]), np.copy(A[1])
>>> P[1], P[2] = P[2], P[1]
>>> print(A)                                              # A with the new row arrangement.
[[ 0.  0.  0.]
 [ 2.  2.  2.]
 [ 1.  1.  1.]]

```

```
[ 2.  2.  2.]
[ 1.  1.  1.]]
>>> print(P)                                # The permutation of the rows.
[0 2 1]
>>> print(A[P])                            # A with the original row arrangement←
.
[[ 0.  0.  0.]
 [ 1.  1.  1.]
 [ 2.  2.  2.]]
```

There are potential cases where even partial pivoting does not eliminate catastrophic numerical errors in Gaussian elimination, but the odds of having such an amazingly poor matrix are essentially zero. The numerical analyst J.H. Wilkinson captured the likelihood of encountering such a matrix in a natural application when he said, “Anyone that unlucky has already been run over by a bus!”

### In Place

---

The LU decomposition can be performed in place (overwriting the original matrix  $A$ ) by storing  $U$  on and above the main diagonal of the array and storing  $L$  below it. The main diagonal of  $L$  does not need to be stored since all of its entries are 1. This format saves an entire array of memory, and is how `scipy.linalg.lu_factor()` returns the factorization.

## More Applications of the LU Decomposition

---

The LU decomposition can also be used to compute inverses and determinants with relative efficiency.

- **Inverse:**  $(PA)^{-1} = (LU)^{-1} \implies A^{-1}P^{-1} = U^{-1}L^{-1} \implies LUA^{-1} = P$ . Solve  $LUA_i = \mathbf{p}_i$  with forward and backward substitution (as in Problem B) for every column  $\mathbf{p}_i$  of  $P$ . Then

$$A^{-1} = \left[ \begin{array}{c|c|c|c} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{array} \right],$$

the matrix where  $\mathbf{a}_k$  is the  $k$ th column.

- **Determinant:**  $\det(A) = \det(P^{-1}LU) = \frac{\det(L)\det(U)}{\det(P)}$ . The determinant of a triangular matrix is the product of its diagonal entries. Since every diagonal entry of  $L$  is 1,  $\det(L) = 1$ . Also,  $P$  is just a row permutation of the identity matrix (which has determinant 1), and a single row swap negates the determinant. Then if  $S$  is the number of row swaps, the determinant is

$$\det(A) = (-1)^S \prod_{i=1}^n u_{ii}.$$

## The Cholesky Decomposition

---

A square matrix  $A$  is called *positive definite* if  $\mathbf{z}^\top A \mathbf{z} > 0$  for all nonzero vectors  $\mathbf{z}$ . In addition,  $A$  is called *Hermitian* if  $A = A^\text{H} = \overline{A^\top}$ . If  $A$  is Hermitian positive definite, it has a *Cholesky Decomposition*  $A = U^\text{H}U$  where  $U$  is upper triangular with real, positive entries on the diagonal. This is the matrix equivalent to taking the square root of a positive real number.

The Cholesky decomposition takes advantage of the conjugate symmetry of  $A$  to simultaneously reduce the columns *and* rows of  $A$  to zeros (except for the diagonal). It thus requires only half of the calculations and memory of the LU decomposition. Furthermore, the algorithm is *numerically stable*, which means, roughly speaking, that round-off errors do not propagate throughout the computation.

---

### Algorithm 10

---

```

1: procedure CHOLESKY DECOMPOSITION( $A$ )
2:    $n, n \leftarrow \text{shape}(A)$ 
3:    $U \leftarrow \text{np.triu}(A)$                                  $\triangleright$  Get the upper-triangular part of  $A$ .
4:   for  $i = 0 \dots n - 1$  do
5:     for  $j = i + 1 \dots n - 1$  do
6:        $U_{j,j:} \leftarrow U_{j,j:} - U_{i,j:} \overline{U_{ij}} / U_{ii}$ 
7:        $U_{i,i:} \leftarrow U_{i,i:} / \sqrt{U_{ii}}$ 
8:   return  $U$ 

```

---

As with the LU decomposition, SciPy's `linalg` module has optimized routines, `la.cho_factor()` and `la.cho_solve()`, for using the Cholesky decomposition.

# C. Systems of Equations

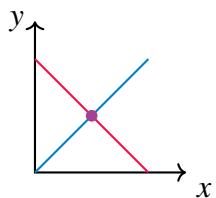
## C.1 Systems of Equations, Geometry

### Outcomes

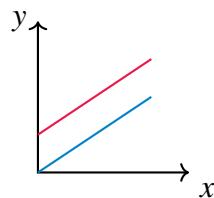
- A. Relate the types of solution sets of a system of two (three) variables to the intersections of lines in a plane (the intersection of planes in three space)

As you may remember, linear equations like  $2x + 3y = 6$  can be graphed as straight lines in the coordinate plane. We say that this equation is in two variables, in this case  $x$  and  $y$ . Suppose you have two such equations, each of which can be graphed as a straight line, and consider the resulting graph of two lines. What would it mean if there exists a point of intersection between the two lines? This point, which lies on *both* graphs, gives  $x$  and  $y$  values for which both equations are true. In other words, this point gives the ordered pair  $(x, y)$  that satisfy both equations. If the point  $(x, y)$  is a point of intersection, we say that  $(x, y)$  is a **solution** to the two equations. In linear algebra, we often are concerned with finding the solution(s) to a system of equations, if such solutions exist. First, we consider graphical representations of solutions and later we will consider the algebraic methods for finding solutions.

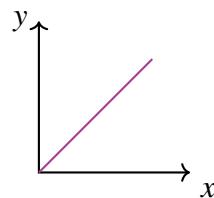
When looking for the intersection of two lines in a graph, several situations may arise. The following picture demonstrates the possible situations when considering two equations (two lines in the graph) involving two variables.



One Solution



No Solutions



Infinitely Many Solutions

In the first diagram, there is a unique point of intersection, which means that there is only one (unique) solution to the two equations. In the second, there are no points of intersection and no solution. When no solution exists, this means that the two lines are parallel and they never intersect. The third situation which can occur, as demonstrated in diagram three, is that the two lines are really the same line. For example,  $x + y = 1$  and  $2x + 2y = 2$  are equations which when graphed yield the same line. In this case there are infinitely many points which are solutions of these two equations, as every ordered pair which is on the graph of the line satisfies both equations. When considering

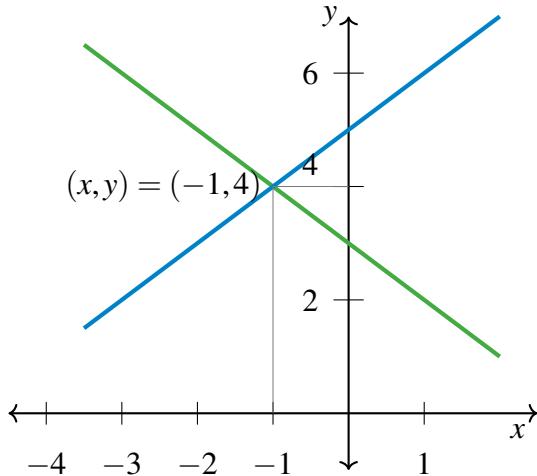
linear systems of equations, there are always three types of solutions possible; exactly one (unique) solution, infinitely many solutions, or no solution.

### Example C.1: A Graphical Solution

*Use a graph to find the solution to the following system of equations*

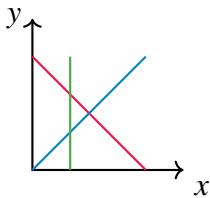
$$\begin{aligned}x + y &= 3 \\y - x &= 5\end{aligned}$$

**Solution.** Through graphing the above equations and identifying the point of intersection, we can find the solution(s). Remember that we must have either one solution, infinitely many, or no solutions at all. The following graph shows the two equations, as well as the intersection. Remember, the point of intersection represents the solution of the two equations, or the  $(x, y)$  which satisfy both equations. In this case, there is one point of intersection at  $(-1, 4)$  which means we have one unique solution,  $x = -1, y = 4$ .

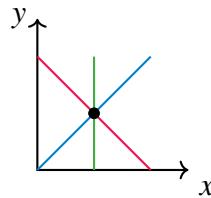


In the above example, we investigated the intersection point of two equations in two variables,  $x$  and  $y$ . Now we will consider the graphical solutions of three equations in two variables.

Consider a system of three equations in two variables. Again, these equations can be graphed as straight lines in the plane, so that the resulting graph contains three straight lines. Recall the three possible types of solutions; no solution, one solution, and infinitely many solutions. There are now more complex ways of achieving these situations, due to the presence of the third line. For example, you can imagine the case of three intersecting lines having no common point of intersection. Perhaps you can also imagine three intersecting lines which do intersect at a single point. These two situations are illustrated below.



No Solution

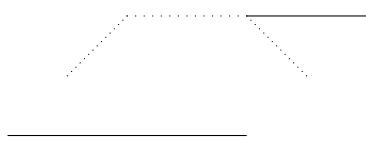


One Solution

Consider the first picture above. While all three lines intersect with one another, there is no common point of intersection where all three lines meet at one point. Hence, there is no solution to the three equations. Remember, a solution is a point  $(x, y)$  which satisfies **all** three equations. In the case of the second picture, the lines intersect at a common point. This means that there is one solution to the three equations whose graphs are the given lines. You should take a moment now to draw the graph of a system which results in three parallel lines. Next, try the graph of three identical lines. Which type of solution is represented in each of these graphs?

We have now considered the graphical solutions of systems of two equations in two variables, as well as three equations in two variables. However, there is no reason to limit our investigation to equations in two variables. We will now consider equations in three variables.

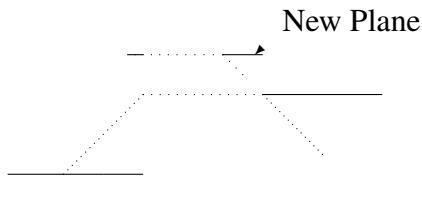
You may recall that equations in three variables, such as  $2x + 4y - 5z = 8$ , form a plane. Above, we were looking for intersections of lines in order to identify any possible solutions. When graphically solving systems of equations in three variables, we look for intersections of planes. These points of intersection give the  $(x, y, z)$  that satisfy all the equations in the system. What types of solutions are possible when working with three variables? Consider the following picture involving two planes, which are given by two equations in three variables.



Notice how these two planes intersect in a line. This means that the points  $(x, y, z)$  on this line satisfy both equations in the system. Since the line contains infinitely many points, this system has infinitely many solutions.

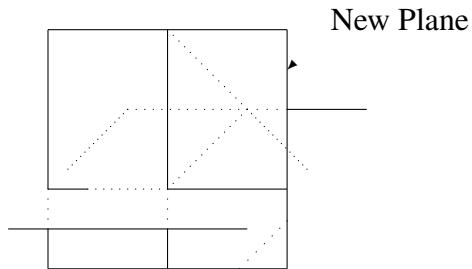
It could also happen that the two planes fail to intersect. However, is it possible to have two planes intersect at a single point? Take a moment to attempt drawing this situation, and convince yourself that it is not possible! This means that when we have only two equations in three variables, there is no way to have a unique solution! Hence, the types of solutions possible for two equations in three variables are no solution or infinitely many solutions.

Now imagine adding a third plane. In other words, consider three equations in three variables. What types of solutions are now possible? Consider the following diagram.

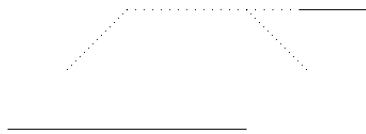


In this diagram, there is no point which lies in all three planes. There is no intersection between **all** planes so there is no solution. The picture illustrates the situation in which the line of intersection of the new plane with one of the original planes forms a line parallel to the line of intersection of the first two planes. However, in three dimensions, it is possible for two lines to fail to intersect even though they are not parallel. Such lines are called **skew lines**.

Recall that when working with two equations in three variables, it was not possible to have a unique solution. Is it possible when considering three equations in three variables? In fact, it is possible, and we demonstrate this situation in the following picture.



In this case, the three planes have a single point of intersection. Can you think of other types of solutions possible? Another is that the three planes could intersect in a line, resulting in infinitely many solutions, as in the following diagram.



We have now seen how three equations in three variables can have no solution, a unique solution, or intersect in a line resulting in infinitely many solutions. It is also possible that the three equations graph the same plane, which also leads to infinitely many solutions.

You can see that when working with equations in three variables, there are many more ways to achieve the different types of solutions than when working with two variables. It may prove enlightening to spend time imagining (and drawing) many possible scenarios, and you should take some time to try a few.

You should also take some time to imagine (and draw) graphs of systems in more than three variables. Equations like  $x + y - 2z + 4w = 8$  with more than three variables are often called **hyper-planes**. You may soon realize that it is tricky to draw the graphs of hyper-planes! Through the tools of linear algebra, we can algebraically examine these types of systems which are difficult to graph. In the following section, we will consider these algebraic tools.

## C.2 Systems Of Equations, Algebraic Procedures

### Outcomes

- A. Use elementary operations to find the solution to a linear system of equations.
- B. Find the row-echelon form and reduced row-echelon form of a matrix.
- C. Determine whether a system of linear equations has no solution, a unique solution or an infinite number of solutions from its row-echelon form.
- D. Solve a system of equations using Gaussian Elimination and Gauss-Jordan Elimination.
- E. Model a physical system with linear equations and then solve.

We have taken an in depth look at graphical representations of systems of equations, as well as how to find possible solutions graphically. Our attention now turns to working with systems algebraically.

### Definition C.2: System of Linear Equations

A **system of linear equations** is a list of equations,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

where  $a_{ij}$  and  $b_j$  are real numbers. The above is a system of  $m$  equations in the  $n$  variables,  $x_1, x_2, \dots, x_n$ . Written more simply in terms of summation notation, the above can be written in the form

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad i = 1, 2, 3, \dots, m$$

The relative size of  $m$  and  $n$  is not important here. Notice that we have allowed  $a_{ij}$  and  $b_j$  to be any real number. We can also call these numbers **scalars**. We will use this term throughout the text, so keep in mind that the term **scalar** just means that we are working with real numbers.

Now, suppose we have a system where  $b_i = 0$  for all  $i$ . In other words every equation equals 0. This is a special type of system.

**Definition C.3: Homogeneous System of Equations**

A system of equations is called **homogeneous** if each equation in the system is equal to 0. A homogeneous system has the form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= 0 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= 0 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= 0 \end{aligned}$$

where  $a_{ij}$  are scalars and  $x_i$  are variables.

Recall from the previous section that our goal when working with systems of linear equations was to find the point of intersection of the equations when graphed. In other words, we looked for the solutions to the system. We now wish to find these solutions algebraically. We want to find values for  $x_1, \dots, x_n$  which solve all of the equations. If such a set of values exists, we call  $(x_1, \dots, x_n)$  the **solution set**.

Recall the above discussions about the types of solutions possible. We will see that systems of linear equations will have one unique solution, infinitely many solutions, or no solution. Consider the following definition.

**Definition C.4: Consistent and Inconsistent Systems**

A system of linear equations is called **consistent** if there exists at least one solution. It is called **inconsistent** if there is no solution.

If you think of each equation as a condition which must be satisfied by the variables, consistent would mean there is some choice of variables which can satisfy **all** the conditions. Inconsistent would mean there is no choice of the variables which can satisfy all of the conditions.

The following sections provide methods for determining if a system is consistent or inconsistent, and finding solutions if they exist.

### C.2.1. Elementary Operations

---

We begin this section with an example. Recall from Example C.1 that the solution to the given system was  $(x, y) = (-1, 4)$ .

**Example C.5: Verifying an Ordered Pair is a Solution**

Algebraically verify that  $(x, y) = (-1, 4)$  is a solution to the following system of equations.

$$\begin{aligned} x + y &= 3 \\ y - x &= 5 \end{aligned}$$

**Solution.** By graphing these two equations and identifying the point of intersection, we previously found that  $(x, y) = (-1, 4)$  is the unique solution.

We can verify algebraically by substituting these values into the original equations, and ensuring that the equations hold. First, we substitute the values into the first equation and check that it equals 3.

$$x + y = (-1) + (4) = 3$$

This equals 3 as needed, so we see that  $(-1, 4)$  is a solution to the first equation. Substituting the values into the second equation yields

$$y - x = (4) - (-1) = 4 + 1 = 5$$

which is true. For  $(x, y) = (-1, 4)$  each equation is true and therefore, this is a solution to the system. ♠

Now, the interesting question is this: If you were not given these numbers to verify, how could you algebraically determine the solution? Linear algebra gives us the tools needed to answer this question. The following basic operations are important tools that we will utilize.

### Definition C.6: Elementary Operations

**Elementary operations** are those operations consisting of the following.

- (a) Interchange the order in which the equations are listed.
- (b) Multiply any equation by a nonzero number.
- (c) Replace any equation with itself added to a multiple of another equation.

It is important to note that none of these operations will change the set of solutions of the system of equations. In fact, elementary operations are the *key tool* we use in linear algebra to find solutions to systems of equations.

Consider the following example.

### Example C.7: Effects of an Elementary Operation

Show that the system

$$\begin{aligned} x + y &= 7 \\ 2x - y &= 8 \end{aligned}$$

has the same solution as the system

$$\begin{aligned} x + y &= 7 \\ -3y &= -6 \end{aligned}$$

**Solution.** Notice that the second system has been obtained by taking the second equation of the first system and adding -2 times the first equation, as follows:

$$2x - y + (-2)(x + y) = 8 + (-2)(7)$$

By simplifying, we obtain

$$-3y = -6$$

which is the second equation in the second system. Now, from here we can solve for  $y$  and see that  $y = 2$ . Next, we substitute this value into the first equation as follows

$$x + y = x + 2 = 7$$

Hence  $x = 5$  and so  $(x, y) = (5, 2)$  is a solution to the second system. We want to check if  $(5, 2)$  is also a solution to the first system. We check this by substituting  $(x, y) = (5, 2)$  into the system and ensuring the equations are true.

$$\begin{aligned} x + y &= (5) + (2) = 7 \\ 2x - y &= 2(5) - (2) = 8 \end{aligned}$$

Hence,  $(5, 2)$  is also a solution to the first system. ♠

This example illustrates how an elementary operation applied to a system of two equations in two variables does not affect the solution set. However, a linear system may involve many equations and many variables and there is no reason to limit our study to small systems. For any size of system in any number of variables, the solution set is still the collection of solutions to the equations. In every case, the above operations of Definition C.6 do not change the set of solutions to the system of linear equations.

In the following theorem, we use the notation  $E_i$  to represent an equation, while  $b_i$  denotes a constant.

### Theorem C.8: Elementary Operations and Solutions

Suppose you have a system of two linear equations

$$\begin{aligned} E_1 &= b_1 \\ E_2 &= b_2 \end{aligned} \tag{3.1}$$

Then the following systems have the same solution set as 3.1:

(a)

$$\begin{aligned} E_2 &= b_2 \\ E_1 &= b_1 \end{aligned} \tag{3.2}$$

(b)

$$\begin{aligned} E_1 &= b_1 \\ kE_2 &= kb_2 \end{aligned} \tag{3.3}$$

for any scalar  $k$ , provided  $k \neq 0$ .

(c)

$$\begin{aligned} E_1 &= b_1 \\ E_2 + kE_1 &= b_2 + kb_1 \end{aligned} \tag{3.4}$$

for any scalar  $k$  (including  $k = 0$ ).

Before we proceed with the proof of Theorem C.8, let us consider this theorem in context of Example

C.7. Then,

$$\begin{aligned} E_1 &= x + y, \quad b_1 = 7 \\ E_2 &= 2x - y, \quad b_2 = 8 \end{aligned}$$

Recall the elementary operations that we used to modify the system in the solution to the example. First, we added  $(-2)$  times the first equation to the second equation. In terms of Theorem C.8, this action is given by

$$E_2 + (-2)E_1 = b_2 + (-2)b_1$$

or

$$2x - y + (-2)(x + y) = 8 + (-2)7$$

This gave us the second system in Example C.7, given by

$$\begin{aligned} E_1 &= b_1 \\ E_2 + (-2)E_1 &= b_2 + (-2)b_1 \end{aligned}$$

From this point, we were able to find the solution to the system. Theorem C.8 tells us that the solution we found is in fact a solution to the original system.

Stated simply, the above theorem shows that the elementary operations do not change the solution set of a system of equations.

We will now look at an example of a system of three equations and three variables. Similarly to the previous examples, the goal is to find values for  $x, y, z$  such that each of the given equations are satisfied when these values are substituted in.

### Example C.9: Solving a System of Equations with Elementary Operations

*Find the solutions to the system,*

$$\begin{aligned} x + 3y + 6z &= 25 \\ 2x + 7y + 14z &= 58 \\ 2y + 5z &= 19 \end{aligned} \tag{3.5}$$

**Solution.** We can relate this system to Theorem C.8 above. In this case, we have

$$\begin{aligned} E_1 &= x + 3y + 6z, \quad b_1 = 25 \\ E_2 &= 2x + 7y + 14z, \quad b_2 = 58 \\ E_3 &= 2y + 5z, \quad b_3 = 19 \end{aligned}$$

Theorem C.8 claims that if we do elementary operations on this system, we will not change the solution set. Therefore, we can solve this system using the elementary operations given in Definition C.6. First, replace the second equation by  $(-2)$  times the first equation added to the second. This yields the system

$$\begin{aligned} x + 3y + 6z &= 25 \\ y + 2z &= 8 \\ 2y + 5z &= 19 \end{aligned} \tag{3.6}$$

Now, replace the third equation with  $(-2)$  times the second added to the third. This yields the system

$$\begin{aligned}x + 3y + 6z &= 25 \\y + 2z &= 8 \\z &= 3\end{aligned}\tag{3.7}$$

At this point, we can easily find the solution. Simply take  $z = 3$  and substitute this back into the previous equation to solve for  $y$ , and similarly to solve for  $x$ .

$$\begin{aligned}x + 3y + 6(3) &= x + 3y + 18 = 25 \\y + 2(3) &= y + 6 = 8 \\z &= 3\end{aligned}$$

The second equation is now

$$y + 6 = 8$$

You can see from this equation that  $y = 2$ . Therefore, we can substitute this value into the first equation as follows:

$$x + 3(2) + 18 = 25$$

By simplifying this equation, we find that  $x = 1$ . Hence, the solution to this system is  $(x, y, z) = (1, 2, 3)$ . This process is called **back substitution**.

Alternatively, in 3.7 you could have continued as follows. Add  $(-2)$  times the third equation to the second and then add  $(-6)$  times the second to the first. This yields

$$\begin{aligned}x + 3y &= 7 \\y &= 2 \\z &= 3\end{aligned}$$

Now add  $(-3)$  times the second to the first. This yields

$$\begin{aligned}x &= 1 \\y &= 2 \\z &= 3\end{aligned}$$

a system which has the same solution set as the original system. This avoided back substitution and led to the same solution set. It is your decision which you prefer to use, as both methods lead to the correct solution,  $(x, y, z) = (1, 2, 3)$ .



### C.2.2. Gaussian Elimination

---

The work we did in the previous section will always find the solution to the system. In this section, we will explore a less cumbersome way to find the solutions. First, we will represent a linear system with an **augmented matrix**. A **matrix** is simply a rectangular array of numbers. The size or dimension of a matrix is defined as  $m \times n$  where  $m$  is the number of rows and  $n$  is the number of columns. In order to construct an augmented matrix from a linear system, we create a **coefficient**

**matrix** from the coefficients of the variables in the system, as well as a **constant matrix** from the constants. The coefficients from one equation of the system create one row of the augmented matrix. For example, consider the linear system in Example C.9

$$\begin{aligned}x + 3y + 6z &= 25 \\2x + 7y + 14z &= 58 \\2y + 5z &= 19\end{aligned}$$

This system can be written as an augmented matrix, as follows

$$\left[ \begin{array}{ccc|c} 1 & 3 & 6 & 25 \\ 2 & 7 & 14 & 58 \\ 0 & 2 & 5 & 19 \end{array} \right]$$

Notice that it has exactly the same information as the original system. Here it is understood that the first column contains the coefficients from  $x$  in each equation, in order,  $\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$ . Similarly, we create

a column from the coefficients on  $y$  in each equation,  $\begin{bmatrix} 3 \\ 7 \\ 2 \end{bmatrix}$  and a column from the coefficients on

$z$  in each equation,  $\begin{bmatrix} 6 \\ 14 \\ 5 \end{bmatrix}$ . For a system of more than three variables, we would continue in this way constructing a column for each variable. Similarly, for a system of less than three variables, we simply construct a column for each variable.

Finally, we construct a column from the constants of the equations,  $\begin{bmatrix} 25 \\ 58 \\ 19 \end{bmatrix}$ .

The rows of the augmented matrix correspond to the equations in the system. For example, the top row in the augmented matrix,  $[ 1 \ 3 \ 6 \ | \ 25 ]$  corresponds to the equation

$$x + 3y + 6z = 25.$$

Consider the following definition.

**Definition C.10: Augmented Matrix of a Linear System**

For a linear system of the form

$$a_{11}x_1 + \cdots + a_{1n}x_n = b_1$$

⋮

$$a_{m1}x_1 + \cdots + a_{mn}x_n = b_m$$

where the  $x_i$  are variables and the  $a_{ij}$  and  $b_i$  are constants, the augmented matrix of this system is given by

$$\left[ \begin{array}{ccc|c} a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{m1} & \cdots & a_{mn} & b_m \end{array} \right]$$

Now, consider elementary operations in the context of the augmented matrix. The elementary operations in Definition C.6 can be used on the rows just as we used them on equations previously. Changes to a system of equations in as a result of an elementary operation are equivalent to changes in the augmented matrix resulting from the corresponding row operation. Note that Theorem C.8 implies that any elementary row operations used on an augmented matrix will not change the solution to the corresponding system of equations. We now formally define elementary row operations. These are the *key tool* we will use to find solutions to systems of equations.

**Definition C.11: Elementary Row Operations**

The **elementary row operations** (also known as **row operations**) consist of the following

- (a) Switch two rows.
- (b) Multiply a row by a nonzero number.
- (c) Replace a row by any multiple of another row added to it.

Recall how we solved Example C.9. We can do the exact same steps as above, except now in the context of an augmented matrix and using row operations. The augmented matrix of this system is

$$\left[ \begin{array}{ccc|c} 1 & 3 & 6 & 25 \\ 2 & 7 & 14 & 58 \\ 0 & 2 & 5 & 19 \end{array} \right]$$

Thus the first step in solving the system given by 3.5 would be to take  $(-2)$  times the first row of the augmented matrix and add it to the second row,

$$\left[ \begin{array}{ccc|c} 1 & 3 & 6 & 25 \\ 0 & 1 & 2 & 8 \\ 0 & 2 & 5 & 19 \end{array} \right]$$

Note how this corresponds to 3.6. Next take  $(-2)$  times the second row and add to the third,

$$\left[ \begin{array}{ccc|c} 1 & 3 & 6 & 25 \\ 0 & 1 & 2 & 8 \\ 0 & 0 & 1 & 3 \end{array} \right]$$

This augmented matrix corresponds to the system

$$\begin{aligned}x + 3y + 6z &= 25 \\y + 2z &= 8 \\z &= 3\end{aligned}$$

which is the same as 3.7. By back substitution you obtain the solution  $x = 1$ ,  $y = 2$ , and  $z = 3$ .

Through a systematic procedure of row operations, we can simplify an augmented matrix and carry it to **row-echelon form** or **reduced row-echelon form**, which we define next. These forms are used to find the solutions of the system of equations corresponding to the augmented matrix.

In the following definitions, the term **leading entry** refers to the first nonzero entry of a row when scanning the row from left to right.

### Definition C.12: Row-Echelon Form

An augmented matrix is in **row-echelon form** if

- (a) All nonzero rows are above any rows of zeros.
- (b) Each leading entry of a row is in a column to the right of the leading entries of any row above it.
- (c) Each leading entry of a row is equal to 1.

We also consider another reduced form of the augmented matrix which has one further condition.

### Definition C.13: Reduced Row-Echelon Form

An augmented matrix is in **reduced row-echelon form** if

- (a) All nonzero rows are above any rows of zeros.
- (b) Each leading entry of a row is in a column to the right of the leading entries of any rows above it.
- (c) Each leading entry of a row is equal to 1.
- (d) All entries in a column above and below a leading entry are zero.

Notice that the first three conditions on a reduced row-echelon form matrix are the same as those for row-echelon form.

Hence, every reduced row-echelon form matrix is also in row-echelon form. The converse is not necessarily true; we cannot assume that every matrix in row-echelon form is also in reduced row-echelon form. However, it often happens that the row-echelon form is sufficient to provide information about the solution of a system.

The following examples describe matrices in these various forms. As an exercise, take the time to carefully verify that they are in the specified form.

**Example C.14: Not in Row-Echelon Form**

The following augmented matrices are not in row-echelon form (and therefore also not in reduced row-echelon form).

$$\left[ \begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array} \right], \left[ \begin{array}{cc|c} 1 & 2 & 3 \\ 2 & 4 & -6 \\ 4 & 0 & 7 \end{array} \right], \left[ \begin{array}{ccc|c} 0 & 2 & 3 & 3 \\ 1 & 5 & 0 & 2 \\ 7 & 5 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array} \right]$$

**Example C.15: Matrices in Row-Echelon Form**

The following augmented matrices are in row-echelon form, but not in reduced row-echelon form.

$$\left[ \begin{array}{ccccc|c} 1 & 0 & 6 & 5 & 8 & 2 \\ 0 & 0 & 1 & 2 & 7 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right], \left[ \begin{array}{ccc|c} 1 & 3 & 5 & 4 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right], \left[ \begin{array}{ccc|c} 1 & 0 & 6 & 0 \\ 0 & 1 & 4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

Notice that we could apply further row operations to these matrices to carry them to reduced row-echelon form. Take the time to try that on your own. Consider the following matrices, which are in reduced row-echelon form.

**Example C.16: Matrices in Reduced Row-Echelon Form**

The following augmented matrices are in reduced row-echelon form.

$$\left[ \begin{array}{ccccc|c} 1 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right], \left[ \begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right], \left[ \begin{array}{ccc|c} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 2 \end{array} \right]$$

One way in which the row-echelon form of a matrix is useful is in identifying the pivot positions and pivot columns of the matrix.

**Definition C.17: Pivot Position and Pivot Column**

A **pivot position** in a matrix is the location of a leading entry in the row-echelon form of a matrix. A **pivot column** is a column that contains a pivot position.

For example consider the following.

**Example C.18: Pivot Position**

Let

$$A = \left[ \begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 6 \\ 4 & 4 & 4 & 10 \end{array} \right]$$

Where are the pivot positions and pivot columns of the augmented matrix  $A$ ?

**Solution.** The row-echelon form of this matrix is

$$\left[ \begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & \frac{3}{2} \\ 0 & 0 & 0 & 0 \end{array} \right]$$

This is all we need in this example, but note that this matrix is not in reduced row-echelon form.

In order to identify the pivot positions in the original matrix, we look for the leading entries in the row-echelon form of the matrix. Here, the entry in the first row and first column, as well as the entry in the second row and second column are the leading entries. Hence, these locations are the pivot positions. We identify the pivot positions in the original matrix, as in the following:

$$\left[ \begin{array}{ccc|c} \boxed{1} & 2 & 3 & 4 \\ 3 & \boxed{2} & 1 & 6 \\ 4 & 4 & 4 & 10 \end{array} \right]$$

Thus the pivot columns in the matrix are the first two columns. ♠

The following is an algorithm for carrying a matrix to row-echelon form and reduced row-echelon form. You may wish to use this algorithm to carry the above matrix to row-echelon form or reduced row-echelon form yourself for practice.

Most often we will apply this algorithm to an augmented matrix in order to find the solution to a system of linear equations. However, we can use this algorithm to compute the reduced row-echelon form of any matrix which could be useful in other applications.

Consider the following example of Algorithm ??.

**Example C.19: Finding Row-Echelon Form and Reduced Row-Echelon Form of a Matrix**

Let

$$A = \left[ \begin{array}{ccc} 0 & -5 & -4 \\ 1 & 4 & 3 \\ 5 & 10 & 7 \end{array} \right]$$

Find the row-echelon form of  $A$ . Then complete the process until  $A$  is in reduced row-echelon form.

**Solution.** In working through this example, we will use the steps outlined in Algorithm ??.

---

### Reduced Row-Echelon Form Algorithm

This algorithm provides a method for using row operations to take a matrix to its reduced row-echelon form. We begin with the matrix in its original form.

- Starting from the left, find the first nonzero column. This is the first pivot column, and the position at the top of this column is the first pivot position. Switch rows if necessary to place a nonzero number in the first pivot position.
  - Use row operations to make the entries below the first pivot position (in the first pivot column) equal to zero.
  - Ignoring the row containing the first pivot position, repeat steps 1 and 2 with the remaining rows. Repeat the process until there are no more rows to modify.
  - Divide each nonzero row by the value of the leading entry, so that the leading entry becomes 1. The matrix will then be in row-echelon form.  
The following step will carry the matrix from row-echelon form to reduced row-echelon form.
  - Moving from right to left, use row operations to create zeros in the entries of the pivot columns which are above the pivot positions. The result will be a matrix in reduced row-echelon form.
- 

- The first pivot column is the first column of the matrix, as this is the first nonzero column from the left. Hence the first pivot position is the one in the first row and first column. Switch the first two rows to obtain a nonzero entry in the first pivot position, outlined in a box below.

$$\left[ \begin{array}{ccc} 1 & 4 & 3 \\ 0 & -5 & -4 \\ 5 & 10 & 7 \end{array} \right]$$

- Step two involves creating zeros in the entries below the first pivot position. The first entry of the second row is already a zero. All we need to do is subtract 5 times the first row from the third row. The resulting matrix is

$$\left[ \begin{array}{ccc} 1 & 4 & 3 \\ 0 & -5 & -4 \\ 0 & 10 & 8 \end{array} \right]$$

- Now ignore the top row. Apply steps 1 and 2 to the smaller matrix

$$\left[ \begin{array}{cc} -5 & -4 \\ 10 & 8 \end{array} \right]$$

In this matrix, the first column is a pivot column, and  $-5$  is in the first pivot position. Therefore, we need to create a zero below it. To do this, add 2 times the first row (of this matrix) to the second. The resulting matrix is

$$\left[ \begin{array}{cc} -5 & -4 \\ 0 & 0 \end{array} \right]$$

Our original matrix now looks like

$$\left[ \begin{array}{ccc} 1 & 4 & 3 \\ 0 & -5 & -4 \\ 0 & 0 & 0 \end{array} \right]$$

We can see that there are no more rows to modify.

- (d) Now, we need to create leading 1s in each row. The first row already has a leading 1 so no work is needed here. Divide the second row by  $-5$  to create a leading 1. The resulting matrix is

$$\left[ \begin{array}{ccc} 1 & 4 & 3 \\ 0 & 1 & \frac{4}{5} \\ 0 & 0 & 0 \end{array} \right]$$

This matrix is now in row-echelon form.

- (e) Now create zeros in the entries above pivot positions in each column, in order to carry this matrix all the way to reduced row-echelon form. Notice that there is no pivot position in the third column so we do not need to create any zeros in this column! The column in which we need to create zeros is the second. To do so, subtract 4 times the second row from the first row. The resulting matrix is

$$\left[ \begin{array}{ccc} 1 & 0 & -\frac{1}{5} \\ 0 & 1 & \frac{4}{5} \\ 0 & 0 & 0 \end{array} \right]$$

This matrix is now in reduced row-echelon form. ♠

The above algorithm gives you a simple way to obtain the row-echelon form and reduced row-echelon form of a matrix. The main idea is to do row operations in such a way as to end up with a matrix in row-echelon form or reduced row-echelon form. This process is important because the resulting matrix will allow you to describe the solutions to the corresponding linear system of equations in a meaningful way.

In the next example, we look at how to solve a system of equations using the corresponding augmented matrix.

### Example C.20: Finding the Solution to a System

*Give the complete solution to the following system of equations*

$$\begin{aligned} 2x + 4y - 3z &= -1 \\ 5x + 10y - 7z &= -2 \\ 3x + 6y + 5z &= 9 \end{aligned}$$

**Solution.** The augmented matrix for this system is

$$\left[ \begin{array}{ccc|c} 2 & 4 & -3 & -1 \\ 5 & 10 & -7 & -2 \\ 3 & 6 & 5 & 9 \end{array} \right]$$

In order to find the solution to this system, we wish to carry the augmented matrix to reduced row-echelon form. We will do so using Algorithm ???. Notice that the first column is nonzero, so this is our first pivot column. The first entry in the first row, 2, is the first leading entry and it is in the first

pivot position. We will use row operations to create zeros in the entries below the 2. First, replace the second row with  $-5$  times the first row plus 2 times the second row. This yields

$$\left[ \begin{array}{ccc|c} 2 & 4 & -3 & -1 \\ 0 & 0 & 1 & 1 \\ 3 & 6 & 5 & 9 \end{array} \right]$$

Now, replace the third row with  $-3$  times the first row plus to 2 times the third row. This yields

$$\left[ \begin{array}{ccc|c} 2 & 4 & -3 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 21 \end{array} \right]$$

Now the entries in the first column below the pivot position are zeros. We now look for the second pivot column, which in this case is column three. Here, the 1 in the second row and third column is in the pivot position. We need to do just one row operation to create a zero below the 1.

Taking  $-1$  times the second row and adding it to the third row yields

$$\left[ \begin{array}{ccc|c} 2 & 4 & -3 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 20 \end{array} \right]$$

We could proceed with the algorithm to carry this matrix to row-echelon form or reduced row-echelon form. However, remember that we are looking for the solutions to the system of equations. Take another look at the third row of the matrix. Notice that it corresponds to the equation

$$0x + 0y + 0z = 20$$

There is no solution to this equation because for all  $x, y, z$ , the left side will equal 0 and  $0 \neq 20$ . This shows there is no solution to the given system of equations. In other words, this system is inconsistent. ♠

The following is another example of how to find the solution to a system of equations by carrying the corresponding augmented matrix to reduced row-echelon form.

### Example C.21: An Infinite Set of Solutions

*Give the complete solution to the system of equations*

$$\begin{aligned} 3x - y - 5z &= 9 \\ y - 10z &= 0 \\ -2x + y &= -6 \end{aligned} \tag{3.8}$$

**Solution.** The augmented matrix of this system is

$$\left[ \begin{array}{ccc|c} 3 & -1 & -5 & 9 \\ 0 & 1 & -10 & 0 \\ -2 & 1 & 0 & -6 \end{array} \right]$$

In order to find the solution to this system, we will carry the augmented matrix to reduced row-echelon form, using Algorithm ?? . The first column is the first pivot column. We want to use row operations to create zeros beneath the first entry in this column, which is in the first pivot position. Replace the third row with 2 times the first row added to 3 times the third row. This gives

$$\left[ \begin{array}{ccc|c} 3 & -1 & -5 & 9 \\ 0 & 1 & -10 & 0 \\ 0 & 1 & -10 & 0 \end{array} \right]$$

Now, we have created zeros beneath the 3 in the first column, so we move on to the second pivot column (which is the second column) and repeat the procedure. Take  $-1$  times the second row and add to the third row.

$$\left[ \begin{array}{ccc|c} 3 & -1 & -5 & 9 \\ 0 & 1 & -10 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

The entry below the pivot position in the second column is now a zero. Notice that we have no more pivot columns because we have only two leading entries.

At this stage, we also want the leading entries to be equal to one. To do so, divide the first row by 3.

$$\left[ \begin{array}{ccc|c} 1 & -\frac{1}{3} & -\frac{5}{3} & 3 \\ 0 & 1 & -10 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

This matrix is now in row-echelon form.

Let's continue with row operations until the matrix is in reduced row-echelon form. This involves creating zeros above the pivot positions in each pivot column. This requires only one step, which is to add  $\frac{1}{3}$  times the second row to the first row.

$$\left[ \begin{array}{ccc|c} 1 & 0 & -5 & 3 \\ 0 & 1 & -10 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

This is in reduced row-echelon form, which you should verify using Definition C.13. The equations corresponding to this reduced row-echelon form are

$$\begin{aligned} x - 5z &= 3 \\ y - 10z &= 0 \end{aligned}$$

or

$$\begin{aligned} x &= 3 + 5z \\ y &= 10z \end{aligned}$$

Observe that  $z$  is not restrained by any equation. In fact,  $z$  can equal any number. For example, we can let  $z = t$ , where we can choose  $t$  to be any number. In this context  $t$  is called a **parameter**. Therefore, the solution set of this system is

$$\begin{aligned} x &= 3 + 5t \\ y &= 10t \\ z &= t \end{aligned}$$

where  $t$  is arbitrary. The system has an infinite set of solutions which are given by these equations. For any value of  $t$  we select,  $x, y$ , and  $z$  will be given by the above equations. For example, if we choose  $t = 4$  then the corresponding solution would be

$$\begin{aligned}x &= 3 + 5(4) = 23 \\y &= 10(4) = 40 \\z &= 4\end{aligned}$$



In Example C.21 the solution involved one parameter. It may happen that the solution to a system involves more than one parameter, as shown in the following example.

### Example C.22: A Two Parameter Set of Solutions

*Find the solution to the system*

$$\begin{aligned}x + 2y - z + w &= 3 \\x + y - z + w &= 1 \\x + 3y - z + w &= 5\end{aligned}$$

**Solution.** The augmented matrix is

$$\left[ \begin{array}{cccc|c} 1 & 2 & -1 & 1 & 3 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 3 & -1 & 1 & 5 \end{array} \right]$$

We wish to carry this matrix to row-echelon form. Here, we will outline the row operations used. However, make sure that you understand the steps in terms of Algorithm ??.

Take  $-1$  times the first row and add to the second. Then take  $-1$  times the first row and add to the third. This yields

$$\left[ \begin{array}{cccc|c} 1 & 2 & -1 & 1 & 3 \\ 0 & -1 & 0 & 0 & -2 \\ 0 & 1 & 0 & 0 & 2 \end{array} \right]$$

Now add the second row to the third row and divide the second row by  $-1$ .

$$\left[ \begin{array}{cccc|c} 1 & 2 & -1 & 1 & 3 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right] \tag{3.9}$$

This matrix is in row-echelon form and we can see that  $x$  and  $y$  correspond to pivot columns, while  $z$  and  $w$  do not. Therefore, we will assign parameters to the variables  $z$  and  $w$ . Assign the parameter  $s$  to  $z$  and the parameter  $t$  to  $w$ . Then the first row yields the equation  $x + 2y - s + t = 3$ , while the second row yields the equation  $y = 2$ . Since  $y = 2$ , the first equation becomes  $x + 4 - s + t = 3$  showing that the solution is given by

$$\begin{aligned}x &= -1 + s - t \\y &= 2 \\z &= s \\w &= t\end{aligned}$$

It is customary to write this solution in the form

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} -1+s-t \\ 2 \\ s \\ t \end{bmatrix} \quad (3.10)$$



This example shows a system of equations with an infinite solution set which depends on two parameters. It can be less confusing in the case of an infinite solution set to first place the augmented matrix in reduced row-echelon form rather than just row-echelon form before seeking to write down the description of the solution.

In the above steps, this means we don't stop with the row-echelon form in equation 3.9. Instead we first place it in reduced row-echelon form as follows.

$$\left[ \begin{array}{cccc|c} 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right]$$

Then the solution is  $y = 2$  from the second row and  $x = -1 + z - w$  from the first. Thus letting  $z = s$  and  $w = t$ , the solution is given by 3.10.

You can see here that there are two paths to the correct answer, which both yield the same answer. Hence, either approach may be used. The process which we first used in the above solution is called **Gaussian Elimination**. This process involves carrying the matrix to row-echelon form, converting back to equations, and using back substitution to find the solution. When you do row operations until you obtain reduced row-echelon form, the process is called **Gauss-Jordan Elimination**.

We have now found solutions for systems of equations with no solution and infinitely many solutions, with one parameter as well as two parameters. Recall the three types of solution sets which we discussed in the previous section; no solution, one solution, and infinitely many solutions. Each of these types of solutions could be identified from the graph of the system. It turns out that we can also identify the type of solution from the reduced row-echelon form of the augmented matrix.

- *No Solution:* In the case where the system of equations has no solution, the row-echelon form of the augmented matrix will have a row of the form

$$\left[ \begin{array}{ccc|c} 0 & 0 & 0 & 1 \end{array} \right]$$

This row indicates that the system is inconsistent and has no solution.

- *One Solution:* In the case where the system of equations has one solution, every column of the coefficient matrix is a pivot column. The following is an example of an augmented matrix in reduced row-echelon form for a system of equations with one solution.

$$\left[ \begin{array}{ccc|c} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{array} \right]$$

- *Infinitely Many Solutions:* In the case where the system of equations has infinitely many solutions, the solution contains parameters. There will be columns of the coefficient matrix which are not pivot columns. The following are examples of augmented matrices in reduced row-echelon form for systems of equations with infinitely many solutions.

$$\left[ \begin{array}{ccc|c} 1 & 0 & 0 & 5 \\ 0 & 1 & 2 & -3 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

or

$$\left[ \begin{array}{ccc|c} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & -3 \end{array} \right]$$

### C.2.3. Uniqueness of the Reduced Row-Echelon Form

---

As we have seen in earlier sections, we know that every matrix can be brought into reduced row-echelon form by a sequence of elementary row operations. Here we will prove that the resulting matrix is unique; in other words, the resulting matrix in reduced row-echelon form does not depend upon the particular sequence of elementary row operations or the order in which they were performed.

Let  $A$  be the augmented matrix of a homogeneous system of linear equations in the variables  $x_1, x_2, \dots, x_n$  which is also in reduced row-echelon form. The matrix  $A$  divides the set of variables in two different types. We say that  $x_i$  is a *basic variable* whenever  $A$  has a leading 1 in column number  $i$ , in other words, when column  $i$  is a pivot column. Otherwise we say that  $x_i$  is a *free variable*.

Recall Example C.22.

#### Example C.23: Basic and Free Variables

*Find the basic and free variables in the system*

$$\begin{aligned} x + 2y - z + w &= 3 \\ x + y - z + w &= 1 \\ x + 3y - z + w &= 5 \end{aligned}$$

**Solution.** Recall from the solution of Example C.22 that the row-echelon form of the augmented matrix of this system is given by

$$\left[ \begin{array}{cccc|c} 1 & 2 & -1 & 1 & 3 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right]$$

You can see that columns 1 and 2 are pivot columns. These columns correspond to variables  $x$  and  $y$ , making these the basic variables. Columns 3 and 4 are not pivot columns, which means that  $z$  and  $w$  are free variables.

We can write the solution to this system as

$$\begin{aligned}x &= -1 + s - t \\y &= 2 \\z &= s \\w &= t\end{aligned}$$

Here the free variables are written as parameters, and the basic variables are given by linear functions of these parameters. ♠

In general, all solutions can be written in terms of the free variables. In such a description, the free variables can take any values (they become parameters), while the basic variables become simple linear functions of these parameters. Indeed, a basic variable  $x_i$  is a linear function of *only* those free variables  $x_j$  with  $j > i$ . This leads to the following observation.

### Proposition C.24: Basic and Free Variables

*If  $x_i$  is a basic variable of a homogeneous system of linear equations, then any solution of the system with  $x_j = 0$  for all those free variables  $x_j$  with  $j > i$  must also have  $x_i = 0$ .*

Using this proposition, we prove a lemma which will be used in the proof of the main result of this section below.

### Lemma C.25: Solutions and the Reduced Row-Echelon Form of a Matrix

*Let  $A$  and  $B$  be two distinct augmented matrices for two homogeneous systems of  $m$  equations in  $n$  variables, such that  $A$  and  $B$  are each in reduced row-echelon form. Then, the two systems do not have exactly the same solutions.*

Now, we say that the matrix  $B$  is **equivalent** to the matrix  $A$  provided that  $B$  can be obtained from  $A$  by performing a sequence of elementary row operations beginning with  $A$ . The importance of this concept lies in the following result.

### Theorem C.26: Equivalent Matrices

*The two linear systems of equations corresponding to two equivalent augmented matrices have exactly the same solutions.*

The proof of this theorem is left as an exercise.

Now, we can use Lemma C.25 and Theorem C.26 to prove the main result of this section.

### Theorem C.27: Uniqueness of the Reduced Row-Echelon Form

*Every matrix  $A$  is equivalent to a unique matrix in reduced row-echelon form.*

According to this theorem we can say that each matrix  $A$  has a unique reduced row-echelon form.

### C.2.4. Rank and Homogeneous Systems

There is a special type of system which requires additional study. This type of system is called a homogeneous system of equations, which we defined above in Definition C.3. Our focus in this section is to consider what types of solutions are possible for a homogeneous system of equations.

Consider the following definition.

#### Definition C.28: Trivial Solution

*Consider the homogeneous system of equations given by*

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= 0 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= 0 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= 0 \end{aligned}$$

*Then,  $x_1 = 0, x_2 = 0, \dots, x_n = 0$  is always a solution to this system. We call this the **trivial solution**.*

If the system has a solution in which not all of the  $x_1, \dots, x_n$  are equal to zero, then we call this solution **nontrivial**. The trivial solution does not tell us much about the system, as it says that  $0 = 0!$  Therefore, when working with homogeneous systems of equations, we want to know when the system has a nontrivial solution.

Suppose we have a homogeneous system of  $m$  equations, using  $n$  variables, and suppose that  $n > m$ . In other words, there are more variables than equations. Then, it turns out that this system always has a nontrivial solution. Not only will the system have a nontrivial solution, but it also will have infinitely many solutions. It is also possible, but not required, to have a nontrivial solution if  $n = m$  and  $n < m$ .

Consider the following example.

#### Example C.29: Solutions to a Homogeneous System of Equations

*Find the nontrivial solutions to the following homogeneous system of equations*

$$\begin{aligned} 2x + y - z &= 0 \\ x + 2y - 2z &= 0 \end{aligned}$$

**Solution.** Notice that this system has  $m = 2$  equations and  $n = 3$  variables, so  $n > m$ . Therefore by our previous discussion, we expect this system to have infinitely many solutions.

The process we use to find the solutions for a homogeneous system of equations is the same process we used in the previous section. First, we construct the augmented matrix, given by

$$\left[ \begin{array}{ccc|c} 2 & 1 & -1 & 0 \\ 1 & 2 & -2 & 0 \end{array} \right]$$

Then, we carry this matrix to its reduced row-echelon form, given below.

$$\left[ \begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \end{array} \right]$$

The corresponding system of equations is

$$\begin{aligned} x &= 0 \\ y - z &= 0 \end{aligned}$$

Since  $z$  is not restrained by any equation, we know that this variable will become our parameter. Let  $z = t$  where  $t$  is any number. Therefore, our solution has the form

$$\begin{aligned} x &= 0 \\ y &= z = t \\ z &= t \end{aligned}$$

Hence this system has infinitely many solutions, with one parameter  $t$ . ♠

Suppose we were to write the solution to the previous example in another form. Specifically,

$$\begin{aligned} x &= 0 \\ y &= 0 + t \\ z &= 0 + t \end{aligned}$$

can be written as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + t \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Notice that we have constructed a column from the constants in the solution (all equal to 0), as well as a column corresponding to the coefficients on  $t$  in each equation. While we will discuss this form of solution more in further chapters, for now consider the column of coefficients of the parameter  $t$ .

In this case, this is the column  $\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$ .

There is a special name for this column, which is **basic solution**. The basic solutions of a system are columns constructed from the coefficients on parameters in the solution. We often denote basic solutions by  $X_1, X_2$  etc., depending on how many solutions occur. Therefore, Example C.29 has the

$$\text{basic solution } X_1 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}.$$

We explore this further in the following example.

**Example C.30: Basic Solutions of a Homogeneous System**

Consider the following homogeneous system of equations.

$$\begin{aligned}x + 4y + 3z &= 0 \\3x + 12y + 9z &= 0\end{aligned}$$

Find the basic solutions to this system.

**Solution.** The augmented matrix of this system and the resulting reduced row-echelon form are

$$\left[ \begin{array}{ccc|c} 1 & 4 & 3 & 0 \\ 3 & 12 & 9 & 0 \end{array} \right] \rightarrow \dots \rightarrow \left[ \begin{array}{ccc|c} 1 & 4 & 3 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

When written in equations, this system is given by

$$x + 4y + 3z = 0$$

Notice that only  $x$  corresponds to a pivot column. In this case, we will have two parameters, one for  $y$  and one for  $z$ . Let  $y = s$  and  $z = t$  for any numbers  $s$  and  $t$ . Then, our solution becomes

$$\begin{aligned}x &= -4s - 3t \\y &= s \\z &= t\end{aligned}$$

which can be written as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + s \begin{bmatrix} -4 \\ 1 \\ 0 \end{bmatrix} + t \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix}$$

You can see here that we have two columns of coefficients corresponding to parameters, specifically one for  $s$  and one for  $t$ . Therefore, this system has two basic solutions! These are

$$X_1 = \begin{bmatrix} -4 \\ 1 \\ 0 \end{bmatrix}, X_2 = \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix}$$



We now present a new definition.

**Definition C.31: Linear Combination**

Let  $X_1, \dots, X_n, V$  be column matrices. Then  $V$  is said to be a **linear combination** of the columns  $X_1, \dots, X_n$  if there exist scalars,  $a_1, \dots, a_n$  such that

$$V = a_1X_1 + \dots + a_nX_n$$

A remarkable result of this section is that a linear combination of the basic solutions is again a solution to the system. Even more remarkable is that every solution can be written as a linear

combination of these solutions. Therefore, if we take a linear combination of the two solutions to Example C.30, this would also be a solution. For example, we could take the following linear combination

$$3 \begin{bmatrix} -4 \\ 1 \\ 0 \end{bmatrix} + 2 \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -18 \\ 3 \\ 2 \end{bmatrix}$$

You should take a moment to verify that

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -18 \\ 3 \\ 2 \end{bmatrix}$$

is in fact a solution to the system in Example C.30.

Another way in which we can find out more information about the solutions of a homogeneous system is to consider the **rank** of the associated coefficient matrix. We now define what is meant by the rank of a matrix.

### Definition C.32: Rank of a Matrix

*Let  $A$  be a matrix and consider any row-echelon form of  $A$ . Then, the number  $r$  of leading entries of  $A$  does not depend on the row-echelon form you choose, and is called the **rank** of  $A$ . We denote it by  $\text{rank}(A)$ .*

Similarly, we could count the number of pivot positions (or pivot columns) to determine the rank of  $A$ .

### Example C.33: Finding the Rank of a Matrix

Consider the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 5 & 9 \\ 2 & 4 & 6 \end{bmatrix}$$

What is its rank?

**Solution.** First, we need to find the reduced row-echelon form of  $A$ . Through the usual algorithm, we find that this is

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

Here we have two leading entries, or two pivot positions, shown above in boxes. The rank of  $A$  is  $r = 2$ .

Notice that we would have achieved the same answer if we had found the row-echelon form of  $A$  instead of the reduced row-echelon form.

Suppose we have a homogeneous system of  $m$  equations in  $n$  variables, and suppose that  $n > m$ . From our above discussion, we know that this system will have infinitely many solutions. If we consider the rank of the coefficient matrix of this system, we can find out even more about the solution. Note that we are looking at just the coefficient matrix, not the entire augmented matrix.

### Theorem C.34: Rank and Solutions to a Homogeneous System

*Let  $A$  be the  $m \times n$  coefficient matrix corresponding to a homogeneous system of equations, and suppose  $A$  has rank  $r$ . Then, the solution to the corresponding system has  $n - r$  parameters.*

Consider our above Example C.30 in the context of this theorem. The system in this example has  $m = 2$  equations in  $n = 3$  variables. First, because  $n > m$ , we know that the system has a nontrivial solution, and therefore infinitely many solutions. This tells us that the solution will contain at least one parameter. The rank of the coefficient matrix can tell us even more about the solution! The rank of the coefficient matrix of the system is 1, as it has one leading entry in row-echelon form. Theorem C.34 tells us that the solution will have  $n - r = 3 - 1 = 2$  parameters. You can check that this is true in the solution to Example C.30.

Notice that if  $n = m$  or  $n < m$ , it is possible to have either a unique solution (which will be the trivial solution) or infinitely many solutions.

We are not limited to homogeneous systems of equations here. The rank of a matrix can be used to learn about the solutions of any system of linear equations. In the previous section, we discussed that a system of equations can have no solution, a unique solution, or infinitely many solutions. Suppose the system is consistent, whether it is homogeneous or not. The following theorem tells us how we can use the rank to learn about the type of solution we have.

### Theorem C.35: Rank and Solutions to a Consistent System of Equations

*Let  $A$  be the  $m \times (n + 1)$  augmented matrix corresponding to a consistent system of equations in  $n$  variables, and suppose  $A$  has rank  $r$ . Then*

- (a) *the system has a unique solution if  $r = n$*
- (b) *the system has infinitely many solutions if  $r < n$*

We will not present a formal proof of this, but consider the following discussions.

- (a) *No Solution* The above theorem assumes that the system is consistent, that is, that it has a solution. It turns out that it is possible for the augmented matrix of a system with no solution to have any rank  $r$  as long as  $r > 1$ . Therefore, we must know that the system is consistent in order to use this theorem!
- (b) *Unique Solution* Suppose  $r = n$ . Then, there is a pivot position in every column of the coefficient matrix of  $A$ . Hence, there is a unique solution.
- (c) *Infinitely Many Solutions* Suppose  $r < n$ . Then there are infinitely many solutions. There are less pivot positions (and hence less leading entries) than columns, meaning that not every column is a pivot column. The columns which are *not* pivot columns correspond to parameters. In fact, in this case we have  $n - r$  parameters.

# D. Matrices

---

## D.1 Matrix Arithmetic

---

### Outcomes

- A. Perform the matrix operations of matrix addition, scalar multiplication, transposition and matrix multiplication. Identify when these operations are not defined. Represent these operations in terms of the entries of a matrix.
- B. Prove algebraic properties for matrix addition, scalar multiplication, transposition, and matrix multiplication. Apply these properties to manipulate an algebraic expression involving matrices.
- C. Compute the inverse of a matrix using row operations, and prove identities involving matrix inverses.
- E. Solve a linear system using matrix algebra.
- F. Use multiplication by an elementary matrix to apply row operations.
- G. Write a matrix as a product of elementary matrices.

You have now solved systems of equations by writing them in terms of an augmented matrix and then doing row operations on this augmented matrix. It turns out that matrices are important not only for systems of equations but also in many applications.

Recall that a **matrix** is a rectangular array of numbers. Several of them are referred to as **matrices**. For example, here is a matrix.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 2 & 8 & 7 \\ 6 & -9 & 1 & 2 \end{bmatrix} \quad (4.1)$$

Recall that the size or dimension of a matrix is defined as  $m \times n$  where  $m$  is the number of rows and  $n$  is the number of columns. The above matrix is a  $3 \times 4$  matrix because there are three rows and four columns. You can remember the columns are like columns in a Greek temple. They stand upright while the rows lay flat like rows made by a tractor in a plowed field.

When specifying the size of a matrix, you always list the number of rows before the number of columns. You might remember that you always list the rows before the columns by using the phrase **Rowman Catholic**.

Consider the following definition.

**Definition D.1: Square Matrix**

A matrix  $A$  which has size  $n \times n$  is called a **square matrix**. In other words,  $A$  is a square matrix if it has the same number of rows and columns.

There is some notation specific to matrices which we now introduce. We denote the columns of a matrix  $A$  by  $A_j$  as follows

$$A = [ A_1 \ A_2 \ \cdots \ A_n ]$$

Therefore,  $A_j$  is the  $j^{\text{th}}$  column of  $A$ , when counted from left to right.

The individual elements of the matrix are called **entries** or **components** of  $A$ . Elements of the matrix are identified according to their position. The **(i,j)-entry** of a matrix is the entry in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column. For example, in the matrix 4.1 above, 8 is in position  $(2,3)$  (and is called the  $(2,3)$ -entry) because it is in the second row and the third column.

In order to remember which matrix we are speaking of, we will denote the entry in the  $i^{\text{th}}$  row and the  $j^{\text{th}}$  column of matrix  $A$  by  $a_{ij}$ . Then, we can write  $A$  in terms of its entries, as  $A = [a_{ij}]$ . Using this notation on the matrix in 4.1,  $a_{23} = 8, a_{32} = -9, a_{12} = 2$ , etc.

There are various operations which are done on matrices of appropriate sizes. Matrices can be added to and subtracted from other matrices, multiplied by a scalar, and multiplied by other matrices. We will never divide a matrix by another matrix, but we will see later how matrix inverses play a similar role.

In doing arithmetic with matrices, we often define the action by what happens in terms of the entries (or components) of the matrices. Before looking at these operations in depth, consider a few general definitions.

**Definition D.2: The Zero Matrix**

The  $m \times n$  **zero matrix** is the  $m \times n$  matrix having every entry equal to zero. It is denoted by 0.

One possible zero matrix is shown in the following example.

**Example D.3: The Zero Matrix**

The  $2 \times 3$  zero matrix is  $0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ .

Note there is a  $2 \times 3$  zero matrix, a  $3 \times 4$  zero matrix, etc. In fact there is a zero matrix for every size!

**Definition D.4: Equality of Matrices**

Let  $A$  and  $B$  be two  $m \times n$  matrices. Then  $A = B$  means that for  $A = [a_{ij}]$  and  $B = [b_{ij}]$ ,  $a_{ij} = b_{ij}$  for all  $1 \leq i \leq m$  and  $1 \leq j \leq n$ .

In other words, two matrices are equal exactly when they are the same size and the corresponding entries are identical. Thus

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

because they are different sizes. Also,

$$\begin{bmatrix} 0 & 1 \\ 3 & 2 \end{bmatrix} \neq \begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix}$$

because, although they are the same size, their corresponding entries are not identical.

In the following section, we explore addition of matrices.

### D.1.1. Addition of Matrices

---

When adding matrices, all matrices in the sum need have the same size. For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 2 \end{bmatrix}$$

and

$$\begin{bmatrix} -1 & 4 & 8 \\ 2 & 8 & 5 \end{bmatrix}$$

cannot be added, as one has size  $3 \times 2$  while the other has size  $2 \times 3$ .

However, the addition

$$\begin{bmatrix} 4 & 6 & 3 \\ 5 & 0 & 4 \\ 11 & -2 & 3 \end{bmatrix} + \begin{bmatrix} 0 & 5 & 0 \\ 4 & -4 & 14 \\ 1 & 2 & 6 \end{bmatrix}$$

is possible.

The formal definition is as follows.

#### Definition D.5: Addition of Matrices

Let  $A = [a_{ij}]$  and  $B = [b_{ij}]$  be two  $m \times n$  matrices. Then  $A + B = C$  where  $C$  is the  $m \times n$  matrix  $C = [c_{ij}]$  defined by

$$c_{ij} = a_{ij} + b_{ij}$$

This definition tells us that when adding matrices, we simply add corresponding entries of the matrices. This is demonstrated in the next example.

**Example D.6: Addition of Matrices of Same Size**

Add the following matrices, if possible.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 4 \end{bmatrix}, B = \begin{bmatrix} 5 & 2 & 3 \\ -6 & 2 & 1 \end{bmatrix}$$

**Solution.** Notice that both  $A$  and  $B$  are of size  $2 \times 3$ . Since  $A$  and  $B$  are of the same size, the addition is possible. Using Definition D.5, the addition is done as follows.

$$A + B = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 2 & 3 \\ -6 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+2 & 3+3 \\ 1+(-6) & 0+2 & 4+1 \end{bmatrix} = \begin{bmatrix} 6 & 4 & 6 \\ -5 & 2 & 5 \end{bmatrix}$$



Addition of matrices obeys very much the same properties as normal addition with numbers. Note that when we write for example  $A + B$  then we assume that both matrices are of equal size so that the operation is indeed possible.

**Proposition D.7: Properties of Matrix Addition**

Let  $A, B$  and  $C$  be matrices. Then, the following properties hold.

- Commutative Law of Addition

$$A + B = B + A \quad (4.2)$$

- Associative Law of Addition

$$(A + B) + C = A + (B + C) \quad (4.3)$$

- Existence of an Additive Identity

There exists a zero matrix  $0$  such that  
 $A + 0 = A$  (4.4)

- Existence of an Additive Inverse

There exists a matrix  $-A$  such that  
 $A + (-A) = 0$  (4.5)

We call the zero matrix in 4.4 the **additive identity**. Similarly, we call the matrix  $-A$  in 4.5 the **additive inverse**.  $-A$  is defined to equal  $(-1)A = [-a_{ij}]$ . In other words, every entry of  $A$  is multiplied by  $-1$ . In the next section we will study scalar multiplication in more depth to understand what is meant by  $(-1)A$ .

## D.1.2. Scalar Multiplication of Matrices

Recall that we use the word *scalar* when referring to numbers. Therefore, *scalar multiplication of a matrix* is the multiplication of a matrix by a number. To illustrate this concept, consider the following example in which a matrix is multiplied by the scalar 3.

$$3 \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 2 & 8 & 7 \\ 6 & -9 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 & 12 \\ 15 & 6 & 24 & 21 \\ 18 & -27 & 3 & 6 \end{bmatrix}$$

The new matrix is obtained by multiplying every entry of the original matrix by the given scalar. The formal definition of scalar multiplication is as follows.

### Definition D.8: Scalar Multiplication of Matrices

If  $A = [a_{ij}]$  and  $k$  is a scalar, then  $kA = [ka_{ij}]$ .

Consider the following example.

### Example D.9: Effect of Multiplication by a Scalar

Find the result of multiplying the following matrix  $A$  by 7.

$$A = \begin{bmatrix} 2 & 0 \\ 1 & -4 \end{bmatrix}$$

**Solution.** By Definition D.8, we multiply each element of  $A$  by 7. Therefore,

$$7A = 7 \begin{bmatrix} 2 & 0 \\ 1 & -4 \end{bmatrix} = \begin{bmatrix} 7(2) & 7(0) \\ 7(1) & 7(-4) \end{bmatrix} = \begin{bmatrix} 14 & 0 \\ 7 & -28 \end{bmatrix}$$



Similarly to addition of matrices, there are several properties of scalar multiplication which hold.

### Proposition D.10: Properties of Scalar Multiplication

Let  $A, B$  be matrices, and  $k, p$  be scalars. Then, the following properties hold.

- Distributive Law over Matrix Addition

$$k(A + B) = kA + kB$$

- Distributive Law over Scalar Addition

$$(k + p)A = kA + pA$$

- Associative Law for Scalar Multiplication

$$k(pA) = (kp)A$$

- Rule for Multiplication by 1

$$1A = A$$

The proof of this proposition is similar to the proof of Proposition D.7 and is left an exercise to the reader.

### D.1.3. Multiplication of Matrices

The next important matrix operation we will explore is multiplication of matrices. The operation of matrix multiplication is one of the most important and useful of the matrix operations. Throughout this section, we will also demonstrate how matrix multiplication relates to linear systems of equations.

First, we provide a formal definition of row and column vectors.

#### Definition D.11: Row and Column Vectors

Matrices of size  $n \times 1$  or  $1 \times n$  are called **vectors**. If  $X$  is such a matrix, then we write  $x_i$  to denote the entry of  $X$  in the  $i^{\text{th}}$  row of a column matrix, or the  $i^{\text{th}}$  column of a row matrix.

The  $n \times 1$  matrix

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

is called a **column vector**. The  $1 \times n$  matrix

$$X = [x_1 \ \cdots \ x_n]$$

is called a **row vector**.

We may simply use the term **vector** throughout this text to refer to either a column or row vector. If we do so, the context will make it clear which we are referring to.

In this chapter, we will again use the notion of linear combination of vectors as in Definition F.7. In this context, a linear combination is a sum consisting of vectors multiplied by scalars. For example,

$$\begin{bmatrix} 50 \\ 122 \end{bmatrix} = 7 \begin{bmatrix} 1 \\ 4 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 5 \end{bmatrix} + 9 \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

is a linear combination of three vectors.

It turns out that we can express any system of linear equations as a linear combination of vectors. In fact, the vectors that we will use are just the columns of the corresponding augmented matrix!

### Definition D.12: The Vector Form of a System of Linear Equations

Suppose we have a system of equations given by

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

We can express this system in **vector form** which is as follows:

$$x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{nn} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Notice that each vector used here is one column from the corresponding augmented matrix. There is one vector for each variable in the system, along with the constant vector.

The first important form of matrix multiplication is multiplying a matrix by a vector. Consider the product given by

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix}$$

We will soon see that this equals

$$7 \begin{bmatrix} 1 \\ 4 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 5 \end{bmatrix} + 9 \begin{bmatrix} 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 50 \\ 122 \end{bmatrix}$$

In general terms,

$$\begin{aligned} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} &= x_1 \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix} + x_3 \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \end{bmatrix} \end{aligned}$$

Thus you take  $x_1$  times the first column, add to  $x_2$  times the second column, and finally  $x_3$  times the third column. The above sum is a linear combination of the columns of the matrix. When you multiply a matrix on the left by a vector on the right, the numbers making up the vector are just the scalars to be used in the linear combination of the columns as illustrated above.

Here is the formal definition of how to multiply an  $m \times n$  matrix by an  $n \times 1$  column vector.

### Definition D.13: Multiplication of Vector by Matrix

Let  $A = [a_{ij}]$  be an  $m \times n$  matrix and let  $X$  be an  $n \times 1$  matrix given by

$$A = [A_1 \cdots A_n], X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Then the product  $AX$  is the  $m \times 1$  column vector which equals the following linear combination of the columns of  $A$ :

$$x_1 A_1 + x_2 A_2 + \cdots + x_n A_n = \sum_{j=1}^n x_j A_j$$

If we write the columns of  $A$  in terms of their entries, they are of the form

$$A_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

Then, we can write the product  $AX$  as

$$AX = x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix}$$

Note that multiplication of an  $m \times n$  matrix and an  $n \times 1$  vector produces an  $m \times 1$  vector.

Here is an example.

### Example D.14: A Vector Multiplied by a Matrix

Compute the product  $AX$  for

$$A = \begin{bmatrix} 1 & 2 & 1 & 3 \\ 0 & 2 & 1 & -2 \\ 2 & 1 & 4 & 1 \end{bmatrix}, X = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 1 \end{bmatrix}$$

**Solution.** We will use Definition D.13 to compute the product. Therefore, we compute the product  $AX$  as follows.

$$\begin{aligned}
 & 1 \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + 2 \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} + 0 \begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix} + 1 \begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix} \\
 &= \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix} \\
 &= \begin{bmatrix} 8 \\ 2 \\ 5 \end{bmatrix}
 \end{aligned}$$



Using the above operation, we can also write a system of linear equations in **matrix form**. In this form, we express the system as a matrix multiplied by a vector. Consider the following definition.

### Definition D.15: The Matrix Form of a System of Linear Equations

Suppose we have a system of equations given by

$$\begin{aligned}
 a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\
 a_{21}x_1 + \cdots + a_{2n}x_n &= b_2 \\
 &\vdots \\
 a_{m1}x_1 + \cdots + a_{mn}x_n &= b_m
 \end{aligned}$$

Then we can express this system in **matrix form** as follows.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

The expression  $AX = B$  is also known as the **Matrix Form** of the corresponding system of linear equations. The matrix  $A$  is simply the coefficient matrix of the system, the vector  $X$  is the column vector constructed from the variables of the system, and finally the vector  $B$  is the column vector constructed from the constants of the system. It is important to note that any system of linear equations can be written in this form.

Notice that if we write a homogeneous system of equations in matrix form, it would have the form  $AX = 0$ , for the zero vector  $0$ .

You can see from this definition that a vector

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

will satisfy the equation  $AX = B$  only when the entries  $x_1, x_2, \dots, x_n$  of the vector  $X$  are solutions to the original system.

Now that we have examined how to multiply a matrix by a vector, we wish to consider the case where we multiply two matrices of more general sizes, although these sizes still need to be appropriate as we will see. For example, in Example D.14, we multiplied a  $3 \times 4$  matrix by a  $4 \times 1$  vector. We want to investigate how to multiply other sizes of matrices.

We have not yet given any conditions on when matrix multiplication is possible! For matrices  $A$  and  $B$ , in order to form the product  $AB$ , the number of columns of  $A$  must equal the number of rows of  $B$ . Consider a product  $AB$  where  $A$  has size  $m \times n$  and  $B$  has size  $n \times p$ . Then, the product in terms of size of matrices is given by

$$(m \times \overbrace{n}^{\text{these must match!}} (n \times p)) = m \times p$$

Note the two outside numbers give the size of the product. One of the most important rules regarding matrix multiplication is the following. If the two middle numbers don't match, you can't multiply the matrices!

When the number of columns of  $A$  equals the number of rows of  $B$  the two matrices are said to be **conformable** and the product  $AB$  is obtained as follows.

### Definition D.16: Multiplication of Two Matrices

Let  $A$  be an  $m \times n$  matrix and let  $B$  be an  $n \times p$  matrix of the form

$$B = [B_1 \cdots B_p]$$

where  $B_1, \dots, B_p$  are the  $n \times 1$  columns of  $B$ . Then the  $m \times p$  matrix  $AB$  is defined as follows:

$$AB = A[B_1 \cdots B_p] = [(AB)_1 \cdots (AB)_p]$$

where  $(AB)_k$  is an  $m \times 1$  matrix or column vector which gives the  $k^{\text{th}}$  column of  $AB$ .

Consider the following example.

### Example D.17: Multiplying Two Matrices

Find  $AB$  if possible.

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

**Solution.** The first thing you need to verify when calculating a product is whether the multiplication is possible. The first matrix has size  $2 \times 3$  and the second matrix has size  $3 \times 3$ . The inside numbers are equal, so  $A$  and  $B$  are conformable matrices. According to the above discussion  $AB$  will be a  $2 \times 3$  matrix. Definition D.16 gives us a way to calculate each column of  $AB$ , as follows.

$$\left[ \underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}}_{\text{First column}} \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}, \underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}}_{\text{Second column}} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}, \underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}}_{\text{Third column}} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \right]$$

You know how to multiply a matrix times a vector, using Definition D.13 for each of the three columns. Thus

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 1 \\ -2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 9 & 3 \\ -2 & 7 & 3 \end{bmatrix}$$



Since vectors are simply  $n \times 1$  or  $1 \times m$  matrices, we can also multiply a vector by another vector.

### Example D.18: Vector Times Vector Multiplication

*Multiply if possible*  $\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} [ 1 \ 2 \ 1 \ 0 ]$ .

**Solution.** In this case we are multiplying a matrix of size  $3 \times 1$  by a matrix of size  $1 \times 4$ . The inside numbers match so the product is defined. Note that the product will be a matrix of size  $3 \times 4$ . Using Definition D.16, we can compute this product as follows

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} [ 1 \ 2 \ 1 \ 0 ] = \left[ \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{First column}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{Second column}} \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{Third column}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{Fourth column}} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right]$$

You can use Definition D.13 to verify that this product is

$$\begin{bmatrix} 1 & 2 & 1 & 0 \\ 2 & 4 & 2 & 0 \\ 1 & 2 & 1 & 0 \end{bmatrix}$$



**Example D.19: A Multiplication Which is Not Defined**

*Find  $BA$  if possible.*

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

**Solution.** First check if it is possible. This product is of the form  $(3 \times 3)(2 \times 3)$ . The inside numbers do not match and so you can't do this multiplication. ♠

In this case, we say that the multiplication is not defined. Notice that these are the same matrices which we used in Example D.17. In this example, we tried to calculate  $BA$  instead of  $AB$ . This demonstrates another property of matrix multiplication. While the product  $AB$  maybe be defined, we cannot assume that the product  $BA$  will be possible. Therefore, it is important to always check that the product is defined before carrying out any calculations.

Earlier, we defined the zero matrix  $0$  to be the matrix (of appropriate size) containing zeros in all entries. Consider the following example for multiplication by the zero matrix.

**Example D.20: Multiplication by the Zero Matrix**

*Compute the product  $A0$  for the matrix*

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

*and the  $2 \times 2$  zero matrix given by*

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

**Solution.** In this product, we compute

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Hence,  $A0 = 0$ . ♠

Notice that we could also multiply  $A$  by the  $2 \times 1$  zero vector given by  $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$ . The result would be the  $2 \times 1$  zero vector. Therefore, it is always the case that  $A0 = 0$ , for an appropriately sized zero matrix or vector.

### D.1.4. The $ij^{\text{th}}$ Entry of a Product

In previous sections, we used the entries of a matrix to describe the action of matrix addition and scalar multiplication. We can also study matrix multiplication using the entries of matrices.

What is the  $ij^{\text{th}}$  entry of  $AB$ ? It is the entry in the  $i^{\text{th}}$  row and the  $j^{\text{th}}$  column of the product  $AB$ .

Now if  $A$  is  $m \times n$  and  $B$  is  $n \times p$ , then we know that the product  $AB$  has the form

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2p} \\ \vdots & \vdots & & \vdots & & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nj} & \cdots & b_{np} \end{bmatrix}$$

The  $j^{\text{th}}$  column of  $AB$  is of the form

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{nj} \end{bmatrix}$$

which is an  $m \times 1$  column vector. It is calculated by

$$b_{1j} \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + b_{2j} \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + b_{nj} \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix}$$

Therefore, the  $ij^{\text{th}}$  entry is the entry in row  $i$  of this vector. This is computed by

$$a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj}$$

The following is the formal definition for the  $ij^{\text{th}}$  entry of a product of matrices.

#### Definition D.21: The $ij^{\text{th}}$ Entry of a Product

Let  $A = [a_{ij}]$  be an  $m \times n$  matrix and let  $B = [b_{ij}]$  be an  $n \times p$  matrix. Then  $AB$  is an  $m \times p$  matrix and the  $(i, j)$ -entry of  $AB$  is defined as

$$(AB)_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$$

Another way to write this is

$$(AB)_{ij} = [a_{i1} \ a_{i2} \ \cdots \ a_{in}] \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{nj} \end{bmatrix} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

In other words, to find the  $(i, j)$ -entry of the product  $AB$ , or  $(AB)_{ij}$ , you multiply the  $i^{\text{th}}$  row of  $A$ , on the left by the  $j^{\text{th}}$  column of  $B$ . To express  $AB$  in terms of its entries, we write  $AB = [(AB)_{ij}]$ .

Consider the following example.

### Example D.22: The Entries of a Product

*Compute  $AB$  if possible. If it is, find the  $(3, 2)$ -entry of  $AB$  using Definition D.21.*

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix}, B = \begin{bmatrix} 2 & 3 & 1 \\ 7 & 6 & 2 \end{bmatrix}$$

**Solution.** First check if the product is possible. It is of the form  $(3 \times 2)(2 \times 3)$  and since the inside numbers match, it is possible to do the multiplication. The result should be a  $3 \times 3$  matrix. We can first compute  $AB$ :

$$\left[ \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 2 \\ 7 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right]$$

where the commas separate the columns in the resulting product. Thus the above product equals

$$\begin{bmatrix} 16 & 15 & 5 \\ 13 & 15 & 5 \\ 46 & 42 & 14 \end{bmatrix}$$

which is a  $3 \times 3$  matrix as desired. Thus, the  $(3, 2)$ -entry equals 42.

Now using Definition D.21, we can find that the  $(3, 2)$ -entry equals

$$\begin{aligned} \sum_{k=1}^2 a_{3k} b_{k2} &= a_{31} b_{12} + a_{32} b_{22} \\ &= 2 \times 3 + 6 \times 6 = 42 \end{aligned}$$

Consulting our result for  $AB$  above, this is correct!

You may wish to use this method to verify that the rest of the entries in  $AB$  are correct. ♠

Here is another example.

### Example D.23: Finding the Entries of a Product

*Determine if the product  $AB$  is defined. If it is, find the  $(2, 1)$ -entry of the product.*

$$A = \begin{bmatrix} 2 & 3 & 1 \\ 7 & 6 & 2 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix}$$

**Solution.** This product is of the form  $(3 \times 3)(3 \times 2)$ . The middle numbers match so the matrices are conformable and it is possible to compute the product.

We want to find the  $(2, 1)$ -entry of  $AB$ , that is, the entry in the second row and first column of the product. We will use Definition D.21, which states

$$(AB)_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

In this case,  $n = 3$ ,  $i = 2$  and  $j = 1$ . Hence the  $(2, 1)$ -entry is found by computing

$$(AB)_{21} = \sum_{k=1}^3 a_{2k} b_{k1} = \begin{bmatrix} a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix}$$

Substituting in the appropriate values, this product becomes

$$\begin{bmatrix} a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 7 & 6 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = 1 \times 7 + 3 \times 6 + 2 \times 2 = 29$$

Hence,  $(AB)_{21} = 29$ .

You should take a moment to find a few other entries of  $AB$ . You can multiply the matrices to check that your answers are correct. The product  $AB$  is given by

$$AB = \begin{bmatrix} 13 & 13 \\ 29 & 32 \\ 0 & 0 \end{bmatrix}$$



## D.1.5. Properties of Matrix Multiplication

As pointed out above, it is sometimes possible to multiply matrices in one order but not in the other order. However, even if both  $AB$  and  $BA$  are defined, they may not be equal.

### Example D.24: Matrix Multiplication is Not Commutative

Compare the products  $AB$  and  $BA$ , for matrices  $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ ,  $B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

**Solution.** First, notice that  $A$  and  $B$  are both of size  $2 \times 2$ . Therefore, both products  $AB$  and  $BA$  are defined. The first product,  $AB$  is

$$AB = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}$$

The second product,  $BA$  is

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}$$

Therefore,  $AB \neq BA$ . ♠

This example illustrates that you cannot assume  $AB = BA$  even when multiplication is defined in both orders. If for some matrices  $A$  and  $B$  it is true that  $AB = BA$ , then we say that  $A$  and  $B$  **commute**. This is one important property of matrix multiplication.

The following are other important properties of matrix multiplication. Notice that these properties hold only when the size of matrices are such that the products are defined.

### Proposition D.25: Properties of Matrix Multiplication

*The following hold for matrices  $A, B$ , and  $C$  and for scalars  $r$  and  $s$ ,*

$$A(rB + sC) = r(AB) + s(AC) \quad (4.6)$$

$$(B + C)A = BA + CA \quad (4.7)$$

$$A(BC) = (AB)C \quad (4.8)$$

## D.1.6. The Transpose

Another important operation on matrices is that of taking the **transpose**. For a matrix  $A$ , we denote the **transpose** of  $A$  by  $A^T$ . Before formally defining the transpose, we explore this operation on the following matrix.

$$\begin{bmatrix} 1 & 4 \\ 3 & 1 \\ 2 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 2 \\ 4 & 1 & 6 \end{bmatrix}$$

What happened? The first column became the first row and the second column became the second row. Thus the  $3 \times 2$  matrix became a  $2 \times 3$  matrix. The number 4 was in the first row and the second column and it ended up in the second row and first column.

The definition of the transpose is as follows.

### Definition D.26: The Transpose of a Matrix

*Let  $A$  be an  $m \times n$  matrix. Then  $A^T$ , the **transpose** of  $A$ , denotes the  $n \times m$  matrix given by*

$$A^T = [a_{ij}]^T = [a_{ji}]$$

The  $(i, j)$ -entry of  $A$  becomes the  $(j, i)$ -entry of  $A^T$ .

Consider the following example.

### Example D.27: The Transpose of a Matrix

Calculate  $A^T$  for the following matrix

$$A = \begin{bmatrix} 1 & 2 & -6 \\ 3 & 5 & 4 \end{bmatrix}$$

**Solution.** By Definition D.26, we know that for  $A = [a_{ij}]$ ,  $A^T = [a_{ji}]$ . In other words, we switch the row and column location of each entry. The  $(1,2)$ -entry becomes the  $(2,1)$ -entry.

Thus,

$$A^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ -6 & 4 \end{bmatrix}$$

Notice that  $A$  is a  $2 \times 3$  matrix, while  $A^T$  is a  $3 \times 2$  matrix. ♠

The transpose of a matrix has the following important properties .

### Lemma D.28: Properties of the Transpose of a Matrix

Let  $A$  be an  $m \times n$  matrix,  $B$  an  $n \times p$  matrix, and  $r$  and  $s$  scalars. Then

- (a)  $(A^T)^T = A$
- (b)  $(AB)^T = B^T A^T$
- (c)  $(rA + sB)^T = rA^T + sB^T$

The transpose of a matrix is related to other important topics. Consider the following definition.

### Definition D.29: Symmetric and Skew Symmetric Matrices

An  $n \times n$  matrix  $A$  is said to be **symmetric** if  $A = A^T$ . It is said to be **skew symmetric** if  $A = -A^T$ .

We will explore these definitions in the following examples.

### Example D.30: Symmetric Matrices

Let

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & -3 \\ 3 & -3 & 7 \end{bmatrix}$$

Use Definition D.29 to show that  $A$  is symmetric.

**Solution.** By Definition D.29, we need to show that  $A = A^T$ . Now, using Definition D.26,

$$A^T = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & -3 \\ 3 & -3 & 7 \end{bmatrix}$$

Hence,  $A = A^T$ , so  $A$  is symmetric. ♠

### Example D.31: A Skew Symmetric Matrix

Let

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

Show that  $A$  is skew symmetric.

**Solution.** By Definition D.29,

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

You can see that each entry of  $A^T$  is equal to  $-1$  times the same entry of  $A$ . Hence,  $A^T = -A$  and so by Definition D.29,  $A$  is skew symmetric. ♠

## D.1.7. The Identity and Inverses

---

There is a special matrix, denoted  $I$ , which is called to as the **identity matrix**. The identity matrix is always a square matrix, and it has the property that there are ones down the main diagonal and zeroes elsewhere. Here are some identity matrices of various sizes.

$$[1], \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The first is the  $1 \times 1$  identity matrix, the second is the  $2 \times 2$  identity matrix, and so on. By extension, you can likely see what the  $n \times n$  identity matrix would be. When it is necessary to distinguish which size of identity matrix is being discussed, we will use the notation  $I_n$  for the  $n \times n$  identity matrix.

The identity matrix is so important that there is a special symbol to denote the  $ij^{th}$  entry of the identity matrix. This symbol is given by  $I_{ij} = \delta_{ij}$  where  $\delta_{ij}$  is the **Kronecker symbol** defined by

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

$I_n$  is called the **identity matrix** because it is a **multiplicative identity** in the following sense.

**Lemma D.32: Multiplication by the Identity Matrix**

Suppose  $A$  is an  $m \times n$  matrix and  $I_n$  is the  $n \times n$  identity matrix. Then  $AI_n = A$ . If  $I_m$  is the  $m \times m$  identity matrix, it also follows that  $I_mA = A$ .

We now define the matrix operation which in some ways plays the role of division.

**Definition D.33: The Inverse of a Matrix**

A square  $n \times n$  matrix  $A$  is said to have an **inverse**  $A^{-1}$  if and only if

$$AA^{-1} = A^{-1}A = I_n$$

In this case, the matrix  $A$  is called **invertible**.

Such a matrix  $A^{-1}$  will have the same size as the matrix  $A$ . It is very important to observe that the inverse of a matrix, if it exists, is unique. Another way to think of this is that if it acts like the inverse, then it **is** the inverse.

**Theorem D.34: Uniqueness of Inverse**

Suppose  $A$  is an  $n \times n$  matrix such that an inverse  $A^{-1}$  exists. Then there is only one such inverse matrix. That is, given any matrix  $B$  such that  $AB = BA = I$ ,  $B = A^{-1}$ .

The next example demonstrates how to check the inverse of a matrix.

**Example D.35: Verifying the Inverse of a Matrix**

Let  $A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ . Show  $\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$  is the inverse of  $A$ .

**Solution.** To check this, multiply

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

and

$$\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

showing that this matrix is indeed the inverse of  $A$ . ♠

Unlike ordinary multiplication of numbers, it can happen that  $A \neq 0$  but  $A$  may fail to have an inverse. This is illustrated in the following example.

**Example D.36: A Nonzero Matrix With No Inverse**

Let  $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ . Show that  $A$  does not have an inverse.

**Solution.** One might think  $A$  would have an inverse because it does not equal zero. However, note that

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

If  $A^{-1}$  existed, we would have the following

$$\begin{aligned} \begin{bmatrix} 0 \\ 0 \end{bmatrix} &= A^{-1} \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) \\ &= A^{-1} \left( A \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) \\ &= (A^{-1}A) \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= I \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} -1 \\ 1 \end{bmatrix} \end{aligned}$$

This says that

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

which is impossible! Therefore,  $A$  does not have an inverse. ♠

In the next section, we will explore how to find the inverse of a matrix, if it exists.

### D.1.8. Finding the Inverse of a Matrix

In Example D.35, we were given  $A^{-1}$  and asked to verify that this matrix was in fact the inverse of  $A$ . In this section, we explore how to find  $A^{-1}$ .

Let

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

as in Example D.35. In order to find  $A^{-1}$ , we need to find a matrix  $\begin{bmatrix} x & z \\ y & w \end{bmatrix}$  such that

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x & z \\ y & w \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We can multiply these two matrices, and see that in order for this equation to be true, we must find the solution to the systems of equations,

$$\begin{aligned}x + y &= 1 \\x + 2y &= 0\end{aligned}$$

and

$$\begin{aligned}z + w &= 0 \\z + 2w &= 1\end{aligned}$$

Writing the augmented matrix for these two systems gives

$$\left[ \begin{array}{cc|c} 1 & 1 & 1 \\ 1 & 2 & 0 \end{array} \right]$$

for the first system and

$$\left[ \begin{array}{cc|c} 1 & 1 & 0 \\ 1 & 2 & 1 \end{array} \right] \quad (4.9)$$

for the second.

Let's solve the first system. Take  $-1$  times the first row and add to the second to get

$$\left[ \begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 1 & -1 \end{array} \right]$$

Now take  $-1$  times the second row and add to the first to get

$$\left[ \begin{array}{cc|c} 1 & 0 & 2 \\ 0 & 1 & -1 \end{array} \right]$$

Writing in terms of variables, this says  $x = 2$  and  $y = -1$ .

Now solve the second system, 4.9 to find  $z$  and  $w$ . You will find that  $z = -1$  and  $w = 1$ .

If we take the values found for  $x, y, z$ , and  $w$  and put them into our inverse matrix, we see that the inverse is

$$A^{-1} = \begin{bmatrix} x & z \\ y & w \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

After taking the time to solve the second system, you may have noticed that exactly the same row operations were used to solve both systems. In each case, the end result was something of the form  $[I|X]$  where  $I$  is the identity and  $X$  gave a column of the inverse. In the above,

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

the first column of the inverse was obtained by solving the first system and then the second column

$$\begin{bmatrix} z \\ w \end{bmatrix}$$

To simplify this procedure, we could have solved both systems at once! To do so, we could have written

$$\left[ \begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{array} \right]$$

and row reduced until we obtained

$$\left[ \begin{array}{cc|cc} 1 & 0 & 2 & -1 \\ 0 & 1 & -1 & 1 \end{array} \right]$$

and read off the inverse as the  $2 \times 2$  matrix on the right side.

This exploration motivates the following important algorithm.

**Matrix Inverse Algorithm** Suppose  $A$  is an  $n \times n$  matrix. To find  $A^{-1}$  if it exists, form the augmented  $n \times 2n$  matrix

$$[A|I]$$

If possible do row operations until you obtain an  $n \times 2n$  matrix of the form

$$[I|B]$$

When this has been done,  $B = A^{-1}$ . In this case, we say that  $A$  is **invertible**. If it is impossible to row reduce to a matrix of the form  $[I|B]$ , then  $A$  has no inverse.

This algorithm shows how to find the inverse if it exists. It will also tell you if  $A$  does not have an inverse.

Consider the following example.

### Example D.37: Finding the Inverse

Let  $A = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 0 & 2 \\ 3 & 1 & -1 \end{bmatrix}$ . Find  $A^{-1}$  if it exists.

**Solution.** Set up the augmented matrix

$$[A|I] = \left[ \begin{array}{ccc|ccc} 1 & 2 & 2 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 3 & 1 & -1 & 0 & 0 & 1 \end{array} \right]$$

Now we row reduce, with the goal of obtaining the  $3 \times 3$  identity matrix on the left hand side. First, take  $-1$  times the first row and add to the second followed by  $-3$  times the first row added to the third row. This yields

$$\left[ \begin{array}{ccc|ccc} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -2 & 0 & -1 & 1 & 0 \\ 0 & -5 & -7 & -3 & 0 & 1 \end{array} \right]$$

Then take 5 times the second row and add to -2 times the third row.

$$\left[ \begin{array}{ccc|ccc} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -10 & 0 & -5 & 5 & 0 \\ 0 & 0 & 14 & 1 & 5 & -2 \end{array} \right]$$

Next take the third row and add to  $-7$  times the first row. This yields

$$\left[ \begin{array}{ccc|ccc} -7 & -14 & 0 & -6 & 5 & -2 \\ 0 & -10 & 0 & -5 & 5 & 0 \\ 0 & 0 & 14 & 1 & 5 & -2 \end{array} \right]$$

Now take  $-\frac{7}{5}$  times the second row and add to the first row.

$$\left[ \begin{array}{ccc|ccc} -7 & 0 & 0 & 1 & -2 & -2 \\ 0 & -10 & 0 & -5 & 5 & 0 \\ 0 & 0 & 14 & 1 & 5 & -2 \end{array} \right]$$

Finally divide the first row by  $-7$ , the second row by  $-10$  and the third row by  $14$  which yields

$$\left[ \begin{array}{ccc|ccc} 1 & 0 & 0 & -\frac{1}{7} & \frac{2}{7} & \frac{2}{7} \\ 0 & 1 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & 0 & 1 & \frac{1}{14} & \frac{5}{14} & -\frac{1}{7} \end{array} \right]$$

Notice that the left hand side of this matrix is now the  $3 \times 3$  identity matrix  $I_3$ . Therefore, the inverse is the  $3 \times 3$  matrix on the right hand side, given by

$$\left[ \begin{array}{ccc} -\frac{1}{7} & \frac{2}{7} & \frac{2}{7} \\ \frac{1}{2} & -\frac{1}{2} & 0 \\ \frac{1}{14} & \frac{5}{14} & -\frac{1}{7} \end{array} \right]$$



It may happen that through this algorithm, you discover that the left hand side cannot be row reduced to the identity matrix. Consider the following example of this situation.

### Example D.38: A Matrix Which Has No Inverse

Let  $A = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 0 & 2 \\ 2 & 2 & 4 \end{bmatrix}$ . Find  $A^{-1}$  if it exists.

**Solution.** Write the augmented matrix  $[A|I]$

$$\left[ \begin{array}{ccc|ccc} 1 & 2 & 2 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 2 & 2 & 4 & 0 & 0 & 1 \end{array} \right]$$

and proceed to do row operations attempting to obtain  $[I|A^{-1}]$ . Take  $-1$  times the first row and add to the second. Then take  $-2$  times the first row and add to the third row.

$$\left[ \begin{array}{ccc|ccc} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -2 & 0 & -1 & 1 & 0 \\ 0 & -2 & 0 & -2 & 0 & 1 \end{array} \right]$$

Next add  $-1$  times the second row to the third row.

$$\left[ \begin{array}{ccc|ccc} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -2 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 \end{array} \right]$$

At this point, you can see there will be no way to obtain  $I$  on the left side of this augmented matrix. Hence, there is no way to complete this algorithm, and therefore the inverse of  $A$  does not exist. In this case, we say that  $A$  is not invertible. ♠

If the algorithm provides an inverse for the original matrix, it is always possible to check your answer. To do so, use the method demonstrated in Example D.35. Check that the products  $AA^{-1}$  and  $A^{-1}A$  both equal the identity matrix. Through this method, you can always be sure that you have calculated  $A^{-1}$  properly!

One way in which the inverse of a matrix is useful is to find the solution of a system of linear equations. Recall from Definition D.15 that we can write a system of equations in matrix form, which is of the form  $AX = B$ . Suppose you find the inverse of the matrix  $A^{-1}$ . Then you could multiply both sides of this equation on the left by  $A^{-1}$  and simplify to obtain

$$\begin{aligned} (A^{-1})AX &= A^{-1}B \\ (A^{-1}A)X &= A^{-1}B \\ IX &= A^{-1}B \\ X &= A^{-1}B \end{aligned}$$

Therefore we can find  $X$ , the solution to the system, by computing  $X = A^{-1}B$ . Note that once you have found  $A^{-1}$ , you can easily get the solution for different right hand sides (different  $B$ ). It is always just  $A^{-1}B$ .

We will explore this method of finding the solution to a system in the following example.

### Example D.39: Using the Inverse to Solve a System of Equations

Consider the following system of equations. Use the inverse of a suitable matrix to give the solutions to this system.

$$\begin{aligned} x + z &= 1 \\ x - y + z &= 3 \\ x + y - z &= 2 \end{aligned}$$

**Solution.** First, we can write the system of equations in matrix form

$$AX = \left[ \begin{array}{ccc} 1 & 0 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{array} \right] \left[ \begin{array}{c} x \\ y \\ z \end{array} \right] = \left[ \begin{array}{c} 1 \\ 3 \\ 2 \end{array} \right] = B \quad (4.10)$$

The inverse of the matrix

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

is

$$A^{-1} = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & -1 & 0 \\ 1 & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

Verifying this inverse is left as an exercise.

From here, the solution to the given system 4.10 is found by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = A^{-1}B = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & -1 & 0 \\ 1 & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ -2 \\ -\frac{3}{2} \end{bmatrix}$$



What if the right side,  $B$ , of 4.10 had been  $\begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$ ? In other words, what would be the solution to

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}?$$

By the above discussion, the solution is given by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = A^{-1}B = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & -1 & 0 \\ 1 & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ -2 \end{bmatrix}$$

This illustrates that for a system  $AX = B$  where  $A^{-1}$  exists, it is easy to find the solution when the vector  $B$  is changed.



# E. Determinants

---

## E.1 Basic Techniques and Properties

---

### Outcomes

- A. Evaluate the determinant of a square matrix using either Laplace Expansion or row operations.
- B. Demonstrate the effects that row operations have on determinants.
- C. Verify the following:
  - i. The determinant of a product of matrices is the product of the determinants.
  - ii. The determinant of a matrix is equal to the determinant of its transpose.

### E.1.1. Cofactors and $2 \times 2$ Determinants

---

Let  $A$  be an  $n \times n$  matrix. That is, let  $A$  be a square matrix. The **determinant** of  $A$ , denoted by  $\det(A)$  is a very important number which we will explore throughout this section.

If  $A$  is a  $2 \times 2$  matrix, the determinant is given by the following formula.

#### Definition E.1: Determinant of a Two By Two Matrix

Let  $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ . Then

$$\det(A) = ad - cb$$

The determinant is also often denoted by enclosing the matrix with two vertical lines. Thus

$$\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

The following is an example of finding the determinant of a  $2 \times 2$  matrix.

#### Example E.2: A Two by Two Determinant

Find  $\det(A)$  for the matrix  $A = \begin{bmatrix} 2 & 4 \\ -1 & 6 \end{bmatrix}$ .

**Solution.** From Definition E.1,

$$\det(A) = (2)(6) - (-1)(4) = 12 + 4 = 16$$



The  $2 \times 2$  determinant can be used to find the determinant of larger matrices. We will now explore how to find the determinant of a  $3 \times 3$  matrix, using several tools including the  $2 \times 2$  determinant. We begin with the following definition.

### Definition E.3: The $i, j^{\text{th}}$ Minor of a Matrix

Let  $A$  be a  $3 \times 3$  matrix. The  $i, j^{\text{th}}$  **minor** of  $A$ , denoted as  $\text{minor}(A)_{ij}$ , is the determinant of the  $2 \times 2$  matrix which results from deleting the  $i^{\text{th}}$  row and the  $j^{\text{th}}$  column of  $A$ .

In general, if  $A$  is an  $n \times n$  matrix, then the  $i, j^{\text{th}}$  minor of  $A$  is the determinant of the  $(n-1) \times (n-1)$  matrix which results from deleting the  $i^{\text{th}}$  row and the  $j^{\text{th}}$  column of  $A$ .

Hence, there is a minor associated with each entry of  $A$ . Consider the following example which demonstrates this definition.

### Example E.4: Finding Minors of a Matrix

Let

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

Find  $\text{minor}(A)_{12}$  and  $\text{minor}(A)_{23}$ .

**Solution.** First we will find  $\text{minor}(A)_{12}$ . By Definition E.3, this is the determinant of the  $2 \times 2$  matrix which results when you delete the first row and the second column. This minor is given by

$$\text{minor}(A)_{12} = \det \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Using Definition E.1, we see that

$$\det \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix} = (4)(1) - (3)(2) = 4 - 6 = -2$$

Therefore  $\text{minor}(A)_{12} = -2$ .

Similarly,  $\text{minor}(A)_{23}$  is the determinant of the  $2 \times 2$  matrix which results when you delete the second row and the third column. This minor is therefore

$$\text{minor}(A)_{23} = \det \begin{bmatrix} 1 & 2 \\ 3 & 2 \end{bmatrix} = -4$$

Finding the other minors of  $A$  is left as an exercise. ♠

The  $ij^{\text{th}}$  minor of a matrix  $A$  is used in another important definition, given next.

### Definition E.5: The $ij^{\text{th}}$ Cofactor of a Matrix

Suppose  $A$  is an  $n \times n$  matrix. The  $ij^{\text{th}}$  cofactor, denoted by  $\text{cof}(A)_{ij}$ , is defined to be

$$\text{cof}(A)_{ij} = (-1)^{i+j} \text{minor}(A)_{ij}$$

It is also convenient to refer to the cofactor of an entry of a matrix as follows. If  $a_{ij}$  is the  $ij^{\text{th}}$  entry of the matrix, then its cofactor is just  $\text{cof}(A)_{ij}$ .

### Example E.6: Finding Cofactors of a Matrix

Consider the matrix

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

Find  $\text{cof}(A)_{12}$  and  $\text{cof}(A)_{23}$ .

**Solution.** We will use Definition E.5 to compute these cofactors.

First, we will compute  $\text{cof}(A)_{12}$ . Therefore, we need to find  $\text{minor}(A)_{12}$ . This is the determinant of the  $2 \times 2$  matrix which results when you delete the first row and the second column. Thus  $\text{minor}(A)_{12}$  is given by

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

Then,

$$\text{cof}(A)_{12} = (-1)^{1+2} \text{minor}(A)_{12} = (-1)^{1+2} (-2) = 2$$

Hence,  $\text{cof}(A)_{12} = 2$ .

Similarly, we can find  $\text{cof}(A)_{23}$ . First, find  $\text{minor}(A)_{23}$ , which is the determinant of the  $2 \times 2$  matrix which results when you delete the second row and the third column. This minor is therefore

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

Hence,

$$\text{cof}(A)_{23} = (-1)^{2+3} \text{minor}(A)_{23} = (-1)^{2+3} (-4) = 4$$

You may wish to find the remaining cofactors for the above matrix. Remember that there is a cofactor for every entry in the matrix.

We have now established the tools we need to find the determinant of a  $3 \times 3$  matrix.

### Definition E.7: The Determinant of a Three By Three Matrix

Let  $A$  be a  $3 \times 3$  matrix. Then,  $\det(A)$  is calculated by picking a row (or column) and taking the product of each entry in that row (column) with its cofactor and adding these products together.

This process when applied to the  $i^{\text{th}}$  row (column) is known as **expanding along the  $i^{\text{th}}$  row (column)** as is given by

$$\det(A) = a_{i1} \text{cof}(A)_{i1} + a_{i2} \text{cof}(A)_{i2} + a_{i3} \text{cof}(A)_{i3}$$

When calculating the determinant, you can choose to expand any row or any column. Regardless of your choice, you will always get the same number which is the determinant of the matrix  $A$ . This method of evaluating a determinant by expanding along a row or a column is called **Laplace Expansion or Cofactor Expansion**.

Consider the following example.

### Example E.8: Finding the Determinant of a Three by Three Matrix

Let

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

Find  $\det(A)$  using the method of Laplace Expansion.

**Solution.** First, we will calculate  $\det(A)$  by expanding along the first column. Using Definition E.7, we take the 1 in the first column and multiply it by its cofactor,

$$1(-1)^{1+1} \begin{vmatrix} 3 & 2 \\ 2 & 1 \end{vmatrix} = (1)(1)(-1) = -1$$

Similarly, we take the 4 in the first column and multiply it by its cofactor, as well as with the 3 in the first column. Finally, we add these numbers together, as given in the following equation.

$$\det(A) = \underbrace{1(-1)^{1+1} \begin{vmatrix} 3 & 2 \\ 2 & 1 \end{vmatrix}}_{\text{cof}(A)_{11}} + \underbrace{4(-1)^{2+1} \begin{vmatrix} 2 & 3 \\ 2 & 1 \end{vmatrix}}_{\text{cof}(A)_{21}} + \underbrace{3(-1)^{3+1} \begin{vmatrix} 2 & 3 \\ 3 & 2 \end{vmatrix}}_{\text{cof}(A)_{31}}$$

Calculating each of these, we obtain

$$\det(A) = 1(1)(-1) + 4(-1)(-4) + 3(1)(-5) = -1 + 16 - 15 = 0$$

Hence,  $\det(A) = 0$ .

As mentioned in Definition E.7, we can choose to expand along any row or column. Let's try now by expanding along the second row. Here, we take the 4 in the second row and multiply it to its

cofactor, then add this to the 3 in the second row multiplied by its cofactor, and the 2 in the second row multiplied by its cofactor. The calculation is as follows.

$$\det(A) = 4(-1)^{2+1} \overbrace{\begin{vmatrix} 2 & 3 \\ 2 & 1 \end{vmatrix}}^{\text{cof}(A)_{21}} + 3(-1)^{2+2} \overbrace{\begin{vmatrix} 1 & 3 \\ 3 & 1 \end{vmatrix}}^{\text{cof}(A)_{22}} + 2(-1)^{2+3} \overbrace{\begin{vmatrix} 1 & 2 \\ 3 & 2 \end{vmatrix}}^{\text{cof}(A)_{23}}$$

Calculating each of these products, we obtain

$$\det(A) = 4(-1)(-2) + 3(1)(-8) + 2(-1)(-4) = 0$$

You can see that for both methods, we obtained  $\det(A) = 0$ . ♠

As mentioned above, we will always come up with the same value for  $\det(A)$  regardless of the row or column we choose to expand along. You should try to compute the above determinant by expanding along other rows and columns. This is a good way to check your work, because you should come up with the same number each time!

We present this idea formally in the following theorem.

### Theorem E.9: The Determinant is Well Defined

*Expanding the  $n \times n$  matrix along any row or column always gives the same answer, which is the determinant.*

We have now looked at the determinant of  $2 \times 2$  and  $3 \times 3$  matrices. It turns out that the method used to calculate the determinant of a  $3 \times 3$  matrix can be used to calculate the determinant of any sized matrix. Notice that Definition E.3, Definition E.5 and Definition E.7 can all be applied to a matrix of any size.

For example, the  $ij^{th}$  minor of a  $4 \times 4$  matrix is the determinant of the  $3 \times 3$  matrix you obtain when you delete the  $i^{th}$  row and the  $j^{th}$  column. Just as with the  $3 \times 3$  determinant, we can compute the determinant of a  $4 \times 4$  matrix by Laplace Expansion, along any row or column

Consider the following example.

### Example E.10: Determinant of a Four by Four Matrix

Find  $\det(A)$  where

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 4 & 2 & 3 \\ 1 & 3 & 4 & 5 \\ 3 & 4 & 3 & 2 \end{bmatrix}$$

**Solution.** As in the case of a  $3 \times 3$  matrix, you can expand this along any row or column. Lets pick the third column. Then, using Laplace Expansion,

$$\det(A) = 3(-1)^{1+3} \begin{vmatrix} 5 & 4 & 3 \\ 1 & 3 & 5 \\ 3 & 4 & 2 \end{vmatrix} + 2(-1)^{2+3} \begin{vmatrix} 1 & 2 & 4 \\ 1 & 3 & 5 \\ 3 & 4 & 2 \end{vmatrix} +$$

$$4(-1)^{3+3} \begin{vmatrix} 1 & 2 & 4 \\ 5 & 4 & 3 \\ 3 & 4 & 2 \end{vmatrix} + 3(-1)^{4+3} \begin{vmatrix} 1 & 2 & 4 \\ 5 & 4 & 3 \\ 1 & 3 & 5 \end{vmatrix}$$

Now, you can calculate each  $3 \times 3$  determinant using Laplace Expansion, as we did above. You should complete these as an exercise and verify that  $\det(A) = -12$ . ♠

The following provides a formal definition for the determinant of an  $n \times n$  matrix. You may wish to take a moment and consider the above definitions for  $2 \times 2$  and  $3 \times 3$  determinants in context of this definition.

### Definition E.11: The Determinant of an $n \times n$ Matrix

Let  $A$  be an  $n \times n$  matrix where  $n \geq 2$  and suppose the determinant of an  $(n-1) \times (n-1)$  has been defined. Then

$$\det(A) = \sum_{j=1}^n a_{ij} \text{cof}(A)_{ij} = \sum_{i=1}^n a_{ij} \text{cof}(A)_{ij}$$

The first formula consists of expanding the determinant along the  $i^{th}$  row and the second expands the determinant along the  $j^{th}$  column.

In the following sections, we will explore some important properties and characteristics of the determinant.

### E.1.2. The Determinant of a Triangular Matrix

There is a certain type of matrix for which finding the determinant is a very simple procedure. Consider the following definition.

#### Definition E.12: Triangular Matrices

A matrix  $A$  is upper triangular if  $a_{ij} = 0$  whenever  $i > j$ . Thus the entries of such a matrix below the main diagonal equal 0, as shown. Here, \* refers to any nonzero number.

$$\begin{bmatrix} * & * & \cdots & * \\ 0 & * & \cdots & \vdots \\ \vdots & \vdots & \ddots & * \\ 0 & \cdots & 0 & * \end{bmatrix}$$

A lower triangular matrix is defined similarly as a matrix for which all entries above the main diagonal are equal to zero.

The following theorem provides a useful way to calculate the determinant of a triangular matrix.

**Theorem E.13: Determinant of a Triangular Matrix**

Let  $A$  be an upper or lower triangular matrix. Then  $\det(A)$  is obtained by taking the product of the entries on the main diagonal.

The verification of this Theorem can be done by computing the determinant using Laplace Expansion along the first row or column.

Consider the following example.

**Example E.14: Determinant of a Triangular Matrix**

Let

$$A = \begin{bmatrix} 1 & 2 & 3 & 77 \\ 0 & 2 & 6 & 7 \\ 0 & 0 & 3 & 33.7 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Find  $\det(A)$ .

**Solution.** From Theorem E.13, it suffices to take the product of the elements on the main diagonal. Thus  $\det(A) = 1 \times 2 \times 3 \times (-1) = -6$ .

Without using Theorem E.13, you could use Laplace Expansion. We will expand along the first column. This gives

$$\begin{aligned} \det(A) &= 1 \left| \begin{array}{ccc|c} 2 & 6 & 7 & \\ 0 & 3 & 33.7 & \\ 0 & 0 & -1 & \end{array} \right| + 0(-1)^{2+1} \left| \begin{array}{ccc|c} 2 & 3 & 77 & \\ 0 & 3 & 33.7 & \\ 0 & 0 & -1 & \end{array} \right| + \\ &\quad 0(-1)^{3+1} \left| \begin{array}{ccc|c} 2 & 3 & 77 & \\ 2 & 6 & 7 & \\ 0 & 0 & -1 & \end{array} \right| + 0(-1)^{4+1} \left| \begin{array}{ccc|c} 2 & 3 & 77 & \\ 2 & 6 & 7 & \\ 0 & 3 & 33.7 & \end{array} \right| \end{aligned}$$

and the only nonzero term in the expansion is

$$1 \left| \begin{array}{ccc|c} 2 & 6 & 7 & \\ 0 & 3 & 33.7 & \\ 0 & 0 & -1 & \end{array} \right|$$

Now find the determinant of this  $3 \times 3$  matrix, by expanding along the first column to obtain

$$\begin{aligned} \det(A) &= 1 \times \left( 2 \times \left| \begin{array}{cc|c} 3 & 33.7 & \\ 0 & -1 & \end{array} \right| + 0(-1)^{2+1} \left| \begin{array}{cc|c} 6 & 7 & \\ 0 & -1 & \end{array} \right| + 0(-1)^{3+1} \left| \begin{array}{cc|c} 6 & 7 & \\ 3 & 33.7 & \end{array} \right| \right) \\ &= 1 \times 2 \times \left| \begin{array}{cc|c} 3 & 33.7 & \\ 0 & -1 & \end{array} \right| \end{aligned}$$

Next use Definition E.1 to find the determinant of this  $2 \times 2$  matrix, which is just  $3 \times -1 - 0 \times 33.7 = -3$ . Putting all these steps together, we have

$$\det(A) = 1 \times 2 \times 3 \times (-1) = -6$$

which is just the product of the entries down the main diagonal of the original matrix! ♠

You can see that while both methods result in the same answer, Theorem E.13 provides a much quicker method.

In the next section, we explore some important properties of determinants.

## E.2 Applications of the Determinant

---

### Outcomes

- A. Apply Cramer's Rule to solve a  $2 \times 2$  or a  $3 \times 3$  linear system.

### E.2.1. Cramer's Rule

---

Recall that we can represent a system of linear equations in the form  $AX = B$ , where the solutions to this system are given by  $X$ . Cramer's Rule gives a formula for the solutions  $X$  in the special case that  $A$  is a square invertible matrix. Note this rule does not apply if you have a system of equations in which there is a different number of equations than variables (in other words, when  $A$  is not square), or when  $A$  is not invertible.

Suppose we have a system of equations given by  $AX = B$ , and we want to find solutions  $X$  which satisfy this system. Then recall that if  $A^{-1}$  exists,

$$\begin{aligned} AX &= B \\ A^{-1}(AX) &= A^{-1}B \\ (A^{-1}A)X &= A^{-1}B \\ IX &= A^{-1}B \\ X &= A^{-1}B \end{aligned}$$

Hence, the solutions  $X$  to the system are given by  $X = A^{-1}B$ . Since we assume that  $A^{-1}$  exists, we can use the formula for  $A^{-1}$  given above. Substituting this formula into the equation for  $X$ , we have

$$X = A^{-1}B = \frac{1}{\det(A)} \text{adj}(A)B$$

Let  $x_i$  be the  $i^{th}$  entry of  $X$  and  $b_j$  be the  $j^{th}$  entry of  $B$ . Then this equation becomes

$$x_i = \sum_{j=1}^n [a_{ij}]^{-1} b_j = \sum_{j=1}^n \frac{1}{\det(A)} \text{adj}(A)_{ij} b_j$$

where  $\text{adj}(A)_{ij}$  is the  $ij^{th}$  entry of  $\text{adj}(A)$ .

By the formula for the expansion of a determinant along a column,

$$x_i = \frac{1}{\det(A)} \det \begin{bmatrix} * & \cdots & b_1 & \cdots & * \\ \vdots & & \vdots & & \vdots \\ * & \cdots & b_n & \cdots & * \end{bmatrix}$$

where here the  $i^{th}$  column of  $A$  is replaced with the column vector  $[b_1 \cdots, b_n]^T$ . The determinant of this modified matrix is taken and divided by  $\det(A)$ . This formula is known as Cramer's rule.

We formally define this method now.

### Procedure E.15: Using Cramer's Rule

Suppose  $A$  is an  $n \times n$  invertible matrix and we wish to solve the system  $AX = B$  for  $X = [x_1, \dots, x_n]^T$ . Then Cramer's rule says

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where  $A_i$  is the matrix obtained by replacing the  $i^{th}$  column of  $A$  with the column matrix

$$B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

We illustrate this procedure in the following example.

### Example E.16: Using Cramer's Rule

Find  $x, y, z$  if

$$\begin{bmatrix} 1 & 2 & 1 \\ 3 & 2 & 1 \\ 2 & -3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

**Solution.** We will use method outlined in Procedure E.15 to find the values for  $x, y, z$  which give the solution to this system. Let

$$B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

In order to find  $x$ , we calculate

$$x = \frac{\det(A_1)}{\det(A)}$$

where  $A_1$  is the matrix obtained from replacing the first column of  $A$  with  $B$ .

Hence,  $A_1$  is given by

$$A_1 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 1 \\ 3 & -3 & 2 \end{bmatrix}$$

Therefore,

$$x = \frac{\det(A_1)}{\det(A)} = \frac{\begin{vmatrix} 1 & 2 & 1 \\ 2 & 2 & 1 \\ 3 & -3 & 2 \end{vmatrix}}{\begin{vmatrix} 1 & 2 & 1 \\ 3 & 2 & 1 \\ 2 & -3 & 2 \end{vmatrix}} = \frac{1}{2}$$

Similarly, to find  $y$  we construct  $A_2$  by replacing the second column of  $A$  with  $B$ . Hence,  $A_2$  is given by

$$A_2 = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 2 & 1 \\ 2 & 3 & 2 \end{bmatrix}$$

Therefore,

$$y = \frac{\det(A_2)}{\det(A)} = \frac{\begin{vmatrix} 1 & 1 & 1 \\ 3 & 2 & 1 \\ 2 & 3 & 2 \end{vmatrix}}{\begin{vmatrix} 1 & 2 & 1 \\ 3 & 2 & 1 \\ 2 & -3 & 2 \end{vmatrix}} = -\frac{1}{7}$$

Similarly,  $A_3$  is constructed by replacing the third column of  $A$  with  $B$ . Then,  $A_3$  is given by

$$A_3 = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 2 & 2 \\ 2 & -3 & 3 \end{bmatrix}$$

Therefore,  $z$  is calculated as follows.

$$z = \frac{\det(A_3)}{\det(A)} = \frac{\begin{vmatrix} 1 & 2 & 1 \\ 3 & 2 & 2 \\ 2 & -3 & 3 \end{vmatrix}}{\begin{vmatrix} 1 & 2 & 1 \\ 3 & 2 & 1 \\ 2 & -3 & 2 \end{vmatrix}} = \frac{11}{14}$$



Cramer's Rule gives you another tool to consider when solving a system of linear equations.

## F. $\mathbb{R}^n$

---

### F.1 Vectors in $\mathbb{R}^n$

---

#### Outcomes

A. Find the position vector of a point in  $\mathbb{R}^n$ .

The notation  $\mathbb{R}^n$  refers to the collection of ordered lists of  $n$  real numbers, that is

$$\mathbb{R}^n = \{(x_1 \cdots x_n) : x_j \in \mathbb{R} \text{ for } j = 1, \dots, n\}$$

In this chapter, we take a closer look at vectors in  $\mathbb{R}^n$ . First, we will consider what  $\mathbb{R}^n$  looks like in more detail. Recall that the point given by  $0 = (0, \dots, 0)$  is called the **origin**.

Now, consider the case of  $\mathbb{R}^n$  for  $n = 1$ . Then from the definition we can identify  $\mathbb{R}$  with points in  $\mathbb{R}^1$  as follows:

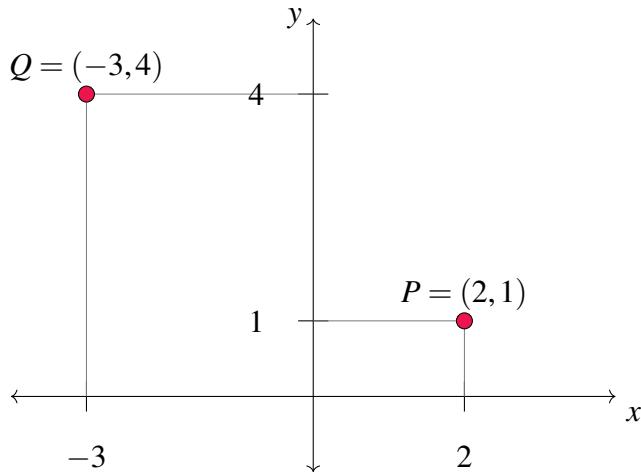
$$\mathbb{R} = \mathbb{R}^1 = \{(x_1) : x_1 \in \mathbb{R}\}$$

Hence,  $\mathbb{R}$  is defined as the set of all real numbers and geometrically, we can describe this as all the points on a line.

Now suppose  $n = 2$ . Then, from the definition,

$$\mathbb{R}^2 = \{(x_1, x_2) : x_j \in \mathbb{R} \text{ for } j = 1, 2\}$$

Consider the familiar coordinate plane, with an  $x$  axis and a  $y$  axis. Any point within this coordinate plane is identified by where it is located along the  $x$  axis, and also where it is located along the  $y$  axis. Consider as an example the following diagram.



Hence, every element in  $\mathbb{R}^2$  is identified by two components,  $x$  and  $y$ , in the usual manner. The coordinates  $x, y$  (or  $x_1, x_2$ ) uniquely determine a point in the plane. Note that while the definition uses  $x_1$  and  $x_2$  to label the coordinates and you may be used to  $x$  and  $y$ , these notations are equivalent.

Now suppose  $n = 3$ . You may have previously encountered the 3-dimensional coordinate system, given by

$$\mathbb{R}^3 = \{(x_1, x_2, x_3) : x_j \in \mathbb{R} \text{ for } j = 1, 2, 3\}$$

Points in  $\mathbb{R}^3$  will be determined by three coordinates, often written  $(x, y, z)$  which correspond to the  $x$ ,  $y$ , and  $z$  axes. We can think as above that the first two coordinates determine a point in a plane. The third component determines the height above or below the plane, depending on whether this number is positive or negative, and all together this determines a point in space. You see that the ordered triples correspond to points in space just as the ordered pairs correspond to points in a plane and single real numbers correspond to points on a line.

The idea behind the more general  $\mathbb{R}^n$  is that we can extend these ideas beyond  $n = 3$ . This discussion regarding points in  $\mathbb{R}^n$  leads into a study of vectors in  $\mathbb{R}^n$ . While we consider  $\mathbb{R}^n$  for all  $n$ , we will largely focus on  $n = 2, 3$  in this section.

Consider the following definition.

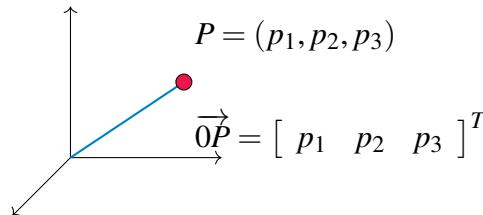
### Definition F.1: The Position Vector

Let  $P = (p_1, \dots, p_n)$  be the coordinates of a point in  $\mathbb{R}^n$ . Then the vector  $\overrightarrow{0P}$  with its tail at  $0 = (0, \dots, 0)$  and its tip at  $P$  is called the **position vector** of the point  $P$ . We write

$$\overrightarrow{0P} = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}$$

For this reason we may write both  $P = (p_1, \dots, p_n) \in \mathbb{R}^n$  and  $\overrightarrow{0P} = [p_1 \dots p_n]^T \in \mathbb{R}^n$ .

This definition is illustrated in the following picture for the special case of  $\mathbb{R}^3$ .

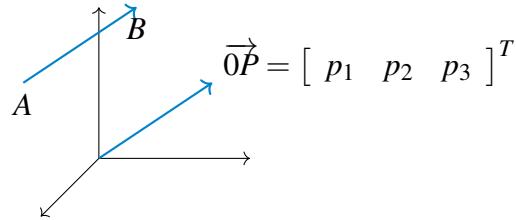


Thus every point  $P$  in  $\mathbb{R}^n$  determines its position vector  $\overrightarrow{0P}$ . Conversely, every such position vector  $\overrightarrow{0P}$  which has its tail at  $0$  and point at  $P$  determines the point  $P$  of  $\mathbb{R}^n$ .

Now suppose we are given two points,  $P, Q$  whose coordinates are  $(p_1, \dots, p_n)$  and  $(q_1, \dots, q_n)$  respectively. We can also determine the **position vector from  $P$  to  $Q$**  (also called the **vector from  $P$  to  $Q$** ) defined as follows.

$$\overrightarrow{PQ} = \begin{bmatrix} q_1 - p_1 \\ \vdots \\ q_n - p_n \end{bmatrix} = \overrightarrow{0Q} - \overrightarrow{0P}$$

Now, imagine taking a vector in  $\mathbb{R}^n$  and moving it around, always keeping it pointing in the same direction as shown in the following picture.



After moving it around, it is regarded as the same vector. Each vector,  $\overrightarrow{OP}$  and  $\overrightarrow{AB}$  has the same length (or magnitude) and direction. Therefore, they are equal.

Consider now the general definition for a vector in  $\mathbb{R}^n$ .

### Definition F.2: Vectors in $\mathbb{R}^n$

Let  $\mathbb{R}^n = \{(x_1, \dots, x_n) : x_j \in \mathbb{R} \text{ for } j = 1, \dots, n\}$ . Then,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

is called a **vector**. Vectors have both size (magnitude) and direction. The numbers  $x_j$  are called the **components** of  $\mathbf{x}$ .

Using this notation, we may use  $\mathbf{p}$  to denote the position vector of point  $P$ . Notice that in this context,  $\mathbf{p} = \overrightarrow{OP}$ . These notations may be used interchangeably.

You can think of the components of a vector as directions for obtaining the vector. Consider  $n = 3$ . Draw a vector with its tail at the point  $(0,0,0)$  and its tip at the point  $(a,b,c)$ . This vector it is obtained by starting at  $(0,0,0)$ , moving parallel to the  $x$  axis to  $(a,0,0)$  and then from here, moving parallel to the  $y$  axis to  $(a,b,0)$  and finally parallel to the  $z$  axis to  $(a,b,c)$ . Observe that the same vector would result if you began at the point  $(d,e,f)$ , moved parallel to the  $x$  axis to  $(d+a,e,f)$ , then parallel to the  $y$  axis to  $(d+a,e+b,f)$ , and finally parallel to the  $z$  axis to  $(d+a,e+b,f+c)$ . Here, the vector would have its tail sitting at the point determined by  $A = (d,e,f)$  and its point at  $B = (d+a,e+b,f+c)$ . It is the **same vector** because it will point in the same direction and have the same length. It is like you took an actual arrow, and moved it from one location to another keeping it pointing the same direction.

We conclude this section with a brief discussion regarding notation. In previous sections, we have written vectors as columns, or  $n \times 1$  matrices. For convenience in this chapter we may write vectors as the transpose of row vectors, or  $1 \times n$  matrices. These are of course equivalent and we may move between both notations. Therefore, recognize that

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} = [2 \ 3]^T$$

Notice that two vectors  $\mathbf{u} = [u_1 \dots u_n]^T$  and  $\mathbf{v} = [v_1 \dots v_n]^T$  are equal if and only if all corresponding

components are equal. Precisely,

$$\begin{aligned}\mathbf{u} &= \mathbf{v} \text{ if and only if} \\ u_j &= v_j \text{ for all } j = 1, \dots, n\end{aligned}$$

Thus  $[1 \ 2 \ 4]^T \in \mathbb{R}^3$  and  $[2 \ 1 \ 4]^T \in \mathbb{R}^3$  but  $[1 \ 2 \ 4]^T \neq [2 \ 1 \ 4]^T$  because, even though the same numbers are involved, the order of the numbers is different.

For the specific case of  $\mathbb{R}^3$ , there are three special vectors which we often use. They are given by

$$\mathbf{i} = [1 \ 0 \ 0]^T$$

$$\mathbf{j} = [0 \ 1 \ 0]^T$$

$$\mathbf{k} = [0 \ 0 \ 1]^T$$

We can write any vector  $\mathbf{u} = [u_1 \ u_2 \ u_3]^T$  as a linear combination of these vectors, written as  $\mathbf{u} = u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k}$ . This notation will be used throughout this chapter.

## F.2 Algebra in $\mathbb{R}^n$

---

### Outcomes

- A. Understand vector addition and scalar multiplication, algebraically.
- B. Introduce the notion of linear combination of vectors.

Addition and scalar multiplication are two important algebraic operations done with vectors. Notice that these operations apply to vectors in  $\mathbb{R}^n$ , for any value of  $n$ . We will explore these operations in more detail in the following sections.

### F.2.1. Addition of Vectors in $\mathbb{R}^n$

---

Addition of vectors in  $\mathbb{R}^n$  is defined as follows.

**Definition F.3: Addition of Vectors in  $\mathbb{R}^n$** 

If  $\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$ ,  $\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \in \mathbb{R}^n$  then  $\mathbf{u} + \mathbf{v} \in \mathbb{R}^n$  and is defined by

$$\begin{aligned}\mathbf{u} + \mathbf{v} &= \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \\ &= \begin{bmatrix} u_1 + v_1 \\ \vdots \\ u_n + v_n \end{bmatrix}\end{aligned}$$

To add vectors, we simply add corresponding components. Therefore, in order to add vectors, they must be the same size.

Addition of vectors satisfies some important properties which are outlined in the following theorem.

**Theorem F.4: Properties of Vector Addition**

The following properties hold for vectors  $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ .

- The Commutative Law of Addition

$$\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$$

- The Associative Law of Addition

$$(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$$

- The Existence of an Additive Identity

$$\mathbf{u} + \mathbf{0} = \mathbf{u} \tag{6.1}$$

- The Existence of an Additive Inverse

$$\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$$

The additive identity shown in equation 6.1 is also called the **zero vector**, the  $n \times 1$  vector in which all components are equal to 0. Further,  $-\mathbf{u}$  is simply the vector with all components having same value as those of  $\mathbf{u}$  but opposite sign; this is just  $(-1)\mathbf{u}$ . This will be made more explicit in the next section when we explore scalar multiplication of vectors. Note that subtraction is defined as  $\mathbf{u} - \mathbf{v} = \mathbf{u} + (-\mathbf{v})$ .

## F.2.2. Scalar Multiplication of Vectors in $\mathbb{R}^n$

Scalar multiplication of vectors in  $\mathbb{R}^n$  is defined as follows.

### Definition F.5: Scalar Multiplication of Vectors in $\mathbb{R}^n$

If  $\mathbf{u} \in \mathbb{R}^n$  and  $k \in \mathbb{R}$  is a scalar, then  $k\mathbf{u} \in \mathbb{R}^n$  is defined by

$$k\mathbf{u} = k \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} ku_1 \\ \vdots \\ ku_n \end{bmatrix}$$

Just as with addition, scalar multiplication of vectors satisfies several important properties. These are outlined in the following theorem.

### Theorem F.6: Properties of Scalar Multiplication

The following properties hold for vectors  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$  and  $k, p$  scalars.

- The Distributive Law over Vector Addition

$$k(\mathbf{u} + \mathbf{v}) = k\mathbf{u} + k\mathbf{v}$$

- The Distributive Law over Scalar Addition

$$(k + p)\mathbf{u} = k\mathbf{u} + p\mathbf{u}$$

- The Associative Law for Scalar Multiplication

$$k(p\mathbf{u}) = (kp)\mathbf{u}$$

- Rule for Multiplication by 1

$$1\mathbf{u} = \mathbf{u}$$

We now present a useful notion you may have seen earlier combining vector addition and scalar multiplication

### Definition F.7: Linear Combination

A vector  $\mathbf{v}$  is said to be a **linear combination** of the vectors  $\mathbf{u}_1, \dots, \mathbf{u}_n$  if there exist scalars,  $a_1, \dots, a_n$  such that

$$\mathbf{v} = a_1\mathbf{u}_1 + \dots + a_n\mathbf{u}_n$$

For example,

$$3 \begin{bmatrix} -4 \\ 1 \\ 0 \end{bmatrix} + 2 \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -18 \\ 3 \\ 2 \end{bmatrix}.$$

Thus we can say that

$$\mathbf{v} = \begin{bmatrix} -18 \\ 3 \\ 2 \end{bmatrix}$$

is a linear combination of the vectors

$$\mathbf{u}_1 = \begin{bmatrix} -4 \\ 1 \\ 0 \end{bmatrix} \text{ and } \mathbf{u}_2 = \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix}$$

## F.3 Geometric Meaning of Vector Addition

---

### Outcomes

A. Understand vector addition, geometrically.

Recall that an element of  $\mathbb{R}^n$  is an ordered list of numbers. For the specific case of  $n = 2, 3$  this can be used to determine a point in two or three dimensional space. This point is specified relative to some coordinate axes.

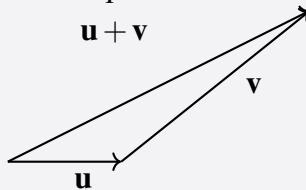
Consider the case  $n = 3$ . Recall that taking a vector and moving it around without changing its length or direction does not change the vector. This is important in the geometric representation of vector addition.

Suppose we have two vectors,  $\mathbf{u}$  and  $\mathbf{v}$  in  $\mathbb{R}^3$ . Each of these can be drawn geometrically by placing the tail of each vector at 0 and its point at  $(u_1, u_2, u_3)$  and  $(v_1, v_2, v_3)$  respectively. Suppose we slide the vector  $\mathbf{v}$  so that its tail sits at the point of  $\mathbf{u}$ . We know that this does not change the vector  $\mathbf{v}$ . Now, draw a new vector from the tail of  $\mathbf{u}$  to the point of  $\mathbf{v}$ . This vector is  $\mathbf{u} + \mathbf{v}$ .

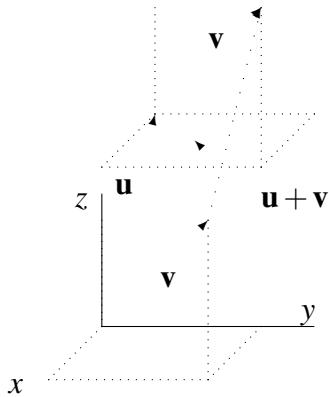
The geometric significance of vector addition in  $\mathbb{R}^n$  for any  $n$  is given in the following definition.

### Definition F.8: Geometry of Vector Addition

*Let  $\mathbf{u}$  and  $\mathbf{v}$  be two vectors. Slide  $\mathbf{v}$  so that the tail of  $\mathbf{v}$  is on the point of  $\mathbf{u}$ . Then draw the arrow which goes from the tail of  $\mathbf{u}$  to the point of  $\mathbf{v}$ . This arrow represents the vector  $\mathbf{u} + \mathbf{v}$ .*



This definition is illustrated in the following picture in which  $\mathbf{u} + \mathbf{v}$  is shown for the special case  $n = 3$ .



Notice the parallelogram created by  $\mathbf{u}$  and  $\mathbf{v}$  in the above diagram. Then  $\mathbf{u} + \mathbf{v}$  is the directed diagonal of the parallelogram determined by the two vectors  $\mathbf{u}$  and  $\mathbf{v}$ .

When you have a vector  $\mathbf{v}$ , its additive inverse  $-\mathbf{v}$  will be the vector which has the same magnitude as  $\mathbf{v}$  but the opposite direction. When one writes  $\mathbf{u} - \mathbf{v}$ , the meaning is  $\mathbf{u} + (-\mathbf{v})$  as with real numbers. The following example illustrates these definitions and conventions.

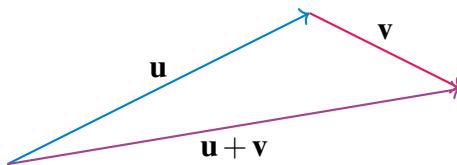
### Example F.9: Graphing Vector Addition

Consider the following picture of vectors  $\mathbf{u}$  and  $\mathbf{v}$ .

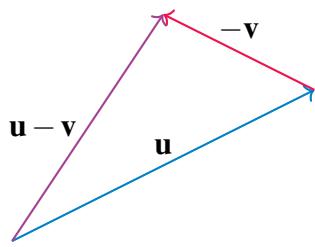


Sketch a picture of  $\mathbf{u} + \mathbf{v}$ ,  $\mathbf{u} - \mathbf{v}$ .

**Solution.** We will first sketch  $\mathbf{u} + \mathbf{v}$ . Begin by drawing  $\mathbf{u}$  and then at the point of  $\mathbf{u}$ , place the tail of  $\mathbf{v}$  as shown. Then  $\mathbf{u} + \mathbf{v}$  is the vector which results from drawing a vector from the tail of  $\mathbf{u}$  to the tip of  $\mathbf{v}$ .



Next consider  $\mathbf{u} - \mathbf{v}$ . This means  $\mathbf{u} + (-\mathbf{v})$ . From the above geometric description of vector addition,  $-\mathbf{v}$  is the vector which has the same length but which points in the opposite direction to  $\mathbf{v}$ . Here is a picture.



## F.4 Length of a Vector

---

### Outcomes

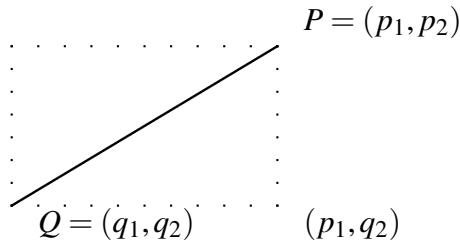
- A. Find the length of a vector and the distance between two points in  $\mathbb{R}^n$ .
- B. Find the corresponding unit vector to a vector in  $\mathbb{R}^n$ .

In this section, we explore what is meant by the length of a vector in  $\mathbb{R}^n$ . We develop this concept by first looking at the distance between two points in  $\mathbb{R}^n$ .

First, we will consider the concept of distance for  $\mathbb{R}$ , that is, for points in  $\mathbb{R}^1$ . Here, the distance between two points  $P$  and  $Q$  is given by the absolute value of their difference. We denote the distance between  $P$  and  $Q$  by  $d(P, Q)$  which is defined as

$$d(P, Q) = \sqrt{(P - Q)^2} \quad (6.1)$$

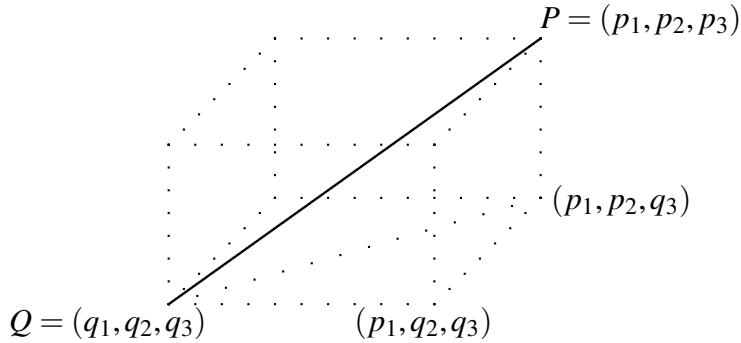
Consider now the case for  $n = 2$ , demonstrated by the following picture.



There are two points  $P = (p_1, p_2)$  and  $Q = (q_1, q_2)$  in the plane. The distance between these points is shown in the picture as a solid line. Notice that this line is the hypotenuse of a right triangle which is half of the rectangle shown in dotted lines. We want to find the length of this hypotenuse which will give the distance between the two points. Note the lengths of the sides of this triangle are  $|p_1 - q_1|$  and  $|p_2 - q_2|$ , the absolute value of the difference in these values. Therefore, the Pythagorean Theorem implies the length of the hypotenuse (and thus the distance between  $P$  and  $Q$ ) equals

$$\left( |p_1 - q_1|^2 + |p_2 - q_2|^2 \right)^{1/2} = \left( (p_1 - q_1)^2 + (p_2 - q_2)^2 \right)^{1/2} \quad (6.2)$$

Now suppose  $n = 3$  and let  $P = (p_1, p_2, p_3)$  and  $Q = (q_1, q_2, q_3)$  be two points in  $\mathbb{R}^3$ . Consider the following picture in which the solid line joins the two points and a dotted line joins the points  $(q_1, q_2, q_3)$  and  $(p_1, p_2, q_3)$ .



Here, we need to use Pythagorean Theorem twice in order to find the length of the solid line. First, by the Pythagorean Theorem, the length of the dotted line joining  $(q_1, q_2, q_3)$  and  $(p_1, p_2, q_3)$  equals

$$\left( (p_1 - q_1)^2 + (p_2 - q_2)^2 \right)^{1/2}$$

while the length of the line joining  $(p_1, p_2, q_3)$  to  $(p_1, p_2, p_3)$  is just  $|p_3 - q_3|$ . Therefore, by the Pythagorean Theorem again, the length of the line joining the points  $P = (p_1, p_2, p_3)$  and  $Q = (q_1, q_2, q_3)$  equals

$$\begin{aligned} & \left( \left( \left( (p_1 - q_1)^2 + (p_2 - q_2)^2 \right)^{1/2} \right)^2 + (p_3 - q_3)^2 \right)^{1/2} \\ &= \left( (p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2 \right)^{1/2} \end{aligned} \quad (6.3)$$

This discussion motivates the following definition for the distance between points in  $\mathbb{R}^n$ .

#### Definition F.10: Distance Between Points

Let  $P = (p_1, \dots, p_n)$  and  $Q = (q_1, \dots, q_n)$  be two points in  $\mathbb{R}^n$ . Then the distance between these points is defined as

$$\text{distance between } P \text{ and } Q = d(P, Q) = \left( \sum_{k=1}^n |p_k - q_k|^2 \right)^{1/2}$$

This is called the **distance formula**. We may also write  $|P - Q|$  as the distance between  $P$  and  $Q$ .

From the above discussion, you can see that Definition F.10 holds for the special cases  $n = 1, 2, 3$ , as in Equations 6.1, 6.2, 6.3. In the following example, we use Definition F.10 to find the distance between two points in  $\mathbb{R}^4$ .

#### Example F.11: Distance Between Points

Find the distance between the points  $P$  and  $Q$  in  $\mathbb{R}^4$ , where  $P$  and  $Q$  are given by

$$P = (1, 2, -4, 6)$$

and

$$Q = (2, 3, -1, 0)$$

**Solution.** We will use the formula given in Definition F.10 to find the distance between  $P$  and  $Q$ . Use the distance formula and write

$$d(P, Q) = \left( (1-2)^2 + (2-3)^2 + (-4-(-1))^2 + (6-0)^2 \right)^{\frac{1}{2}} = 47$$

Therefore,  $d(P, Q) = \sqrt{47}$ .



There are certain properties of the distance between points which are important in our study. These are outlined in the following theorem.

### Theorem F.12: Properties of Distance

Let  $P$  and  $Q$  be points in  $\mathbb{R}^n$ , and let the distance between them,  $d(P, Q)$ , be given as in Definition F.10. Then, the following properties hold .

- $d(P, Q) = d(Q, P)$
- $d(P, Q) \geq 0$ , and equals 0 exactly when  $P = Q$ .

There are many applications of the concept of distance. For instance, given two points, we can ask what collection of points are all the same distance between the given points. This is explored in the following example.

### Example F.13: The Plane Between Two Points

Describe the points in  $\mathbb{R}^3$  which are at the same distance between  $(1, 2, 3)$  and  $(0, 1, 2)$ .

**Solution.** Let  $P = (p_1, p_2, p_3)$  be such a point. Therefore,  $P$  is the same distance from  $(1, 2, 3)$  and  $(0, 1, 2)$ . Then by Definition F.10,

$$\sqrt{(p_1 - 1)^2 + (p_2 - 2)^2 + (p_3 - 3)^2} = \sqrt{(p_1 - 0)^2 + (p_2 - 1)^2 + (p_3 - 2)^2}$$

Squaring both sides we obtain

$$(p_1 - 1)^2 + (p_2 - 2)^2 + (p_3 - 3)^2 = p_1^2 + (p_2 - 1)^2 + (p_3 - 2)^2$$

and so

$$p_1^2 - 2p_1 + 14 + p_2^2 - 4p_2 + p_3^2 - 6p_3 = p_1^2 + p_2^2 - 2p_2 + 5 + p_3^2 - 4p_3$$

Simplifying, this becomes

$$-2p_1 + 14 - 4p_2 - 6p_3 = -2p_2 + 5 - 4p_3$$

which can be written as

$$2p_1 + 2p_2 + 2p_3 = -9 \quad (6.4)$$

Therefore, the points  $P = (p_1, p_2, p_3)$  which are the same distance from each of the given points form a plane whose equation is given by 6.4. ♠

We can now use our understanding of the distance between two points to define what is meant by the length of a vector. Consider the following definition.

#### Definition F.14: Length of a Vector

*Let  $\mathbf{u} = [u_1 \cdots u_n]^T$  be a vector in  $\mathbb{R}^n$ . Then, the length of  $\mathbf{u}$ , written  $\|\mathbf{u}\|$  is given by*

$$\|\mathbf{u}\| = \sqrt{u_1^2 + \cdots + u_n^2}$$

This definition corresponds to Definition F.10, if you consider the vector  $\mathbf{u}$  to have its tail at the point  $0 = (0, \dots, 0)$  and its tip at the point  $U = (u_1, \dots, u_n)$ . Then the length of  $\mathbf{u}$  is equal to the distance between  $0$  and  $U$ ,  $d(0, U)$ . In general,  $d(P, Q) = \|\overrightarrow{PQ}\|$ .

Consider Example F.11. By Definition F.14, we could also find the distance between  $P$  and  $Q$  as the length of the vector connecting them. Hence, if we were to draw a vector  $\overrightarrow{PQ}$  with its tail at  $P$  and its point at  $Q$ , this vector would have length equal to  $\sqrt{47}$ .

We conclude this section with a new definition for the special case of vectors of length 1.

#### Definition F.15: Unit Vector

*Let  $\mathbf{u}$  be a vector in  $\mathbb{R}^n$ . Then, we call  $\mathbf{u}$  a **unit vector** if it has length 1, that is if*

$$\|\mathbf{u}\| = 1$$

Let  $\mathbf{v}$  be a vector in  $\mathbb{R}^n$ . Then, the vector  $\mathbf{u}$  which has the same direction as  $\mathbf{v}$  but length equal to 1 is the corresponding unit vector of  $\mathbf{v}$ . This vector is given by

$$\mathbf{u} = \frac{1}{\|\mathbf{v}\|} \mathbf{v}$$

We often use the term **normalize** to refer to this process. When we **normalize** a vector, we find the corresponding unit vector of length 1. Consider the following example.

#### Example F.16: Finding a Unit Vector

*Let  $\mathbf{v}$  be given by*

$$\mathbf{v} = [ 1 \ -3 \ 4 ]^T$$

*Find the unit vector  $\mathbf{u}$  which has the same direction as  $\mathbf{v}$ .*

**Solution.** We will use Definition F.15 to solve this. Therefore, we need to find the length of  $\mathbf{v}$  which, by Definition F.14 is given by

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + v_3^2}$$

Using the corresponding values we find that

$$\begin{aligned}\|\mathbf{v}\| &= \sqrt{1^2 + (-3)^2 + 4^2} \\ &= \sqrt{1 + 9 + 16} \\ &= \sqrt{26}\end{aligned}$$

In order to find  $\mathbf{u}$ , we divide  $\mathbf{v}$  by  $\sqrt{26}$ . The result is

$$\begin{aligned}\mathbf{u} &= \frac{1}{\|\mathbf{v}\|} \mathbf{v} \\ &= \frac{1}{\sqrt{26}} [ 1 \ -3 \ 4 ]^T \\ &= \left[ \frac{1}{\sqrt{26}} \ -\frac{3}{\sqrt{26}} \ \frac{4}{\sqrt{26}} \right]^T\end{aligned}$$

You can verify using the Definition F.14 that  $\|\mathbf{u}\| = 1$ .



## F.5 Geometric Meaning of Scalar Multiplication

---

### Outcomes

A. Understand scalar multiplication, geometrically.

Recall that the point  $P = (p_1, p_2, p_3)$  determines a vector  $\mathbf{p}$  from 0 to  $P$ . The length of  $\mathbf{p}$ , denoted  $\|\mathbf{p}\|$ , is equal to  $\sqrt{p_1^2 + p_2^2 + p_3^2}$  by Definition F.10.

Now suppose we have a vector  $\mathbf{u} = [ u_1 \ u_2 \ u_3 ]^T$  and we multiply  $\mathbf{u}$  by a scalar  $k$ . By Definition F.5,  $k\mathbf{u} = [ ku_1 \ ku_2 \ ku_3 ]^T$ . Then, by using Definition F.10, the length of this vector is given by

$$\sqrt{(ku_1)^2 + (ku_2)^2 + (ku_3)^2} = |k| \sqrt{u_1^2 + u_2^2 + u_3^2}$$

Thus the following holds.

$$\|k\mathbf{u}\| = |k| \|\mathbf{u}\|$$

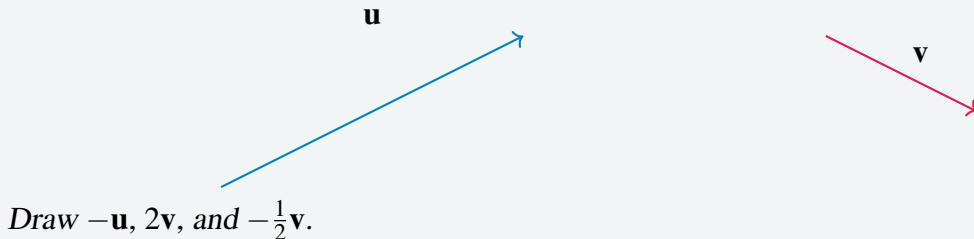
In other words, multiplication by a scalar magnifies or shrinks the length of the vector by a factor of  $|k|$ . If  $|k| > 1$ , the length of the resulting vector will be magnified. If  $|k| < 1$ , the length of the resulting vector will shrink. Remember that by the definition of the absolute value,  $|k| > 0$ .

What about the direction? Draw a picture of  $\mathbf{u}$  and  $k\mathbf{u}$  where  $k$  is negative. Notice that this causes the resulting vector to point in the opposite direction while if  $k > 0$  it preserves the direction the vector points. Therefore the direction can either reverse, if  $k < 0$ , or remain preserved, if  $k > 0$ .

Consider the following example.

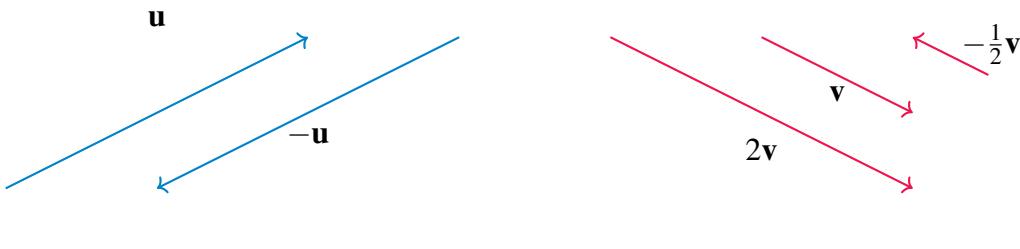
### Example F.17: Graphing Scalar Multiplication

Consider the vectors  $\mathbf{u}$  and  $\mathbf{v}$  drawn below.



### Solution.

In order to find  $-\mathbf{u}$ , we preserve the length of  $\mathbf{u}$  and simply reverse the direction. For  $2\mathbf{v}$ , we double the length of  $\mathbf{v}$ , while preserving the direction. Finally  $-\frac{1}{2}\mathbf{v}$  is found by taking half the length of  $\mathbf{v}$  and reversing the direction. These vectors are shown in the following diagram.

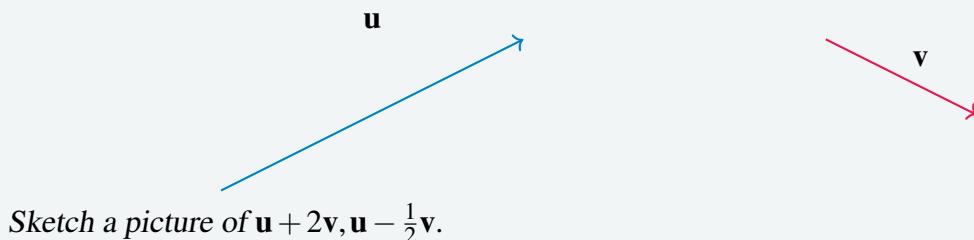


Now that we have studied both vector addition and scalar multiplication, we can combine the two actions. Recall Definition F.7 of linear combinations of column matrices. We can apply this definition to vectors in  $\mathbb{R}^n$ . A linear combination of vectors in  $\mathbb{R}^n$  is a sum of vectors multiplied by scalars.

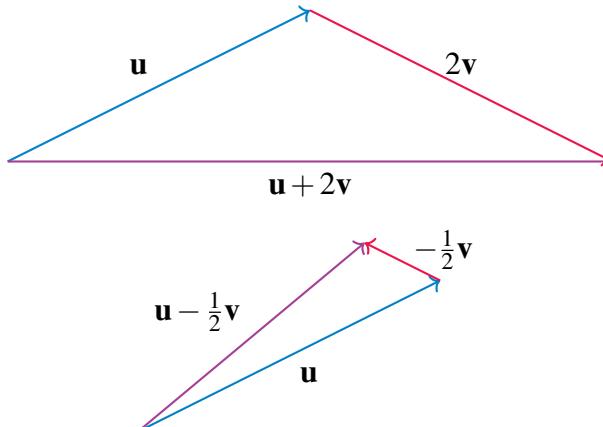
In the following example, we examine the geometric meaning of this concept.

### Example F.18: Graphing a Linear Combination of Vectors

Consider the following picture of the vectors  $\mathbf{u}$  and  $\mathbf{v}$



**Solution.** The two vectors are shown below.



## F.6 The Dot Product

---

### Outcomes

A. Compute the dot product of vectors, and use this to compute vector projections.

### F.6.1. The Dot Product

---

There are two ways of multiplying vectors which are of great importance in applications. The first of these is called the **dot product**. When we take the dot product of vectors, the result is a scalar. For this reason, the dot product is also called the **scalar product** and sometimes the **inner product**. The definition is as follows.

#### Definition F.19: Dot Product

Let  $\mathbf{u}, \mathbf{v}$  be two vectors in  $\mathbb{R}^n$ . Then we define the **dot product**  $\mathbf{u} \bullet \mathbf{v}$  as

$$\mathbf{u} \bullet \mathbf{v} = \sum_{k=1}^n u_k v_k$$

The dot product  $\mathbf{u} \bullet \mathbf{v}$  is sometimes denoted as  $(\mathbf{u}, \mathbf{v})$  where a comma replaces  $\bullet$ . It can also be written as  $\langle \mathbf{u}, \mathbf{v} \rangle$ . If we write the vectors as column or row matrices, it is equal to the matrix product  $\mathbf{v}\mathbf{w}^T$ .

Consider the following example.

**Example F.20: Compute a Dot Product**

Find  $\mathbf{u} \bullet \mathbf{v}$  for

$$\mathbf{u} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ -1 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix}$$

**Solution.** By Definition F.19, we must compute

$$\mathbf{u} \bullet \mathbf{v} = \sum_{k=1}^4 u_k v_k$$

This is given by

$$\begin{aligned} \mathbf{u} \bullet \mathbf{v} &= (1)(0) + (2)(1) + (0)(2) + (-1)(3) \\ &= 0 + 2 + 0 + -3 \\ &= -1 \end{aligned}$$



With this definition, there are several important properties satisfied by the dot product.

**Proposition F.21: Properties of the Dot Product**

Let  $k$  and  $p$  denote scalars and  $\mathbf{u}, \mathbf{v}, \mathbf{w}$  denote vectors. Then the dot product  $\mathbf{u} \bullet \mathbf{v}$  satisfies the following properties.

- $\mathbf{u} \bullet \mathbf{v} = \mathbf{v} \bullet \mathbf{u}$
- $\mathbf{u} \bullet \mathbf{u} \geq 0$  and equals zero if and only if  $\mathbf{u} = 0$
- $(k\mathbf{u} + p\mathbf{v}) \bullet \mathbf{w} = k(\mathbf{u} \bullet \mathbf{w}) + p(\mathbf{v} \bullet \mathbf{w})$
- $\mathbf{u} \bullet (k\mathbf{v} + p\mathbf{w}) = k(\mathbf{u} \bullet \mathbf{v}) + p(\mathbf{u} \bullet \mathbf{w})$
- $\|\mathbf{u}\|^2 = \mathbf{u} \bullet \mathbf{u}$

The proof is left as an exercise. This proposition tells us that we can also use the dot product to find the length of a vector.

**Example F.22: Length of a Vector**

Find the length of

$$\mathbf{u} = \begin{bmatrix} 2 \\ 1 \\ 4 \\ 2 \end{bmatrix}$$

That is, find  $\|\mathbf{u}\|$ .

**Solution.** By Proposition F.21,  $\|\mathbf{u}\|^2 = \mathbf{u} \bullet \mathbf{u}$ . Therefore,  $\|\mathbf{u}\| = \sqrt{\mathbf{u} \bullet \mathbf{u}}$ . First, compute  $\mathbf{u} \bullet \mathbf{u}$ . This is given by

$$\begin{aligned}\mathbf{u} \bullet \mathbf{u} &= (2)(2) + (1)(1) + (4)(4) + (2)(2) \\ &= 4 + 1 + 16 + 4 \\ &= 25\end{aligned}$$

Then,

$$\begin{aligned}\|\mathbf{u}\| &= \sqrt{\mathbf{u} \bullet \mathbf{u}} \\ &= \sqrt{25} \\ &= 5\end{aligned}$$



You may wish to compare this to our previous definition of length, given in Definition F.14.

The **Cauchy Schwarz inequality** is a fundamental inequality satisfied by the dot product. It is given in the following theorem.

### Theorem F.23: Cauchy Schwarz Inequality

*The dot product satisfies the inequality*

$$|\mathbf{u} \bullet \mathbf{v}| \leq \|\mathbf{u}\| \|\mathbf{v}\| \quad (6.1)$$

*Furthermore equality is obtained if and only if one of  $\mathbf{u}$  or  $\mathbf{v}$  is a scalar multiple of the other.*

Notice that this proof was based only on the properties of the dot product listed in Proposition F.21. This means that whenever an operation satisfies these properties, the Cauchy Schwarz inequality holds. There are many other instances of these properties besides vectors in  $\mathbb{R}^n$ .

The Cauchy Schwarz inequality provides another proof of the **triangle inequality** for distances in  $\mathbb{R}^n$ .

### Theorem F.24: Triangle Inequality

*For  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$*

$$\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\| \quad (6.2)$$

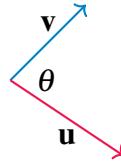
*and equality holds if and only if one of the vectors is a non-negative scalar multiple of the other.*

*Also*

$$|\|\mathbf{u}\| - \|\mathbf{v}\|| \leq \|\mathbf{u} - \mathbf{v}\| \quad (6.3)$$

## F.6.2. The Geometric Significance of the Dot Product

Given two vectors,  $\mathbf{u}$  and  $\mathbf{v}$ , the **included angle** is the angle between these two vectors which is given by  $\theta$  such that  $0 \leq \theta \leq \pi$ . The dot product can be used to determine the included angle between two vectors. Consider the following picture where  $\theta$  gives the included angle.



### Proposition F.25: The Dot Product and the Included Angle

*Let  $\mathbf{u}$  and  $\mathbf{v}$  be two vectors in  $\mathbb{R}^n$ , and let  $\theta$  be the included angle. Then the following equation holds.*

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$$

In words, the dot product of two vectors equals the product of the magnitude (or length) of the two vectors multiplied by the cosine of the included angle. Note this gives a geometric description of the dot product which does not depend explicitly on the coordinates of the vectors.

Consider the following example.

### Example F.26: Find the Angle Between Two Vectors

*Find the angle between the vectors given by*

$$\mathbf{u} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}$$

**Solution.** By Proposition F.25,

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$$

Hence,

$$\cos \theta = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

First, we can compute  $\mathbf{u} \cdot \mathbf{v}$ . By Definition F.19, this equals

$$\mathbf{u} \cdot \mathbf{v} = (2)(3) + (1)(4) + (-1)(1) = 9$$

Then,

$$\begin{aligned} \|\mathbf{u}\| &= \sqrt{(2)(2) + (1)(1) + (-1)(-1)} = \sqrt{6} \\ \|\mathbf{v}\| &= \sqrt{(3)(3) + (4)(4) + (1)(1)} = \sqrt{26} \end{aligned}$$

Therefore, the cosine of the included angle equals

$$\cos \theta = \frac{9}{\sqrt{26}\sqrt{6}} = 0.7205766\dots$$

With the cosine known, the angle can be determined by computing the inverse cosine of that angle, giving approximately  $\theta = 0.76616$  radians. ♠

Another application of the geometric description of the dot product is in finding the angle between two lines. Typically one would assume that the lines intersect. In some situations, however, it may make sense to ask this question when the lines do not intersect, such as the angle between two object trajectories. In any case we understand it to mean the smallest angle between (any of) their direction vectors. The only subtlety here is that if  $\mathbf{u}$  is a direction vector for a line, then so is any multiple  $k\mathbf{u}$ , and thus we will find complementary angles among all angles between direction vectors for two lines, and we simply take the smaller of the two.

### Example F.27: Find the Angle Between Two Lines

*Find the angle between the two lines*

$$L_1 : \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} + t \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$$

and

$$L_2 : \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ -3 \end{bmatrix} + s \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}$$

**Solution.** You can verify that these lines do not intersect, but as discussed above this does not matter and we simply find the smallest angle between any directions vectors for these lines.

To do so we first find the angle between the direction vectors given above:

$$\mathbf{u} = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}$$

In order to find the angle, we solve the following equation for  $\theta$

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$$

to obtain  $\cos \theta = -\frac{1}{2}$  and since we choose included angles between 0 and  $\pi$  we obtain  $\theta = \frac{2\pi}{3}$ .

Now the angles between any two direction vectors for these lines will either be  $\frac{2\pi}{3}$  or its complement  $\phi = \pi - \frac{2\pi}{3} = \frac{\pi}{3}$ . We choose the smaller angle, and therefore conclude that the angle between the two lines is  $\frac{\pi}{3}$ . ♠

We can also use Proposition F.25 to compute the dot product of two vectors.

**Example F.28: Using Geometric Description to Find a Dot Product**

Let  $\mathbf{u}, \mathbf{v}$  be vectors with  $\|\mathbf{u}\| = 3$  and  $\|\mathbf{v}\| = 4$ . Suppose the angle between  $\mathbf{u}$  and  $\mathbf{v}$  is  $\pi/3$ . Find  $\mathbf{u} \bullet \mathbf{v}$ .

**Solution.** From the geometric description of the dot product in Proposition F.25

$$\mathbf{u} \bullet \mathbf{v} = (3)(4) \cos(\pi/3) = 3 \times 4 \times 1/2 = 6$$



Two nonzero vectors are said to be **perpendicular**, sometimes also called **orthogonal**, if the included angle is  $\pi/2$  radians ( $90^\circ$ ).

Consider the following proposition.

**Proposition F.29: Perpendicular Vectors**

Let  $\mathbf{u}$  and  $\mathbf{v}$  be nonzero vectors in  $\mathbb{R}^n$ . Then,  $\mathbf{u}$  and  $\mathbf{v}$  are said to be **perpendicular** exactly when

$$\mathbf{u} \bullet \mathbf{v} = 0$$

Consider the following example.

**Example F.30: Determine if Two Vectors are Perpendicular**

Determine whether the two vectors,

$$\mathbf{u} = \begin{bmatrix} 2 \\ 1 \\ -1 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

are perpendicular.

**Solution.** In order to determine if these two vectors are perpendicular, we compute the dot product. This is given by

$$\mathbf{u} \bullet \mathbf{v} = (2)(1) + (1)(3) + (-1)(5) = 0$$

Therefore, by Proposition F.29 these two vectors are perpendicular.





# G. Spectral Theory

## G.1 Eigenvalues and Eigenvectors of a Matrix

### Outcomes

- A. Describe eigenvalues geometrically and algebraically.
- B. Find eigenvalues and eigenvectors for a square matrix.

Spectral Theory refers to the study of eigenvalues and eigenvectors of a matrix. It is of fundamental importance in many areas and is the subject of our study for this chapter.

### G.1.1. Definition of Eigenvectors and Eigenvalues

In this section, we will work with the entire set of complex numbers, denoted by  $\mathbb{C}$ . Recall that the real numbers,  $\mathbb{R}$  are contained in the complex numbers, so the discussions in this section apply to both real and complex numbers.

To illustrate the idea behind what will be discussed, consider the following example.

#### Example G.1: Eigenvectors and Eigenvalues

Let

$$A = \begin{bmatrix} 0 & 5 & -10 \\ 0 & 22 & 16 \\ 0 & -9 & -2 \end{bmatrix}$$

Compute the product  $AX$  for

$$X = \begin{bmatrix} 5 \\ -4 \\ 3 \end{bmatrix}, X = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

What do you notice about  $AX$  in each of these products?

**Solution.** First, compute  $AX$  for

$$X = \begin{bmatrix} 5 \\ -4 \\ 3 \end{bmatrix}$$

This product is given by

$$AX = \begin{bmatrix} 0 & 5 & -10 \\ 0 & 22 & 16 \\ 0 & -9 & -2 \end{bmatrix} \begin{bmatrix} -5 \\ -4 \\ 3 \end{bmatrix} = \begin{bmatrix} -50 \\ -40 \\ 30 \end{bmatrix} = 10 \begin{bmatrix} -5 \\ -4 \\ 3 \end{bmatrix}$$

In this case, the product  $AX$  resulted in a vector which is equal to 10 times the vector  $X$ . In other words,  $AX = 10X$ .

Let's see what happens in the next product. Compute  $AX$  for the vector

$$X = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

This product is given by

$$AX = \begin{bmatrix} 0 & 5 & -10 \\ 0 & 22 & 16 \\ 0 & -9 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

In this case, the product  $AX$  resulted in a vector equal to 0 times the vector  $X$ ,  $AX = 0X$ .

Perhaps this matrix is such that  $AX$  results in  $kX$ , for every vector  $X$ . However, consider

$$\begin{bmatrix} 0 & 5 & -10 \\ 0 & 22 & 16 \\ 0 & -9 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -5 \\ 38 \\ -11 \end{bmatrix}$$

In this case,  $AX$  did not result in a vector of the form  $kX$  for some scalar  $k$ . ♠

There is something special about the first two products calculated in Example G.1. Notice that for each,  $AX = kX$  where  $k$  is some scalar. When this equation holds for some  $X$  and  $k$ , we call the scalar  $k$  an **eigenvalue** of  $A$ . We often use the special symbol  $\lambda$  instead of  $k$  when referring to eigenvalues. In Example G.1, the values 10 and 0 are eigenvalues for the matrix  $A$  and we can label these as  $\lambda_1 = 10$  and  $\lambda_2 = 0$ .

When  $AX = \lambda X$  for some  $X \neq 0$ , we call such an  $X$  an **eigenvector** of the matrix  $A$ . The eigenvectors of  $A$  are associated to an eigenvalue. Hence, if  $\lambda_1$  is an eigenvalue of  $A$  and  $AX = \lambda_1 X$ , we can label this eigenvector as  $X_1$ . Note again that in order to be an eigenvector,  $X$  must be nonzero.

There is also a geometric significance to eigenvectors. When you have a **nonzero** vector which, when multiplied by a matrix results in another vector which is parallel to the first or equal to  $\mathbf{0}$ , this vector is called an eigenvector of the matrix. This is the meaning when the vectors are in  $\mathbb{R}^n$ .

The formal definition of eigenvalues and eigenvectors is as follows.

**Definition G.2: Eigenvalues and Eigenvectors**

Let  $A$  be an  $n \times n$  matrix and let  $X \in \mathbb{C}^n$  be a **nonzero vector** for which

$$AX = \lambda X \quad (7.1)$$

for some scalar  $\lambda$ . Then  $\lambda$  is called an **eigenvalue** of the matrix  $A$  and  $X$  is called an **eigenvector** of  $A$  associated with  $\lambda$ , or a  $\lambda$ -eigenvector of  $A$ .

The set of all eigenvalues of an  $n \times n$  matrix  $A$  is denoted by  $\sigma(A)$  and is referred to as the **spectrum** of  $A$ .

The eigenvectors of a matrix  $A$  are those vectors  $X$  for which multiplication by  $A$  results in a vector in the same direction or opposite direction to  $X$ . Since the zero vector  $0$  has no direction this would make no sense for the zero vector. As noted above,  $0$  is never allowed to be an eigenvector.

Let's look at eigenvectors in more detail. Suppose  $X$  satisfies 7.1. Then

$$\begin{aligned} AX - \lambda X &= 0 \\ \text{or} \\ (A - \lambda I)X &= 0 \end{aligned}$$

for some  $X \neq 0$ . Equivalently you could write  $(\lambda I - A)X = 0$ , which is more commonly used. Hence, when we are looking for eigenvectors, we are looking for nontrivial solutions to this homogeneous system of equations!

Recall that the solutions to a homogeneous system of equations consist of basic solutions, and the linear combinations of those basic solutions. In this context, we call the basic solutions of the equation  $(\lambda I - A)X = 0$  **basic eigenvectors**. It follows that any (nonzero) linear combination of basic eigenvectors is again an eigenvector.

Suppose the matrix  $(\lambda I - A)$  is invertible, so that  $(\lambda I - A)^{-1}$  exists. Then the following equation would be true.

$$\begin{aligned} X &= IX \\ &= ((\lambda I - A)^{-1}(\lambda I - A))X \\ &= (\lambda I - A)^{-1}((\lambda I - A)X) \\ &= (\lambda I - A)^{-1}0 \\ &= 0 \end{aligned}$$

This claims that  $X = 0$ . However, we have required that  $X \neq 0$ . Therefore  $(\lambda I - A)$  cannot have an inverse!

Recall that if a matrix is not invertible, then its determinant is equal to 0. Therefore we can conclude that

$$\det(\lambda I - A) = 0 \quad (7.2)$$

Note that this is equivalent to  $\det(A - \lambda I) = 0$ .

The expression  $\det(xI - A)$  is a polynomial (in the variable  $x$ ) called the **characteristic polynomial of  $A$** , and  $\det(xI - A) = 0$  is called the **characteristic equation**. For this reason we may also refer to the eigenvalues of  $A$  as **characteristic values**, but the former is often used for historical reasons.

The following theorem claims that the roots of the characteristic polynomial are the eigenvalues of  $A$ . Thus when 7.2 holds,  $A$  has a nonzero eigenvector.

### Theorem G.3: The Existence of an Eigenvector

*Let  $A$  be an  $n \times n$  matrix and suppose  $\det(\lambda I - A) = 0$  for some  $\lambda \in \mathbb{C}$ .*

*Then  $\lambda$  is an eigenvalue of  $A$  and thus there exists a nonzero vector  $X \in \mathbb{C}^n$  such that  $AX = \lambda X$ .*

## G.1.2. Finding Eigenvectors and Eigenvalues

---

Now that eigenvalues and eigenvectors have been defined, we will study how to find them for a matrix  $A$ .

First, consider the following definition.

### Definition G.4: Multiplicity of an Eigenvalue

*Let  $A$  be an  $n \times n$  matrix with characteristic polynomial given by  $\det(xI - A)$ . Then, the multiplicity of an eigenvalue  $\lambda$  of  $A$  is the number of times  $\lambda$  occurs as a root of that characteristic polynomial.*

For example, suppose the characteristic polynomial of  $A$  is given by  $(x - 2)^2$ . Solving for the roots of this polynomial, we set  $(x - 2)^2 = 0$  and solve for  $x$ . We find that  $\lambda = 2$  is a root that occurs twice. Hence, in this case,  $\lambda = 2$  is an eigenvalue of  $A$  of multiplicity equal to 2.

We will now look at how to find the eigenvalues and eigenvectors for a matrix  $A$  in detail. The steps used are summarized in the following procedure.

### Procedure G.5: Finding Eigenvalues and Eigenvectors

*Let  $A$  be an  $n \times n$  matrix.*

- First, find the eigenvalues  $\lambda$  of  $A$  by solving the equation  $\det(xI - A) = 0$ .*
- For each  $\lambda$ , find the basic eigenvectors  $X \neq 0$  by finding the basic solutions to  $(\lambda I - A)X = 0$ .*

*To verify your work, make sure that  $AX = \lambda X$  for each  $\lambda$  and associated eigenvector  $X$ .*

We will explore these steps further in the following example.

**Example G.6: Find the Eigenvalues and Eigenvectors**

Let  $A = \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}$ . Find its eigenvalues and eigenvectors.

**Solution.** We will use Procedure G.5. First we find the eigenvalues of  $A$  by solving the equation

$$\det(xI - A) = 0$$

This gives

$$\begin{aligned} \det\left(x\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}\right) &= 0 \\ \det\begin{bmatrix} x+5 & -2 \\ 7 & x-4 \end{bmatrix} &= 0 \end{aligned}$$

Computing the determinant as usual, the result is

$$x^2 + x - 6 = 0$$

Solving this equation, we find that  $\lambda_1 = 2$  and  $\lambda_2 = -3$ .

Now we need to find the basic eigenvectors for each  $\lambda$ . First we will find the eigenvectors for  $\lambda_1 = 2$ . We wish to find all vectors  $X \neq 0$  such that  $AX = 2X$ . These are the solutions to  $(2I - A)X = 0$ .

$$\begin{aligned} \left(2\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}\right) \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 7 & -2 \\ 7 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

The augmented matrix for this system and corresponding reduced row-echelon form are given by

$$\left[ \begin{array}{cc|c} 7 & -2 & 0 \\ 7 & -2 & 0 \end{array} \right] \rightarrow \dots \rightarrow \left[ \begin{array}{cc|c} 1 & -\frac{2}{7} & 0 \\ 0 & 0 & 0 \end{array} \right]$$

The solution is any vector of the form

$$\begin{bmatrix} \frac{2}{7}s \\ s \end{bmatrix} = s \begin{bmatrix} \frac{2}{7} \\ 1 \end{bmatrix}$$

Multiplying this vector by 7 we obtain a simpler description for the solution to this system, given by

$$t \begin{bmatrix} 2 \\ 7 \end{bmatrix}$$

This gives the basic eigenvector for  $\lambda_1 = 2$  as

$$\begin{bmatrix} 2 \\ 7 \end{bmatrix}$$

To check, we verify that  $AX = 2X$  for this basic eigenvector.

$$\begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 7 \end{bmatrix} = \begin{bmatrix} 4 \\ 14 \end{bmatrix} = 2 \begin{bmatrix} 2 \\ 7 \end{bmatrix}$$

This is what we wanted, so we know this basic eigenvector is correct.

Next we will repeat this process to find the basic eigenvector for  $\lambda_2 = -3$ . We wish to find all vectors  $X \neq 0$  such that  $AX = -3X$ . These are the solutions to  $((-3)I - A)X = 0$ .

$$\begin{aligned} ((-3) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}) \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 2 & -2 \\ 7 & -7 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

The augmented matrix for this system and corresponding reduced row-echelon form are given by

$$\left[ \begin{array}{cc|c} 2 & -2 & 0 \\ 7 & -7 & 0 \end{array} \right] \rightarrow \dots \rightarrow \left[ \begin{array}{cc|c} 1 & -1 & 0 \\ 0 & 0 & 0 \end{array} \right]$$

The solution is any vector of the form

$$\begin{bmatrix} s \\ s \end{bmatrix} = s \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

This gives the basic eigenvector for  $\lambda_2 = -3$  as

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

To check, we verify that  $AX = -3X$  for this basic eigenvector.

$$\begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ -3 \end{bmatrix} = -3 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

This is what we wanted, so we know this basic eigenvector is correct. ♠

The following is an example using Procedure G.5 for a  $3 \times 3$  matrix.

**Example G.7: Find the Eigenvalues and Eigenvectors**

*Find the eigenvalues and eigenvectors for the matrix*

$$A = \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix}$$

**Solution.** We will use Procedure G.5. First we need to find the eigenvalues of  $A$ . Recall that they are the solutions of the equation

$$\det(xI - A) = 0$$

In this case the equation is

$$\det\left(x \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix}\right) = 0$$

which becomes

$$\det \begin{bmatrix} x-5 & 10 & 5 \\ -2 & x-14 & -2 \\ 4 & 8 & x-6 \end{bmatrix} = 0$$

Using Laplace Expansion, compute this determinant and simplify. The result is the following equation.

$$(x-5)(x^2 - 20x + 100) = 0$$

Solving this equation, we find that the eigenvalues are  $\lambda_1 = 5$ ,  $\lambda_2 = 10$  and  $\lambda_3 = 10$ . Notice that 10 is a root of multiplicity two due to

$$x^2 - 20x + 100 = (x-10)^2$$

Therefore,  $\lambda_2 = 10$  is an eigenvalue of multiplicity two.

Now that we have found the eigenvalues for  $A$ , we can compute the eigenvectors.

First we will find the basic eigenvectors for  $\lambda_1 = 5$ . In other words, we want to find all non-zero vectors  $X$  so that  $AX = 5X$ . This requires that we solve the equation  $(5I - A)X = 0$  for  $X$  as follows.

$$\left(5 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix}\right) \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

That is you need to find the solution to

$$\begin{bmatrix} 0 & 10 & 5 \\ -2 & -9 & -2 \\ 4 & 8 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

By now this is a familiar problem. You set up the augmented matrix and row reduce to get the solution. Thus the matrix you must row reduce is

$$\left[ \begin{array}{ccc|c} 0 & 10 & 5 & 0 \\ -2 & -9 & -2 & 0 \\ 4 & 8 & -1 & 0 \end{array} \right]$$

The reduced row-echelon form is

$$\left[ \begin{array}{ccc|c} 1 & 0 & -\frac{5}{4} & 0 \\ 0 & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

and so the solution is any vector of the form

$$\begin{bmatrix} \frac{5}{4}s \\ -\frac{1}{2}s \\ s \end{bmatrix} = s \begin{bmatrix} \frac{5}{4} \\ -\frac{1}{2} \\ 1 \end{bmatrix}$$

where  $s \in \mathbb{R}$ . If we multiply this vector by 4, we obtain a simpler description for the solution to this system, as given by

$$t \begin{bmatrix} 5 \\ -2 \\ 4 \end{bmatrix} \tag{7.3}$$

where  $t \in \mathbb{R}$ . Here, the basic eigenvector is given by

$$X_1 = \begin{bmatrix} 5 \\ -2 \\ 4 \end{bmatrix}$$

Notice that we cannot let  $t = 0$  here, because this would result in the zero vector and eigenvectors are never equal to 0! Other than this value, every other choice of  $t$  in 7.3 results in an eigenvector.

It is a good idea to check your work! To do so, we will take the original matrix and multiply by the basic eigenvector  $X_1$ . We check to see if we get  $5X_1$ .

$$\begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix} \begin{bmatrix} 5 \\ -2 \\ 4 \end{bmatrix} = \begin{bmatrix} 25 \\ -10 \\ 20 \end{bmatrix} = 5 \begin{bmatrix} 5 \\ -2 \\ 4 \end{bmatrix}$$

This is what we wanted, so we know that our calculations were correct.

Next we will find the basic eigenvectors for  $\lambda_2, \lambda_3 = 10$ . These vectors are the basic solutions to the equation,

$$\left( 10 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

That is you must find the solutions to

$$\begin{bmatrix} 5 & 10 & 5 \\ -2 & -4 & -2 \\ 4 & 8 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Consider the augmented matrix

$$\left[ \begin{array}{ccc|c} 5 & 10 & 5 & 0 \\ -2 & -4 & -2 & 0 \\ 4 & 8 & 4 & 0 \end{array} \right]$$

The reduced row-echelon form for this matrix is

$$\left[ \begin{array}{ccc|c} 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

and so the eigenvectors are of the form

$$\begin{bmatrix} -2s-t \\ s \\ t \end{bmatrix} = s \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix} + t \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Note that you can't pick  $t$  and  $s$  both equal to zero because this would result in the zero vector and eigenvectors are never equal to zero.

Here, there are two basic eigenvectors, given by

$$X_2 = \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}, X_3 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Taking any (nonzero) linear combination of  $X_2$  and  $X_3$  will also result in an eigenvector for the eigenvalue  $\lambda = 10$ . As in the case for  $\lambda = 5$ , always check your work! For the first basic eigenvector, we can check  $AX_2 = 10X_2$  as follows.

$$\begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -10 \\ 0 \\ 10 \end{bmatrix} = 10 \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

This is what we wanted. Checking the second basic eigenvector,  $X_3$ , is left as an exercise. ♠

It is important to remember that for any eigenvector  $X$ ,  $X \neq 0$ . However, it is possible to have eigenvalues equal to zero. This is illustrated in the following example.

**Example G.8: A Zero Eigenvalue**

Let

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

Find the eigenvalues and eigenvectors of  $A$ .

**Solution.** First we find the eigenvalues of  $A$ . We will do so using Definition G.2.

In order to find the eigenvalues of  $A$ , we solve the following equation.

$$\det(xI - A) = \det \begin{bmatrix} x-2 & -2 & 2 \\ -1 & x-3 & 1 \\ 1 & -1 & x-1 \end{bmatrix} = 0$$

This reduces to  $x^3 - 6x^2 + 8x = 0$ . You can verify that the solutions are  $\lambda_1 = 0, \lambda_2 = 2, \lambda_3 = 4$ . Notice that while eigenvectors can never equal 0, it is possible to have an eigenvalue equal to 0.

Now we will find the basic eigenvectors. For  $\lambda_1 = 0$ , we need to solve the equation  $(0I - A)X = 0$ . This equation becomes  $-AX = 0$ , and so the augmented matrix for finding the solutions is given by

$$\left[ \begin{array}{ccc|c} -2 & -2 & 2 & 0 \\ -1 & -3 & 1 & 0 \\ 1 & -1 & -1 & 0 \end{array} \right]$$

The reduced row-echelon form is

$$\left[ \begin{array}{ccc|c} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

Therefore, the eigenvectors are of the form  $t \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$  where  $t \neq 0$  and the basic eigenvector is given

by

$$X_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

We can verify that this eigenvector is correct by checking that the equation  $AX_1 = 0X_1$  holds. The product  $AX_1$  is given by

$$AX_1 = \begin{bmatrix} 2 & 2 & -2 \\ 1 & 3 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This clearly equals  $0X_1$ , so the equation holds. Hence,  $AX_1 = 0X_1$  and so 0 is an eigenvalue of  $A$ .

Computing the other basic eigenvectors is left as an exercise. ♠

In the following sections, we examine ways to simplify this process of finding eigenvalues and eigenvectors by using properties of special types of matrices.

### G.1.3. Eigenvalues and Eigenvectors for Special Types of Matrices

A special type of matrix we will consider in this section is the triangular matrix. Recall Definition E.12 which states that an upper (lower) triangular matrix contains all zeros below (above) the main diagonal. Remember that finding the determinant of a triangular matrix is a simple procedure of taking the product of the entries on the main diagonal.. It turns out that there is also a simple way to find the eigenvalues of a triangular matrix.

In the next example we will demonstrate that the eigenvalues of a triangular matrix are the entries on the main diagonal.

#### Example G.9: Eigenvalues for a Triangular Matrix

Let  $A = \begin{bmatrix} 1 & 2 & 4 \\ 0 & 4 & 7 \\ 0 & 0 & 6 \end{bmatrix}$ . Find the eigenvalues of  $A$ .

**Solution.** We need to solve the equation  $\det(xI - A) = 0$  as follows

$$\det(xI - A) = \det \begin{bmatrix} x-1 & -2 & -4 \\ 0 & x-4 & -7 \\ 0 & 0 & x-6 \end{bmatrix} = (x-1)(x-4)(x-6) = 0$$

Solving the equation  $(x-1)(x-4)(x-6) = 0$  for  $x$  results in the eigenvalues  $\lambda_1 = 1, \lambda_2 = 4$  and  $\lambda_3 = 6$ . Thus the eigenvalues are the entries on the main diagonal of the original matrix. ♠

The same result is true for lower triangular matrices. For any triangular matrix, the eigenvalues are equal to the entries on the main diagonal. To find the eigenvectors of a triangular matrix, we use the usual procedure.

## G.2 Positive Semi-Definite Matrices

Wikipedia - Definite Symmetric Matrix

In linear algebra, a symmetric  $n \times n$  real matrix  $M$  is said to be **positive-definite** if the scalar  $z^T M z$  is strictly positive for every non-zero column vector  $z$  of  $n$  real numbers. Here  $z^T$  denotes the transpose of  $z$ .<sup>1</sup> When interpreting  $Mz$  as the output of an operator,  $M$ , that is acting on an input,  $z$ , the property of positive definiteness implies that the output always has a positive inner product with the input, as often observed in physical processes.

More generally, a complex  $n \times n$  Hermitian matrix  $M$  is said to be **positive-definite** if the scalar  $z^* M z$  is strictly positive for every non-zero column vector  $z$  of  $n$  complex numbers. Here  $z^*$  denotes the conjugate transpose of  $z$ . Note that  $z^* M z$  is automatically real since  $M$  is Hermitian.

<sup>1</sup>W

**Positive semi-definite** matrices are defined similarly, except that the above scalars  $z^\top Mz$  or  $z^* M z$  must be positive *or zero* (i.e. non-negative). **Negative-definite** and **negative semi-definite** matrices are defined analogously. A matrix that is not positive semi-definite and not negative semi-definite is called **indefinite**.

The matrix  $M$  is positive-definite if and only if the bilinear form  $\langle z, w \rangle = z^\top Mw$  is positive-definite (and similarly for a positive-definite sesquilinear form in the complex case). This is a coordinate realization of an inner product on a vector space.<sup>2</sup> Some authors use more general definitions of definiteness, including some non-symmetric real matrices, or non-Hermitian complex ones.

### G.2.1. Definitions

In the following definitions,  $\mathbf{x}^\top$  is the transpose of  $\mathbf{x}$ ,  $\mathbf{x}^*$  is the conjugate transpose of  $\mathbf{x}$  and  $\mathbf{0}$  denotes the  $n$ -dimensional zero-vector.

#### Definition G.10: Definiteness for Real Matrices

An  $n \times n$  symmetric real matrix  $M$  is said to be

- **positive-definite** if  $\mathbf{x}^\top M \mathbf{x} > 0$  for all non-zero  $\mathbf{x}$  in  $\mathbb{R}^n$ ,
- **positive semidefinite** or **non-negative-definite** if  $\mathbf{x}^\top M \mathbf{x} \geq 0$  for all  $\mathbf{x}$  in  $\mathbb{R}^n$ ,
- **negative-definite** if  $\mathbf{x}^\top M \mathbf{x} < 0$  for all non-zero  $\mathbf{x}$  in  $\mathbb{R}^n$ ,
- **negative-semidefinite** or **non-positive-definite** if  $\mathbf{x}^\top M \mathbf{x} \leq 0$  for all  $\mathbf{x}$  in  $\mathbb{R}^n$ ,
- An  $n \times n$  symmetric real matrix which is neither positive semidefinite nor negative semidefinite is called **indefinite**.

#### Example G.11: Identity Matrix

The identity matrix  $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  is positive-definite (and as such also positive semi-definite). It is a real symmetric matrix, and, for any non-zero column vector  $\mathbf{z}$  with real entries  $a$  and  $b$ , one has

$$\mathbf{z}^\top \mathbf{z} = [a \ b] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a^2 + b^2.$$

Seen as a complex matrix, for any non-zero column vector  $\mathbf{z}$  with complex entries  $a$  and  $b$  one has

$$\mathbf{z}^* \mathbf{z} = [\bar{a} \ \bar{b}] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \bar{a}a + \bar{b}b = |a|^2 + |b|^2$$

Either way, the result is positive since  $\mathbf{z}$  is not the zero vector (that is, at least one of  $a$  and  $b$  is not zero).

**Example G.12: Positive definite**

The real symmetric matrix

$$M = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

is positive-definite since for any non-zero column vector  $\mathbf{z}$  with entries  $a$ ,  $b$  and  $c$ , we have

$$\mathbf{z}^\top M \mathbf{z} = (\mathbf{z}^\top M) \mathbf{z} \quad (7.1)$$

$$= [(2a - b) \quad (-a + 2b - c) \quad (-b + 2c)] \quad (7.2)$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad (7.3)$$

$$= (2a - b)a + (-a + 2b - c)b + (-b + 2c)c \quad (7.4)$$

$$= 2a^2 - ba - ab + 2b^2 - cb - bc + 2c^2 \quad (7.5)$$

$$= 2a^2 - 2ab + 2b^2 - 2bc + 2c^2 \quad (7.6)$$

$$= a^2 + a^2 - 2ab + b^2 + b^2 - 2bc + c^2 + c^2 \quad (7.7)$$

$$= a^2 + (a - b)^2 + (b - c)^2 + c^2 \quad (7.8)$$

This result is a sum of squares, and therefore non-negative; and is zero only if  $a = b = c = 0$ , that is, when  $\mathbf{z}$  is the zero vector.

**Example G.13:  $A^\top A$** 

For any real invertible matrix  $A$ , the product  $A^\top A$  is a positive definite matrix. A simple proof is that for any non-zero vector  $\mathbf{z}$ , the condition  $\mathbf{z}^\top A^\top A \mathbf{z} = (\mathbf{A}\mathbf{z})^\top (\mathbf{A}\mathbf{z}) = \|\mathbf{A}\mathbf{z}\|^2 > 0$ , since the invertibility of matrix  $A$  means that  $\mathbf{A}\mathbf{z} \neq 0$ .

The example  $M$  above shows that a matrix in which some elements are negative may still be positive definite. Conversely, a matrix whose entries are all positive is not necessarily positive definite, as for example

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

for which  $[-1 \ 1] N [-1 \ 1]^\top = -2 < 0$ .

## G.2.2. Eigenvalues

### Theorem G.14: Eigenvalue Characterizations

Let  $M$  be an  $n \times n$  Hermitian matrix.

- $M$  is positive definite if and only if all of its eigenvalues are positive.
- $M$  is positive semi-definite if and only if all of its eigenvalues are non-negative.
- $M$  is negative definite if and only if all of its eigenvalues are negative
- $M$  is negative semi-definite if and only if all of its eigenvalues are non-positive.
- $M$  is indefinite if and only if it has both positive and negative eigenvalues.

Let  $P^{-1}DP$  be an eigendecomposition of  $M$ , where  $P$  is a unitary complex matrix whose rows comprise an orthonormal basis of eigenvectors of  $M$ , and  $D$  is a *real* diagonal matrix whose main diagonal contains the corresponding eigenvalues. The matrix  $M$  may be regarded as a diagonal matrix  $D$  that has been re-expressed in coordinates of the basis  $P$ . In particular, the one-to-one change of variable  $y = Pz$  shows that  $z^*Mz$  is real and positive for any complex vector  $z$  if and only if  $y^*Dy$  is real and positive for any  $y$ ; in other words, if  $D$  is positive definite. For a diagonal matrix, this is true only if each element of the main diagonal—that is, every eigenvalue of  $M$ —is positive. Since the spectral theorem guarantees all eigenvalues of a Hermitian matrix to be real, the positivity of eigenvalues can be checked using Descartes' rule of alternating signs when the characteristic polynomial of a real, symmetric matrix  $M$  is available.

## G.2.3. Quadratic forms

The (purely) quadratic form associated with a real  $n \times n$  matrix  $M$  is the function  $Q : \mathbb{R}^n \rightarrow \mathbb{R}$  such that  $Q(x) = x^\top Mx$  for all  $x$ .  $M$  can be assumed symmetric by replacing it with  $\frac{1}{2}(M + M^\top)$ .

A symmetric matrix  $M$  is positive definite if and only if its quadratic form is a strictly convex function.

More generally, any quadratic function from  $\mathbb{R}^n$  to  $\mathbb{R}$  can be written as  $x^\top Mx + x^\top b + c$  where  $M$  is a symmetric  $n \times n$  matrix,  $b$  is a real  $n$ -vector, and  $c$  a real constant. This quadratic function is strictly convex, and hence has a unique finite global minimum, if and only if  $M$  is positive definite. For this reason, positive definite matrices play an important role in optimization problems.

## G.2.4. Properties

### G.2.4.1. Inverse of positive definite matrix

Every positive definite matrix is invertible and its inverse is also positive definite.<sup>3</sup> If  $M \geq N > 0$  then  $N^{-1} \geq M^{-1} > 0$ .<sup>4</sup> Moreover, by the min-max theorem, the  $k$ th largest eigenvalue of  $M$  is greater

<sup>3</sup>, p. 397

<sup>4</sup>, Corollary 7.7.4(a)

than the  $k$ th largest eigenvalue of  $N$ .

#### G.2.4.2. Scaling

---

If  $M$  is positive definite and  $r > 0$  is a real number, then  $rM$  is positive definite.<sup>5</sup>

#### G.2.4.3. Addition

---

If  $M$  and  $N$  are positive definite, then the sum  $M + N$  is also positive definite.<sup>6</sup>

#### G.2.4.4. Multiplication

---

- If  $M$  and  $N$  are positive definite, then the products  $MNM$  and  $NMN$  are also positive definite. If  $MN = NM$ , then  $MN$  is also positive definite.
- If  $M$  is positive semidefinite, then  $Q^\top MQ$  is positive semidefinite. If  $M$  is positive definite and  $Q$  has full column rank, then  $Q^\top MQ$  is positive definite.

#### G.2.4.5. Cholesky decomposition

---

For any matrix  $A$ , the matrix  $A^*A$  is positive semidefinite, and  $\text{rank}(A) = \text{rank}(A^*A)$ . Conversely, any Hermitian positive semi-definite matrix  $M$  can be written as  $M = LL^*$ , where  $L$  is lower triangular; this is the Cholesky decomposition. If  $M$  is not positive definite, then some of the diagonal elements of  $L$  may be zero.

A hermitian matrix  $M$  is positive definite if and only if it has a unique Cholesky decomposition, i.e. the matrix  $M$  is positive definite if and only if there exists a unique lower triangular matrix  $L$ , with real and strictly positive diagonal elements, such that  $M = LL^*$ .

#### G.2.4.6. Square root

---

A matrix  $M$  is positive semi-definite if and only if there is a positive semi-definite matrix  $B$  with  $B^2 = M$ . This matrix  $B$  is unique,<sup>7</sup> is called the square root of  $M$ , and is denoted with  $B = M^{\frac{1}{2}}$  (the square root  $B$  is not to be confused with the matrix  $L$  in the Cholesky factorization  $M = LL^*$ , which is also sometimes called the square root of  $M$ ).

If  $M > N > 0$  then  $M^{\frac{1}{2}} > N^{\frac{1}{2}} > 0$ .

---

<sup>5</sup>, Observation 7.1.3

<sup>6</sup>

<sup>7</sup>, Theorem 7.2.6 with  $k = 2$

### G.2.4.7. Submatrices

---

Every principal submatrix of a positive definite matrix is positive definite.

## G.2.5. Convexity

---

The set of positive semidefinite symmetric matrices is convex. That is, if  $M$  and  $N$  are positive semidefinite, then for any  $\alpha$  between 0 and 1,  $\alpha M + (1 - \alpha)N$  is also positive semidefinite. For any vector  $x$ :

$$x^\top (\alpha M + (1 - \alpha)N)x = \alpha x^\top Mx + (1 - \alpha)x^\top Nx \geq 0.$$

This property guarantees that semidefinite programming problems converge to a globally optimal solution.

### G.2.5.1. Further properties

---

A Hermitian matrix is positive semidefinite if and only if all of its principal minors are nonnegative. It is however not enough to consider the leading principal minors only, as is checked on the diagonal matrix with entries 0 and 1.

### G.2.5.2. Block matrices

---

A positive  $2n \times 2n$  matrix may also be defined by blocks:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

where each block is  $n \times n$ . By applying the positivity condition, it immediately follows that  $A$  and  $D$  are hermitian, and  $C = B^*$ .

We have that  $z^* M z \geq 0$  for all complex  $z$ , and in particular for  $z = [v, 0]^\top$ . Then

$$\begin{bmatrix} v^* & 0 \end{bmatrix} \begin{bmatrix} A & B \\ B^* & D \end{bmatrix} \begin{bmatrix} v \\ 0 \end{bmatrix} = v^* A v \geq 0.$$

A similar argument can be applied to  $D$ , and thus we conclude that both  $A$  and  $D$  must be positive definite matrices, as well.

Converse results can be proved with stronger conditions on the blocks, for instance using the Schur complement.

### G.2.5.3. Local extrema

---

A general quadratic form  $f(\mathbf{x})$  on  $n$  real variables  $x_1, \dots, x_n$  can always be written as  $\mathbf{x}^\top M \mathbf{x}$  where  $\mathbf{x}$  is the column vector with those variables, and  $M$  is a symmetric real matrix. Therefore, the matrix being positive definite means that  $f$  has a unique minimum (zero) when  $\mathbf{x}$  is zero, and is strictly positive for any other  $\mathbf{x}$ .

More generally, a twice-differentiable real function  $f$  on  $n$  real variables has local minimum at arguments  $x_1, \dots, x_n$  if its gradient is zero and its Hessian (the matrix of all second derivatives) is positive semi-definite at that point. Similar statements can be made for negative definite and semi-definite matrices.

### G.2.5.4. Covariance

---

In statistics, the covariance matrix of a multivariate probability distribution is always positive semi-definite; and it is positive definite unless one variable is an exact linear function of the others. Conversely, every positive semi-definite matrix is the covariance matrix of some multivariate distribution.

## G.2.6. External links

---

- Wolfram MathWorld: Positive Definite Matrix



## **Part VII**

# **Other Appendices**



# A. Some Prerequisite Topics

---

The topics presented in this section are important concepts in mathematics and therefore should be examined.

## A.1 Sets and Set Notation

---

A set is a collection of things called elements. For example  $\{1, 2, 3, 8\}$  would be a set consisting of the elements 1, 2, 3, and 8. To indicate that 3 is an element of  $\{1, 2, 3, 8\}$ , it is customary to write  $3 \in \{1, 2, 3, 8\}$ . We can also indicate when an element is not in a set, by writing  $9 \notin \{1, 2, 3, 8\}$  which says that 9 is not an element of  $\{1, 2, 3, 8\}$ . Sometimes a rule specifies a set. For example you could specify a set as all integers larger than 2. This would be written as  $S = \{x \in \mathbb{Z} : x > 2\}$ . This notation says:  $S$  is the set of all integers,  $x$ , such that  $x > 2$ .

Suppose  $A$  and  $B$  are sets with the property that every element of  $A$  is an element of  $B$ . Then we say that  $A$  is a subset of  $B$ . For example,  $\{1, 2, 3, 8\}$  is a subset of  $\{1, 2, 3, 4, 5, 8\}$ . In symbols, we write  $\{1, 2, 3, 8\} \subseteq \{1, 2, 3, 4, 5, 8\}$ . It is sometimes said that “ $A$  is contained in  $B$ ” or even “ $B$  contains  $A$ ”. The same statement about the two sets may also be written as  $\{1, 2, 3, 4, 5, 8\} \supseteq \{1, 2, 3, 8\}$ .

We can also talk about the *union* of two sets, which we write as  $A \cup B$ . This is the set consisting of everything which is an element of at least one of the sets,  $A$  or  $B$ . As an example of the union of two sets, consider  $\{1, 2, 3, 8\} \cup \{3, 4, 7, 8\} = \{1, 2, 3, 4, 7, 8\}$ . This set is made up of the numbers which are in at least one of the two sets.

In general

$$A \cup B = \{x : x \in A \text{ or } x \in B\}$$

Notice that an element which is in *both*  $A$  and  $B$  is also in the union, as well as elements which are in only one of  $A$  or  $B$ .

Another important set is the intersection of two sets  $A$  and  $B$ , written  $A \cap B$ . This set consists of everything which is in *both* of the sets. Thus  $\{1, 2, 3, 8\} \cap \{3, 4, 7, 8\} = \{3, 8\}$  because 3 and 8 are those elements the two sets have in common. In general,

$$A \cap B = \{x : x \in A \text{ and } x \in B\}$$

If  $A$  and  $B$  are two sets,  $A \setminus B$  denotes the set of things which are in  $A$  but not in  $B$ . Thus

$$A \setminus B = \{x \in A : x \notin B\}$$

For example, if  $A = \{1, 2, 3, 8\}$  and  $B = \{3, 4, 7, 8\}$ , then  $A \setminus B = \{1, 2, 3, 8\} \setminus \{3, 4, 7, 8\} = \{1, 2\}$ .

A special set which is very important in mathematics is the empty set denoted by  $\emptyset$ . The empty set,  $\emptyset$ , is defined as the set which has no elements in it. It follows that the empty set is a subset of every

set. This is true because if it were not so, there would have to exist a set  $A$ , such that  $\emptyset$  has something in it which is not in  $A$ . However,  $\emptyset$  has nothing in it and so it must be that  $\emptyset \subseteq A$ .

We can also use brackets to denote sets which are intervals of numbers. Let  $a$  and  $b$  be real numbers. Then

- $[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\}$
- $[a, b) = \{x \in \mathbb{R} : a \leq x < b\}$
- $(a, b) = \{x \in \mathbb{R} : a < x < b\}$
- $(a, b] = \{x \in \mathbb{R} : a < x \leq b\}$
- $[a, \infty) = \{x \in \mathbb{R} : x \geq a\}$
- $(-\infty, a] = \{x \in \mathbb{R} : x \leq a\}$

These sorts of sets of real numbers are called intervals. The two points  $a$  and  $b$  are called endpoints, or bounds, of the interval. In particular,  $a$  is the *lower bound* while  $b$  is the *upper bound* of the above intervals, where applicable. Other intervals such as  $(-\infty, b)$  are defined by analogy to what was just explained. In general, the curved parenthesis,  $($ , indicates the end point is not included in the interval, while the square parenthesis,  $[$ , indicates this end point is included. The reason that there will always be a curved parenthesis next to  $\infty$  or  $-\infty$  is that these are not real numbers and cannot be included in the interval in the way a real number can.

To illustrate the use of this notation relative to intervals consider three examples of inequalities. Their solutions will be written in the interval notation just described.

### Example A.1: Solving an Inequality

*Solve the inequality  $2x + 4 \leq x - 8$ .*

**Solution.** We need to find  $x$  such that  $2x + 4 \leq x - 8$ . Solving for  $x$ , we see that  $x \leq -12$  is the answer. This is written in terms of an interval as  $(-\infty, -12]$ . ♠

Consider the following example.

### Example A.2: Solving an Inequality

*Solve the inequality  $(x + 1)(2x - 3) \geq 0$ .*

**Solution.** We need to find  $x$  such that  $(x + 1)(2x - 3) \geq 0$ . The solution is given by  $x \leq -1$  or  $x \geq \frac{3}{2}$ . Therefore,  $x$  which fit into either of these intervals gives a solution. In terms of set notation this is denoted by  $(-\infty, -1] \cup [\frac{3}{2}, \infty)$ . ♠

Consider one last example.

### Example A.3: Solving an Inequality

*Solve the inequality  $x(x + 2) \geq -4$ .*

**Solution.** This inequality is true for any value of  $x$  where  $x$  is a real number. We can write the solution as  $\mathbb{R}$  or  $(-\infty, \infty)$ . ♠

In the next section, we examine another important mathematical concept.

## A.2 Well Ordering and Induction

---

We begin this section with some important notation. Summation notation, written  $\sum_{i=1}^j i$ , represents a sum. Here,  $i$  is called the index of the sum, and we add iterations until  $i = j$ . For example,

$$\sum_{i=1}^j i = 1 + 2 + \cdots + j$$

Another example:

$$a_{11} + a_{12} + a_{13} = \sum_{i=1}^3 a_{1i}$$

The following notation is a specific use of summation notation.

### Notation A.4: Summation Notation

Let  $a_{ij}$  be real numbers, and suppose  $1 \leq i \leq r$  while  $1 \leq j \leq s$ . These numbers can be listed in a rectangular array as given by

$$\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & & \vdots \\ a_{r1} & a_{r2} & \cdots & a_{rs} \end{array}$$

Then  $\sum_{j=1}^s \sum_{i=1}^r a_{ij}$  means to first sum the numbers in each column (using  $i$  as the index) and then to add the sums which result (using  $j$  as the index). Similarly,  $\sum_{i=1}^r \sum_{j=1}^s a_{ij}$  means to sum the vectors in each row (using  $j$  as the index) and then to add the sums which result (using  $i$  as the index).

Notice that since addition is commutative,  $\sum_{j=1}^s \sum_{i=1}^r a_{ij} = \sum_{i=1}^r \sum_{j=1}^s a_{ij}$ .

We now consider the main concept of this section. Mathematical induction and well ordering are two extremely important principles in math. They are often used to prove significant things which would be hard to prove otherwise.

### Definition A.5: Well Ordered

A set is well ordered if every nonempty subset  $S$ , contains a smallest element  $z$  having the property that  $z \leq x$  for all  $x \in S$ .

In particular, the set of natural numbers defined as

$$\mathbb{N} = \{1, 2, \dots\}$$

is well ordered.

Consider the following proposition.

### Proposition A.6: Well Ordered Sets

*Any set of integers larger than a given number is well ordered.*

This proposition claims that if a set has a lower bound which is a real number, then this set is well ordered.

Further, this proposition implies the principle of mathematical induction. The symbol  $\mathbb{Z}$  denotes the set of all integers. Note that if  $a$  is an integer, then there are no integers between  $a$  and  $a + 1$ .

### Theorem A.7: Mathematical Induction

*A set  $S \subseteq \mathbb{Z}$ , having the property that  $a \in S$  and  $n + 1 \in S$  whenever  $n \in S$ , contains all integers  $x \in \mathbb{Z}$  such that  $x \geq a$ .*

**Proof.** Let  $T$  consist of all integers larger than or equal to  $a$  which are not in  $S$ . The theorem will be proved if  $T = \emptyset$ . If  $T \neq \emptyset$  then by the well ordering principle, there would have to exist a smallest element of  $T$ , denoted as  $b$ . It must be the case that  $b > a$  since by definition,  $a \notin T$ . Thus  $b \geq a + 1$ , and so  $b - 1 \geq a$  and  $b - 1 \notin S$  because if  $b - 1 \in S$ , then  $b - 1 + 1 = b \in S$  by the assumed property of  $S$ . Therefore,  $b - 1 \in T$  which contradicts the choice of  $b$  as the smallest element of  $T$ . ( $b - 1$  is smaller.) Since a contradiction is obtained by assuming  $T \neq \emptyset$ , it must be the case that  $T = \emptyset$  and this says that every integer at least as large as  $a$  is also in  $S$ . ♠

Mathematical induction is a very useful device for proving theorems about the integers. The procedure is as follows.

### Procedure A.8: Proof by Mathematical Induction

*Suppose  $S_n$  is a statement which is a function of the number  $n$ , for  $n = 1, 2, \dots$ , and we wish to show that  $S_n$  is true for all  $n \geq 1$ . To do so using mathematical induction, use the following steps.*

- (a) **Base Case:** Show  $S_1$  is true.
- (b) Assume  $S_n$  is true for some  $n$ , which is the **induction hypothesis**. Then, using this assumption, show that  $S_{n+1}$  is true.

*Proving these two steps shows that  $S_n$  is true for all  $n = 1, 2, \dots$*

We can use this procedure to solve the following examples.

**Example A.9: Proving by Induction**

*Prove by induction that  $\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$ .*

**Solution.** By Procedure A.8, we first need to show that this statement is true for  $n = 1$ . When  $n = 1$ , the statement says that

$$\begin{aligned}\sum_{k=1}^1 k^2 &= \frac{1(1+1)(2(1)+1)}{6} \\ &= \frac{6}{6} \\ &= 1\end{aligned}$$

The sum on the left hand side also equals 1, so this equation is true for  $n = 1$ .

Now suppose this formula is valid for some  $n \geq 1$  where  $n$  is an integer. Hence, the following equation is true.

$$\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6} \quad (1.1)$$

We want to show that this is true for  $n + 1$ .

Suppose we add  $(n+1)^2$  to both sides of equation 1.1.

$$\begin{aligned}\sum_{k=1}^{n+1} k^2 &= \sum_{k=1}^n k^2 + (n+1)^2 \\ &= \frac{n(n+1)(2n+1)}{6} + (n+1)^2\end{aligned}$$

The step going from the first to the second line is based on the assumption that the formula is true for  $n$ . Now simplify the expression in the second line,

$$\frac{n(n+1)(2n+1)}{6} + (n+1)^2$$

This equals

$$(n+1) \left( \frac{n(2n+1)}{6} + (n+1) \right)$$

and

$$\frac{n(2n+1)}{6} + (n+1) = \frac{6(n+1) + 2n^2 + n}{6} = \frac{(n+2)(2n+3)}{6}$$

Therefore,

$$\sum_{k=1}^{n+1} k^2 = \frac{(n+1)(n+2)(2n+3)}{6} = \frac{(n+1)((n+1)+1)(2(n+1)+1)}{6}$$

showing the formula holds for  $n + 1$  whenever it holds for  $n$ . This proves the formula by mathematical induction. In other words, this formula is true for all  $n = 1, 2, \dots$  ♠

Consider another example.

**Example A.10: Proving an Inequality by Induction**

Show that for all  $n \in \mathbb{N}$ ,  $\frac{1}{2} \cdot \frac{3}{4} \cdots \frac{2n-1}{2n} < \frac{1}{\sqrt{2n+1}}$ .

**Solution.** Again we will use the procedure given in Procedure A.8 to prove that this statement is true for all  $n$ . Suppose  $n = 1$ . Then the statement says

$$\frac{1}{2} < \frac{1}{\sqrt{3}}$$

which is true.

Suppose then that the inequality holds for  $n$ . In other words,

$$\frac{1}{2} \cdot \frac{3}{4} \cdots \frac{2n-1}{2n} < \frac{1}{\sqrt{2n+1}}$$

is true.

Now multiply both sides of this inequality by  $\frac{2n+1}{2n+2}$ . This yields

$$\frac{1}{2} \cdot \frac{3}{4} \cdots \frac{2n-1}{2n} \cdot \frac{2n+1}{2n+2} < \frac{1}{\sqrt{2n+1}} \frac{2n+1}{2n+2} = \frac{\sqrt{2n+1}}{2n+2}$$

The theorem will be proved if this last expression is less than  $\frac{1}{\sqrt{2n+3}}$ . This happens if and only if

$$\left( \frac{1}{\sqrt{2n+3}} \right)^2 = \frac{1}{2n+3} > \frac{2n+1}{(2n+2)^2}$$

which occurs if and only if  $(2n+2)^2 > (2n+3)(2n+1)$  and this is clearly true which may be seen from expanding both sides. This proves the inequality. ♠

Let's review the process just used. If  $S$  is the set of integers at least as large as 1 for which the formula holds, the first step was to show  $1 \in S$  and then that whenever  $n \in S$ , it follows  $n+1 \in S$ . Therefore, by the principle of mathematical induction,  $S$  contains  $[1, \infty) \cap \mathbb{Z}$ , all positive integers. In doing an inductive proof of this sort, the set  $S$  is normally not mentioned. One just verifies the steps above.

## A.3 GECODE and MiniZinc

---

Open constraint programming toolkit. <https://www.gecode.org/> See also MiniZinc, which is a modeler that uses GECODE. <https://www.minizinc.org/>.

Also, they have two coursera courses on using their code: <https://www.coursera.org/learn/basic-modeling?action=enroll&authMode=signup>  
<https://www.coursera.org/learn/advanced-modeling?action=enroll&authMode=signup>

## A.4 Optaplanner

---

Open source software to solve a variety of problems with local heuristics. All code is in Java.

<https://www.optaplanner.org/>

## A.5 Python Modeling/Optimization

---

### A.5.1. SCIP

---

SCIP youtube channel

Youtube! SCIP solving MINLP Circle Packing Problem Model and Code in SCIP SCIP - Python Interface Demonstration

### A.5.2. Pyomo

---

Excellent modeling language. Open source. Many features. <http://www.pyomo.org/>

### A.5.3. Python-MIP

---

Awesome new modeling language for python that is very efficient at setting up optimization problems. Loads CBC binaries. Can be installed with pip. <https://python-mip.com/>

### A.5.4. Local Solver

---

<https://www.localsolver.com/> <https://www.youtube.com/watch?v=4aw9PM09U5Q>

### A.5.5. GUROBI

---

Solver takes too long to find Incumbent solution:

It might be better to focus on trying to find heuristic solutions faster. You can do t

### A.5.6. CPLEX

---

ILOG CPLEX optimization Studio

<https://www.youtube.com/watch?v=IwYt5bzrhxA>

### A.5.7. Scipy

---

[http://scipy-lectures.org/advanced/mathematical\\_optimization/index.html](http://scipy-lectures.org/advanced/mathematical_optimization/index.html)

## A.6 Julia

---

### A.6.1. JuMP

---

<https://www.juliaopt.org/> <https://jump.dev/JuMP.jl/dev/>

## A.7 LINDO/LINGO

---

<https://www.lindo.com/>

## A.8 Foundations of Machine Learning

---

Free course with excellent videos on foundations of machine learning

## A.9 Convex Optimization

---

<https://www.youtube.com/watch?v=thuYiebq1cE&t=925s>    <https://www.youtube.com/watch?v=40ifjG2kJQ>

## B. Dynamic Programming

---

Repository of Dynamic Programming Examples

**Lab Objective:** *Sequential decision making problems are a class of problems in which the current choice depends on future choices. They are a subset of Markov decision processes, an important class of problems with applications in business, robotics, and economics. Dynamic programming is a method of solving these problems that optimizes the solution by breaking the problem down into steps and optimizing the decision at each time period. In this lab we use dynamic programming to solve two classic dynamic optimization problems.*

### The Marriage Problem

---

Many dynamic optimization problems can be classified as *optimal stopping* problems, where the goal is to determine at what time to take an action to maximize the expected reward. For example, when hiring a secretary, how many people should you interview before hiring the current interviewer? Or how many people should you date before you get married? These problems try to determine at what person  $t$  to stop in order to maximize the chance of getting the best candidate.

For instance, let  $N$  be the number of people you could date. After dating each person, you can either marry them or move on; you can't resume a relationship once it ends. In addition, you can rank your current relationship to all of the previous options, but not to future ones. The goal is to find the policy that maximizes the probability of choosing the best marriage partner. That policy may not always choose the best candidate, but it should get an almost-best candidate most of the time.

Let  $V(t - 1)$  be the probability that we choose the best partner when we have passed over the first  $t - 1$  candidates with an optimal policy. In other words, we have dated  $t - 1$  people and want to know the probability that the  $t^{th}$  person is the one we should marry. Note that the probability that the  $t^{th}$  person is not the best candidate is  $\frac{t-1}{t}$  and the probability that they are is  $\frac{1}{t}$ . If the  $t^{th}$  person is not the best out of the first  $t$ , then probability they are the best overall is 0 and the probability they are not is  $V(t)$ . If the  $t^{th}$  person is the best out of the first  $t$ , then the probability they are the best overall is  $\frac{t}{N}$  and the probability they are not is  $V(t)$ .

By Bellman's optimality equations,

$$V(t - 1) = \frac{t - 1}{t} \max \{0, V(t)\} + \frac{1}{t} \max \left\{ \frac{t}{N}, V(t) \right\} = \max \left\{ \frac{t - 1}{t} V(t) + \frac{1}{N}, V(t) \right\}. \quad (2.1)$$

Notice that (2.1) implies that  $V(t - 1) \geq V(t)$  for all  $t \leq N$ . Hence, the probability of selecting the best match  $V(t)$  is non-increasing. Conversely,  $P(t \text{ is best overall} | t \text{ is best out of the first } t) = \frac{t}{N}$  is strictly increasing. Therefore, there is some  $t_0$ , called the *optimal stopping point*, such that  $V(t) \leq \frac{t}{N}$

for all  $t \geq t_0$ . After  $t_0$  relationships, we choose the next partner who is better than all of the previous ones. We can write (2.1) as

$$V(t-1) = \begin{cases} V(t_0) & t < t_0, \\ \frac{t-1}{t}V(t) + \frac{1}{N} & t \geq t_0. \end{cases}$$

The goal of an optimal stopping problem is to find  $t_0$ , which we can do by backwards induction. We start at the final candidate, who always has probability 0 of being the best overall if they are not the best so far, and work our way backwards, computing the expected value  $V(t)$ , for  $t = N, N-1, \dots, 1$ .

If  $N = 4$ , we have

$$\begin{aligned} V(4) &= 0, \\ V(3) &= \max \left\{ \frac{3}{4}V(4) + \frac{1}{4}, 0 \right\} = .25, \\ V(2) &= \max \left\{ \frac{2}{3}V(3) + \frac{1}{4}, .25 \right\} = .4166, \\ V(1) &= \max \left\{ \frac{1}{4}, .4166 \right\} = .4166. \end{aligned}$$

In this case, the maximum expected value is .4166 and the stopping point is  $t = 2$ . It is also useful to look at the optimal stopping percentage of people to date before getting married. In this case, it is  $2/4 = .5$ .

### Problem B.1: W

Write a function that accepts a number of candidates  $N$ . Calculate the expected values of choosing candidate  $t$  for  $t = 0, 1, \dots, N-1$ .

Return the highest expected value  $V(t_0)$  and the optimal stopping point  $t_0$ .

(Hint: Since Python starts indices at 0, the first candidate is  $t = 0$ .)

Check your answer for  $N = 4$  with the example detailed above.

### Problem B.2: W

Write a function that takes in an integer  $M$  and runs your function from Problem B for each  $N = 3, 4, \dots, M$ . Graph the optimal stopping percentage of candidates ( $t_0/N$ ) to interview and the maximum probability  $V(t_0)$  against  $N$ . Return the optimal stopping percentage for  $M$ .

The optimal stopping percentage for  $M = 1000$  is .367.

Both the stopping time and the probability of choosing the best person converge to  $\frac{1}{e} \approx .36788$ . Then to maximize the chance of having the best marriage, you should date at least  $\frac{N}{e}$  people before choosing the next best person. This famous problem is also known as the *secretary problem*, the *sultan's dowry problem*, and the *best choice problem*. For more information, see [https://en.wikipedia.org/wiki/Secretary\\_problem](https://en.wikipedia.org/wiki/Secretary_problem).

## The Cake Eating Problem

---

Imagine you are given a cake. How do you eat it to maximize your enjoyment? Some people may prefer to eat all of their cake at once and not save any for later. Others may prefer to eat a little bit at a time. If we are to consume a cake of size  $W$  over  $T + 1$  time periods, then our consumption at each step is represented as a vector

$$\mathbf{c} = [c_0 \ c_1 \ \cdots \ c_T]^\top,$$

where

$$\sum_{i=0}^T c_i = W.$$

This vector is called a *policy vector* and describes how much cake is eaten at each time period. The enjoyment of eating a slice of cake is represented by a utility function. For some amount of consumption  $c_i \in [0, W]$ , the utility gained is given by  $u(c_i)$ .

For this lab, we assume the utility function satisfies  $u(0) = 0$ , that  $W = 1$ , and that  $W$  is cut into  $N$  equally-sized pieces so that each  $c_i$  must be of the form  $\frac{i}{N}$  for some integer  $0 \leq i \leq N$ .

### Discount Factors

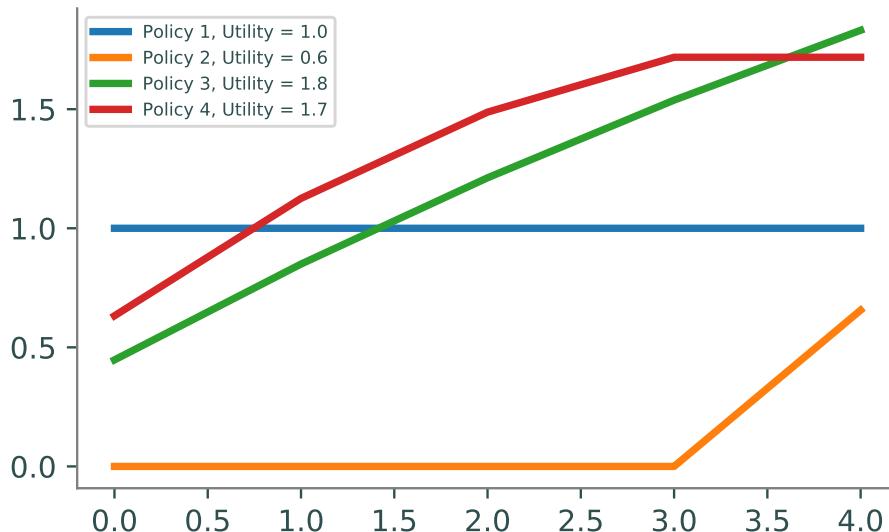
---

A person or firm typically has a time preference for saving or consuming. For example, a dollar today can be invested and yield interest, whereas a dollar received next year does not include the accrued interest. Since cake gets stale as it gets older, we assume that cake in the present yields more utility than cake in the future. We can model this by multiplying future utility by a discount factor  $\beta \in (0, 1)$ . For example, if we were to consume  $c_0$  cake at time 0 and  $c_1$  cake at time 1, with  $c_0 = c_1$  then the utility gained at time 0 is larger than the utility at time 1:

$$u(c_0) > \beta u(c_1).$$

The total utility for eating the cake is

$$\sum_{t=0}^T \beta^t u(c_t).$$



**Figure B.1:** Plots for various policies with  $u(x) = \sqrt{x}$  and  $\beta = 0.9$ . Policy 1 eats all of the cake in the first step while policy 2 eats all of the cake in the last step. Their difference in utility demonstrate the effect of the discount factor on waiting to eat. Policy 3 eats the same amount of cake at each step, while policy 4 begins by eating .4 of the cake, then .3, .2, and .1.

## The Value Function

---

The cake eating problem is an optimization problem where we maximize utility.

$$\begin{aligned} & \max_{\mathbf{c}} \sum_{t=0}^T \beta^t u(c_t) \\ & \text{subject to } \sum_{t=0}^T c_t = W \\ & \quad c_t \geq 0. \end{aligned} \tag{2.2}$$

One way to solve it is with the value function. The value function  $V(a, b, W)$  gives the utility gained from following an optimal policy from time  $a$  to time  $b$ .

$$\begin{aligned} V(a, b, W) &= \max_{\mathbf{c}} \sum_{t=a}^b \beta^t u(c_t) \\ &\text{subject to } \sum_{t=a}^b c_t = W \\ &\quad c_t \geq 0. \end{aligned}$$

$V(0, T, W)$  gives how much utility we gain in  $T$  days and is the same as Equation 2.2.

Let  $W_t$  represent the total amount of cake left at time  $t$ . Observe that  $W_{t+1} \leq W_t$  for all  $t$ , because our problem does not allow for the creation of more cake. Notice that  $V(t+1, T, W_{t+1})$  can be represented by  $\beta V(t, T-1, W_{t+1})$ , which is the value of eating  $W_{t+1}$  cake later. Then we can express the value function as the sum of the utility of eating  $W_t - W_{t+1}$  cake now and  $W_{t+1}$  cake later.

$$V(t, T, W_t) = \max_{W_{t+1}} (u(W_t - W_{t+1}) + \beta V(t, T-1, W_{t+1})) \quad (2.3)$$

where  $u(W_t - W_{t+1})$  is the value gained from eating  $W_t - W_{t+1}$  cake at time  $t$ .

Let  $\mathbf{w} = [0 \ \frac{1}{N} \ \dots \ \frac{N-1}{N} \ 1]^T$ . We define the *consumption matrix*  $C$  by  $C_{ij} = u(w_i - w_j)$ . Note that  $C$  is an  $(N+1) \times (N+1)$  lower triangular matrix since we assume  $j \leq i$ ; we can't consume more cake than we have. The consumption matrix will help solve the value function by calculating all possible value of  $u(W_t - W_{t+1})$  at once. At each time  $t$ ,  $W_t$  can only have  $N+1$  values, which will be represented as  $w_i = \frac{i}{N}$ , which is  $i$  pieces of cake remaining. For example, if  $N=4$ , then  $w = [0, .25, .5, .75, 1]^T$ , and  $w_3 = 0.75$  represents having three pieces of cake left. In this case, we get the following consumption matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ u(0.25) & 0 & 0 & 0 & 0 \\ u(0.5) & u(0.25) & 0 & 0 & 0 \\ u(0.75) & u(0.5) & u(0.25) & 0 & 0 \\ u(1) & u(0.75) & u(0.5) & u(0.25) & 0 \end{bmatrix}.$$

### Problem B.3

Write a function that accepts the number of equal sized pieces  $N$  that divides the cake and a utility function  $u(x)$ . Assume  $W = 1$ . Create a partition vector  $\mathbf{w}$  whose entries correspond to possible amounts of cake. Return the consumption matrix.

## Solving the Optimization Problem

Initially we do not know how much cake to eat at  $t = 0$ : should we eat one piece of cake ( $w_1$ ), or perhaps all of the cake ( $w_N$ )? It may not be obvious which option is best and that option may change depending on the discount factor  $\beta$ . Instead of asking how much cake to eat at some time  $t$ , we ask how valuable  $w_i$  cake is at time  $t$ . As mentioned above,  $V(t, T-1, W_{t+1})$  in 2.3 is a new value function problem with  $a = t, b = T-1$ , and  $W = W_{t+1}$ , making 2.3 a recursion formula. By using the optimal value of the value function in the future,  $V(t, T-1, W_{t+1})$ , we can determine the optimal value for the present,  $V(t, T, W_t)$ .  $V(t, T, W_t)$  can be solved by trying each possible  $W_{t+1}$  and choosing the one that gives the highest utility.

The  $(N+1) \times (T+1)$  matrix  $A$  that solves the value function is called the *value function matrix*.  $A_{ij}$  is the value of having  $w_i$  cake at time  $j$ .  $A_{0j} = 0$  because there is never any value in having  $w_0$  cake, i.e.  $u(w_0) = u(0) = 0$ .

We start at the last time period. Since there is no value in having any cake left over when time runs out, the decision at time  $T$  is obvious: eat the rest of the cake. The amount of utility gained from having  $w_i$  cake at time  $T$  is given by  $u(w_i)$ . So  $A_{iT} = u(w_i)$ . Written in the form of (2.3),

$$A_{iT} = V(0, 0, w_i) = \max_{w_j} (u(w_i - w_j) + \beta V(0, -1, w_j)) = u(w_i). \quad (2.4)$$

This happens because  $V(0, -1, w_j) = 0$ . As mentioned, there is no value in saving cake so this equation is maximized when  $w_j = 0$ . All possible values of  $w_i$  are calculated so that the value of having  $w_i$  cake at time  $T$  is known.

### ACHTUNG!

Given a time interval from  $t = 0$  to  $t = T$  the utility of waiting until time  $T$  to eat  $w_i$  cake is actually  $\beta^T u(W_i)$ . However, through backwards induction, the problem is solved backwards by beginning with  $t = T$  as an isolated state and calculating its value. This is why the value function above is  $V(0, 0, W_i)$  and not  $V(T, T, W_i)$ .

For example, the following matrix results with  $T = 3$ ,  $N = 4$ , and  $\beta = 0.9$ .

$$\begin{bmatrix} 0 & 0 & 0 & u(0) \\ 0 & 0 & 0 & u(0.25) \\ 0 & 0 & 0 & u(0.5) \\ 0 & 0 & 0 & u(0.75) \\ 0 & 0 & 0 & u(1) \end{bmatrix}.$$

### Problem B.4: W

Write a function that accepts a stopping time  $T$ , a number of equal sized pieces  $N$  that divides the cake, a discount factor  $\beta$ , and a utility function  $u(x)$ . Return the value function matrix  $A$  for  $t = T$  (the matrix should have zeros everywhere except the last column). Return a matrix of zeros for the policy matrix  $P$ .

Next, we use the fact that  $A_{jT} = V(0, 0, w_j)$  to evaluate the  $T - 1$  column of the value function matrix,  $A_{i(T-1)}$ , by modifying (2.4) as follows,

$$A_{i(T-1)} = V(0, 1, w_i) = \max_{w_j} (u(w_i - w_j) + \beta V(0, 0, w_j)) = \max_{w_j} (u(w_i - w_j) + \beta A_{jT}). \quad (2.5)$$

Remember that there is a limited set of possibilities for  $w_j$ , and we only need to consider options such that  $w_j \leq w_i$ . Instead of doing these one by one for each  $w_i$ , we can compute the options for each  $w_i$  simultaneously by creating a matrix. This information is stored in an  $(N + 1) \times (N + 1)$  matrix known as the *current value matrix*, or  $CV^t$ , where the  $(ij)$ th entry is the value of eating  $w_i - w_j$  pieces of cake at time  $t$  and saving  $j$  pieces of cake until the next period. For  $t = T - 1$ ,

$$CV_{ij}^{T-1} = u(w_i - w_j) + \beta A_{jT}. \quad (2.6)$$

The largest entry in the  $i$ th row of  $CV^{T-1}$  is the optimal value that the value function can attain at  $T - 1$ , given that we start with  $w_i$  cake. The maximal values of each row of  $CV^{T-1}$  become the column of the value function matrix,  $A$ , at time  $T - 1$ .

### ACHTUNG!

The notation  $CV^t$  does not mean raising the matrix to the  $t$ th power; rather, it indicates what time period we are in. All of the  $CV^t$  could be grouped together into a three-dimensional matrix,  $CV$ , that has dimensions  $(N + 1) \times (N + 1) \times (T + 1)$ . Although this is possible, we will not use  $CV$  in this lab, and will instead only consider  $CV^t$  for any given time  $t$ .

The following matrix is  $CV^2$  where  $T = 3$ ,  $\beta = .9$ ,  $N = 4$ , and  $u(x) = \sqrt{x}$ . The maximum value of each row, circled in red, is used in the 3<sup>rd</sup> column of  $A$ . Remember that  $A$ 's column index begins at 0, so the 3<sup>rd</sup> column represents  $j = 2$ .

$$CV^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.45 & 0 & 0 & 0 \\ 0.707 & 0.95 & 0.636 & 0 & 0 \\ 0.866 & 1.157 & 1.136 & 0.779 & 0 \\ 1 & 1.316 & 1.343 & 1.279 & 0.9 \end{bmatrix}$$

Now that the column of  $A$  corresponding to  $t = T - 1$  has been calculated, we repeat the process for  $T - 2$  and so on until we have calculated each column of  $A$ . In summary, at each time step  $t$ , find  $CV^t$  and then set  $A_{it}$  as the maximum value of the  $i$ th row of  $CV^t$ . Generalizing (2.5) and (2.6) shows

$$CV_{ij}^t = u(w_i - w_j) + \beta A_{j(t+1)}. \quad A_{it} = \max_j (CV_{ij}^t). \quad (2.7)$$

The full value function matrix corresponding to the example is below. The maximum value in the value function matrix is the maximum possible utility to be gained.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0.5 & 0.5 \\ 0.95 & 0.95 & 0.95 & 0.707 \\ 1.355 & 1.355 & 1.157 & 0.866 \\ 1.7195 & 1.562 & 1.343 & 1 \end{bmatrix}.$$

**Figure B.2:** The value function matrix where  $T = 3$ ,  $\beta = .9$ ,  $N = 4$ , and  $u(x) = \sqrt{x}$ . The bottom left entry indicates the highest utility that can be achieved is 1.7195.

**Problem B.5: C**

Complete your function from Problem B so it returns the entire value function matrix. Starting from the next to last column, iterate backwards by

- calculating the current value matrix for time  $t$  using (2.7),
- finding the largest value in each row of the current value matrix, and
- filling in the corresponding column of  $A$  with these values.

(Hint: Use axis arguments.)

## Solving for the Optimal Policy

---

With the value function matrix constructed, the optimization problem is solved in some sense. The value function matrix contains the maximum possible utility to be gained. However, it is not immediately apparent what policy should be followed by only inspecting the value function matrix  $A$ . The  $(N + 1) \times (T + 1)$  policy matrix,  $P$ , is used to find the optimal policy. The  $(ij)$ th entry of the policy matrix indicates how much cake to eat at time  $j$  if we have  $i$  pieces of cake. Like  $A$  and  $CV$ ,  $i$  and  $j$  begin at 0.

The last column of  $P$  is calculated similarly to last column of  $A$ .  $P_{iT} = w_i$ , because at time  $T$  we know that the remainder of the cake should be eaten. Recall that the column of  $A$  corresponding to  $t$  was calculated by the maximum values of  $CV^t$ . The column of  $P$  for time  $t$  is calculated by taking  $w_i - w_j$ , where  $j$  is the smallest index corresponding to the maximum value of  $CV^t$ ,

$$P_{it} = w_i - w_j.$$

$$\text{where } j = \{ \min\{j\} \mid CV_{ij}^t \geq CV_{ik}^t \forall k \in [0, 1, \dots, N] \}$$

Recall  $CV^2$  in our example with  $T = 3$ ,  $\beta = .9$ ,  $N = 4$ , and  $u(x) = \sqrt{x}$  above.

$$CV^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.45 & 0 & 0 & 0 \\ 0.707 & 0.95 & 0.636 & 0 & 0 \\ 0.866 & 1.157 & 1.136 & 0.779 & 0 \\ 1 & 1.316 & 1.343 & 1.279 & 0.9 \end{bmatrix}$$

To calculate  $P_{12}$ , we look at the second row ( $i = 1$ ) in  $CV^2$ . The maximum, .5, occurs at  $CV_{10}^2$ , so  $j = 0$  and  $P_{12} = w_1 - w_0 = .25 - 0 = .25$ . Similarly,  $P_{42} = w_4 - w_2 = 1 - .5 = .5$ . Continuing in this manner,

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.5 \\ 0.25 & 0.25 & 0.5 & 0.75 \\ 0.25 & 0.5 & 0.5 & 1 \end{bmatrix}$$

Given that the rows of  $P$  are the slices of cake available and the columns are the time intervals, we find the policy by starting in the bottom left corner,  $P_{N0}$ , where there are  $N$  slices of cake available and  $t = 0$ . This entry tells us what percentage of the  $N$  slices of cake we should eat. In the example, this entry is .25, telling us we should eat 1 slice of cake at  $t = 0$ . Thus, when  $t = 1$  we have  $N - 1$  slices of cake available, since we ate 1 slice of cake. We look at the entry at  $P_{(N-1)1}$ , which has value .25. So we eat 1 slice of cake at  $t = 1$ . We continue this pattern to find the optimal policy  $\mathbf{c} = [.25 \ 0.25 \ 0.25 \ 0.25]$ .

### ACHTUNG!

The optimal policy will not always be a straight diagonal in the example above. For example, if the bottom left corner had value .5, then we should eat 2 pieces of cake instead of 1. Then the next entry we should evaluate would be  $P_{(N-2)1}$  in order to determine the optimal policy.

To verify the optimal policy found with  $P$ , we can use the value function matrix  $A$ . By expanding the entries of  $A$ , we can see that the optimal policy does give the maximum value.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \sqrt{0.25} & \sqrt{0.25} & \sqrt{0.25} & \sqrt{0.25} \\ \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.5} \\ \sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.5} + \beta\sqrt{0.25} & \sqrt{0.75} \\ \cancel{\sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} - \beta^3\sqrt{0.25}} & \cancel{\sqrt{0.5} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25}} & \cancel{\sqrt{0.5} + \beta\sqrt{0.5}} & \sqrt{1} \end{bmatrix}$$

### Problem B.6: M

*Modify your function from Problem B to determine the policy matrix. Initialize the matrix as zeros and fill it in starting from the last column at the same time that you calculate the value function matrix.*

*(Hint: You may find `np.argmax()` useful.)*

### Problem B.7: W

*Create a function `find_policy()` that will find the optimal policy for the stopping time  $T$ , a cake of size 1 split into  $N$  pieces, a discount factor  $\beta$ , and the utility function  $u$ .*

## B.1 Contributors



### Champions of Access to Knowledge



#### OPEN TEXT

All digital forms of access to our high-quality open texts are entirely FREE! All content is reviewed for excellence and is wholly adaptable; custom editions are produced by Lyryx for those adopting Lyryx assessment. Access to the original source files is also open to anyone!



#### ONLINE ASSESSMENT

We have been developing superior online formative assessment for more than 15 years. Our questions are continuously adapted with the content and reviewed for quality and sound pedagogy. To enhance learning, students receive immediate personalized feedback. Student grade reports and performance statistics are also provided.



#### SUPPORT

Access to our in-house support team is available 7 days/week to provide prompt resolution to both student and instructor inquiries. In addition, we work one-on-one with instructors to provide a comprehensive system, customized for their course. This can include adapting the text, managing multiple sections, and more!



#### INSTRUCTOR SUPPLEMENTS

Additional instructor resources are also freely accessible. Product dependent, these supplements include: full sets of adaptable slides and lecture notes, solutions manuals, and multiple choice question banks with an exam building tool.

### Contact Lyryx Today!

[info@lyryx.com](mailto:info@lyryx.com)

<sup>1</sup>This book was not produced by Lyryx, but this book has made substantial use of their open source material. We leave this page in here as a tribute to Lyryx for sharing their content.



## BE A CHAMPION OF OER!

Contribute suggestions for improvements, new content, or errata:

- A new topic
- A new example
- An interesting new question
- A new or better proof to an existing theorem
- Any other suggestions to improve the material

Contact Lyryx at [info@lyryx.com](mailto:info@lyryx.com) with your ideas.

## CONTRIBUTIONS

Ilijas Farah, York University

Ken Kuttler, Brigham Young University

**Lyryx Learning Team**

# Foundations of Applied Mathematics

<https://github.com/Foundations-of-Applied-Mathematics>

## CONTRIBUTIONS

### List of Contributors

E. Evans	J. Fisher
<i>Brigham Young University</i>	<i>Brigham Young University</i>
R. Evans	R. Flores
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Grout	R. Fowers
<i>Drake University</i>	<i>Brigham Young University</i>
J. Humpherys	A. Frandsen
<i>Brigham Young University</i>	<i>Brigham Young University</i>
T. Jarvis	R. Fuhriman
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Whitehead	S. Giddens
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Adams	C. Gigena
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Bejarano	M. Graham
<i>Brigham Young University</i>	<i>Brigham Young University</i>
Z. Boyd	F. Glines
<i>Brigham Young University</i>	<i>Brigham Young University</i>
M. Brown	C. Glover
<i>Brigham Young University</i>	<i>Brigham Young University</i>
A. Carr	M. Goodwin
<i>Brigham Young University</i>	<i>Brigham Young University</i>
C. Carter	R. Grout
<i>Brigham Young University</i>	<i>Brigham Young University</i>
T. Christensen	D. Grundvig
<i>Brigham Young University</i>	<i>Brigham Young University</i>
M. Cook	E. Hannesson
<i>Brigham Young University</i>	<i>Brigham Young University</i>
R. Dorff	J. Hendricks
<i>Brigham Young University</i>	<i>Brigham Young University</i>
B. Ehlert	A. Henriksen
<i>Brigham Young University</i>	<i>Brigham Young University</i>
M. Fabiano	I. Henriksen
<i>Brigham Young University</i>	<i>Brigham Young University</i>
K. Finlinson	C. Hettinger
<i>Brigham Young University</i>	<i>Brigham Young University</i>

## 2 ■ Dynamic Programming

S. Horst	H. Ringer
<i>Brigham Young University</i>	<i>Brigham Young University</i>
K. Jacobson	C. Robertson
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Leete	M. Russell
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Lytle	R. Sandberg
<i>Brigham Young University</i>	<i>Brigham Young University</i>
R. McMurray	C. Sawyer
<i>Brigham Young University</i>	<i>Brigham Young University</i>
S. McQuarrie	M. Stauffer
<i>Brigham Young University</i>	<i>Brigham Young University</i>
D. Miller	J. Stewart
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Morrise	S. Suggs
<i>Brigham Young University</i>	<i>Brigham Young University</i>
M. Morrise	A. Tate
<i>Brigham Young University</i>	<i>Brigham Young University</i>
A. Morrow	T. Thompson
<i>Brigham Young University</i>	<i>Brigham Young University</i>
R. Murray	M. Victors
<i>Brigham Young University</i>	<i>Brigham Young University</i>
J. Nelson	J. Webb
<i>Brigham Young University</i>	<i>Brigham Young University</i>
E. Parkinson	R. Webb
<i>Brigham Young University</i>	<i>Brigham Young University</i>
M. Probst	J. West
<i>Brigham Young University</i>	<i>Brigham Young University</i>
M. Proudfoot	A. Zaitzeff
<i>Brigham Young University</i>	<i>Brigham Young University</i>
D. Reber	
<i>Brigham Young University</i>	

This project is funded in part by the National Science Foundation, grant no. TUES Phase II DUE-1323785.

### B.1.1. Graph Theory

---

Chapter on Graph Theory adapted from: CC-BY-SA 3.0 Math in Society A survey of mathematics for the liberal arts major Math in Society is a free, open textbook. This book is a survey of contemporary mathematical topics, most non-algebraic, appropriate for a college-level quantitative literacy topics course for liberal arts majors. The text is designed so that most chapters are independent, allowing the instructor to choose a selection of topics to be covered. Emphasis is placed on the applicability of the mathematics. Core material for each topic is covered in the main text, with additional depth available through exploration exercises appropriate for in-class, group, or individual investigation. This book is appropriate for Washington State Community Colleges' Math 107.

The current version is 2.5, released Dec 2017. <http://www.opentextbookstore.com/mathinsociety/2.5/GraphTheory.pdf>

Communicated by Tricia Muldoon Brown, Ph.D. Associate Professor of Mathematics Georgia Southern University Armstrong Campus Savannah, GA 31419 <http://math.armstrong.edu/faculty/brown/MATH1001.html>

