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: April 1, 2022

Table of Contents

This entire book is a working manuscript.

MAJOR ACKNOWLEDGEMENTS

Contents

II Discrete Algorithms

13 Dynamic Programming

14 Graph Algorithms

III Appendix - Linear Algebra Background

A Linear Transformations

B Linear Systems

C Systems of Equations

D Matrices

IV　Other Appendices　　　281

A　Some Prerequisite Topics　　　283

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 Optimizaiton Notes by Komei Fukuda

- Convex Optimization by Bord 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 PolerJosefa 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 Orielly.com.

- Lesson files for the Winston Book

- Video instructions for solver and an example workbook

-

GUROBI LINKS
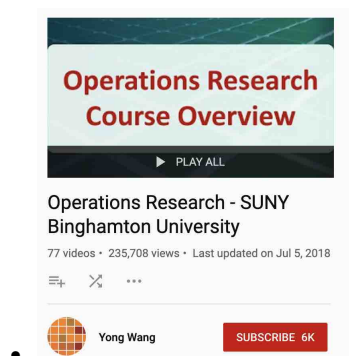
- Go to `https://github.com/Gurobi` and download the example files.

- 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

- 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. $1, 2, 3, -1, -2, -3, ...$)

- $\mathbb{Q}$ - 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. $1, 1.5, \pi, e, -11/5$)

- $\backslash$ - setminus, (e.g. $\{0,1,2,3\} \backslash \{0,3\} = \{1,2\}$)

- $\cup$ - 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. $[3, \pi, -e, \sqrt{2}]$)

- $\sum_{i=1}^4 i = 1+2+3+4$

- $\sum_{i=1}^4 i^2 = 1^2 + 2^2 + 3^2 + 4^4$

- $\sum_{i=1}^4 x_i = x_1 + x_2 + x_3 + x_4$

- $\square$ - this is a typical Q.E.D. symbol that you put at the end of a proof meaning "I proved it."

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

"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).
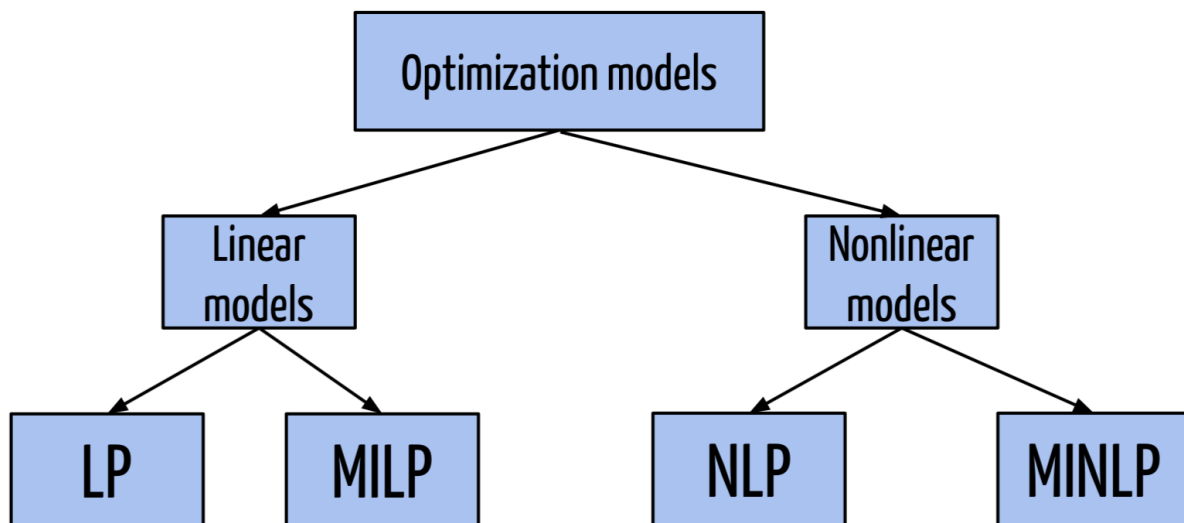
© **problem-class-diagram**[1]

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 **??** 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 **??**.

---

[1]**problem-class-diagram**, from **problem-class-diagram**. **problem-class-diagram**, **problem-class-diagram**.

Linear Programming (LP):

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 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

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.2}$$
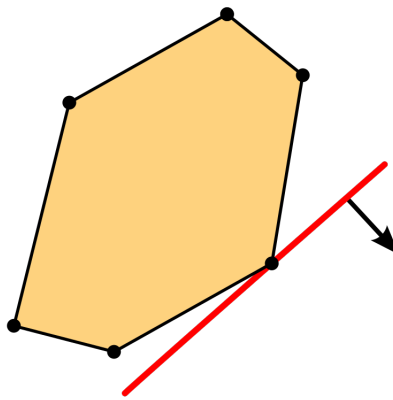
© wiki/File/linear-programming.png[2]

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.

2.2 Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming will be the focus of Sections **??**,**??**, **??**, 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.

Figure 2.3

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

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

---

[3]**wiki/File/integer-programming.png**, from **wiki/File/integer-programming.png**. **wiki/File/integer-programming.png**, **wiki/File/integer-programming.png**.
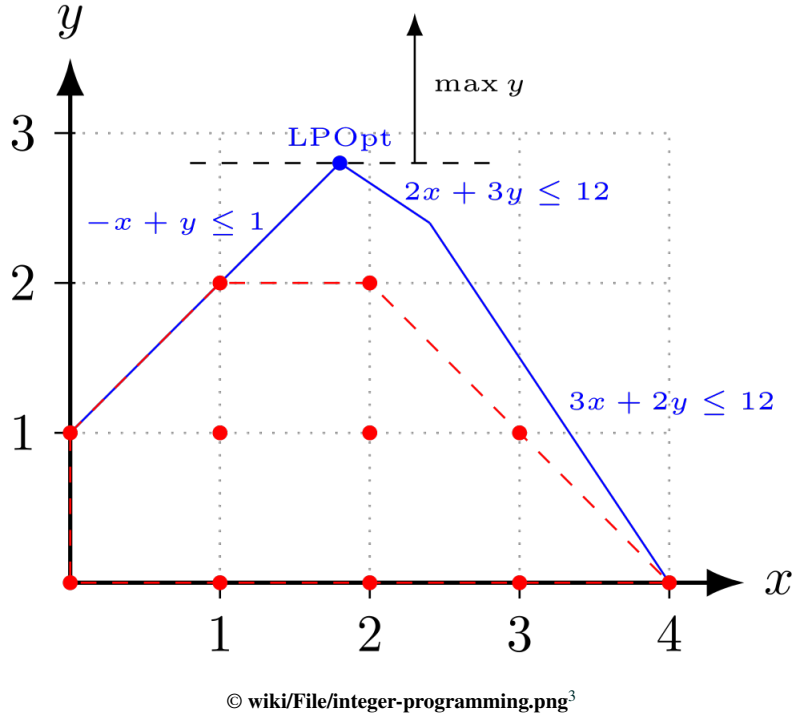
Figure 2.3: Comparing the LP relaxation to the IP solutions.

An even more general class is *Mixed-Integer Linear Programming (MILP)*. This is where we have n integer variables $x_1, \ldots, x_n \in \mathbb{Z}$ and d continuous variables $x_{n+1}, \ldots, x_{n+d} \in \mathbb{R}$. 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 $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and other functions $f_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1,\dots,m$, the *nonlinear programming* problem is

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

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

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

Example 2.2

onvex 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

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

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 [Under Construction]

This part of the book needs substantial revision. It is currently comprised of 3 sets of notes pasted together.

3. Linear Programming

A Generic Linear Program (LP)

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

Parameters (known input parameters):
c_i : cost coefficients $\forall i = 1, \ldots, n$
a_{ij} : constraint coefficients $\forall i = 1, \ldots, n, \ j = 1, \ldots, m$
b_j : right hand side coefficient for constraint $j, \ j = 1, \ldots, m$

The problem we will consider is

$$
\begin{aligned}
\max \ & z = c_1 x_1 + \cdots + c_n x_n \\
\text{s.t.} \ & a_{11} x_1 + \cdots + a_{1n} x_n \leq b_1 \\
& \vdots \\
& a_{m1} x_1 + \cdots + a_{mn} x_n \leq b_m
\end{aligned}
\tag{3.1}
$$

For example, in 3 variables and 4 contraints 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), $\forall i = 1, \cdots, 3$.

Parameters (known input parameters):
c_i : cost coefficients $\forall i = 1, \ldots, 3$
a_{ij} : constraint coefficients $\forall i = 1, \ldots, 3, \ j = 1, \ldots, 4$
b_j : right hand side coefficient for constraint $j, \ j = 1, \ldots, 4$

$$
\begin{aligned}
\text{Min} \ & z = c_1 x_1 + c_2 x_2 + c_3 x_3 & (3.2) \\
\text{s.t.} \ & a_{11} x_1 + a_{12} x_2 + a_{13} x_3 \geq b_1 & (3.3) \\
& a_{21} x_1 + a_{22} x_2 + a_{23} x_3 \leq b_2 & (3.4) \\
& a_{31} x_1 + a_{32} x_2 + a_{33} x_3 = b_3 & (3.5) \\
& a_{41} x_1 + a_{42} x_2 + a_{43} x_3 \geq b_4 & (3.6) \\
& x_1 \geq 0, x_2 \leq 0, x_3 \ urs. & (3.7)
\end{aligned}
$$

Definition 3.1: Linear Function

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

$$z(x_1, \ldots, x_n) = c_1 x_1 + \cdots + c_n x_n \tag{3.8}$$

Lemma 3.2: Linear Function

If $z : \mathbb{R}^n \to \mathbb{R}$ 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:

$$z(\mathbf{x}_1 + \mathbf{x}_2) = z(\mathbf{x}_1) + z(\mathbf{x}_2) \tag{3.9}$$
$$z(\alpha \mathbf{x}_1) = \alpha z(\mathbf{x}_1) \tag{3.10}$$

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:

$$z(x_1, x_2) = 7x_1 + 6x_2 \tag{3.11}$$

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:

$$3x_1 + x_2 \leq 120 \tag{3.12}$$

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:*

$$x_1 + 2x_2 \leq 160 \tag{3.13}$$

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:*

$$\begin{cases} \max\ z(x_1,x_2) = 7x_1 + 6x_2 \\ \text{s.t.}\ 3x_1 + x_2 \leq 120 \\ \qquad x_1 + 2x_2 \leq 160 \\ \qquad x_1 \leq 35 \\ \qquad x_1 \geq 0 \\ \qquad x_2 \geq 0 \end{cases} \tag{3.14}$$

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 chosing 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 $(x_1,\ldots,x_n)$ 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 ($i = 1, \ldots, n$) 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 $x_1, \ldots, x_n$.

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 ($i = 1, \ldots, n$) 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

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:

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

♠

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: <u>Sets:</u>

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

<u>Parameters:</u>

- Iron in each tablet

- Vitamin B in each tablet

- Vitamin C in each tablet

- Vitamin D in each tablet

- Cost of each tablet

<u>Decision variables:</u>
x_i : number of tablet of type i to include in the diet, $\forall i \in \{1,2,3,4\}$.

<u>Objective and Constraints:</u>

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

♠

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.

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

Solution. <u>Sets:</u>

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

<u>Parameters:</u>

- Iron in each chemical

- Vitamin B in each chemical

- Vitamin C in each chemical

- Vitamin D in each chemical

- Cost of each chemical

<u>Decision variables:</u>
x_i : grams of chemical i to include in the purple tablet, $\forall i = 1,2,3,4$.
<u>Objective and Constraints:</u>

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

♠

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 employees 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. <u>Decision variables:</u>

<u>Decision variables:</u>
x_i : the number of workers that start 5 consecutive days of work on day i, $i = 1, \cdots, 7$

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

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 it's 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, $i = 1, \cdots, 7$

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

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

♠

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.

$$\text{Max } z = \sum_{i=1}^{n}\sum_{j=1}^{n} R_{ij}x_{ij}$$

$$\text{s.t. } \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, \cdots, n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = 1, \cdots, n$$

$$x_{ij} \geq 0, \quad \forall i = 1, \cdots, n, j = 1, \cdots, n.$$

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

$$\textit{Max } \; z$$

$$\textit{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, \cdots, n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = 1, \cdots, n$$

$$x_{ij} \geq 0, \quad \forall i = 1, \cdots, n, J = 1, \cdots, n$$

$$z \leq \sum_{i=1}^{n} R_{ij}x_{ij}, \quad \forall j = 1, \cdots, n.$$

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

3.2.4.1. Production Planning

3.2.4.2. Crop Planning

3.2.5. Mixing Problems

3.2.6. Financial Planning

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 $E = \{(4,6), (4,5), (5,1)(1,2), (2,5), (2,3), (3,4)\}$.
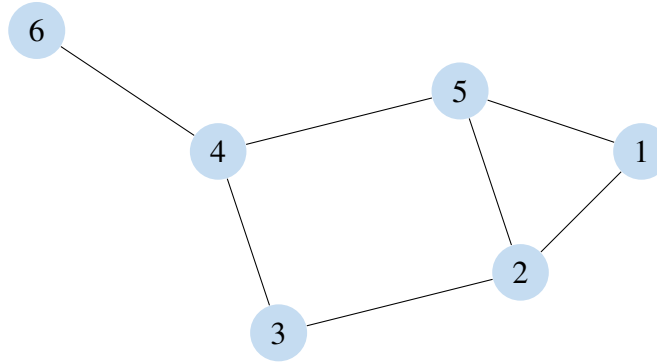
In an undirected graph, we do not distinguish the direction of the edge. That is, for two vertices $i, j \in V$, we can equivalently write (i, j) or (j, i) to represent the edge.

Alternatively, we will want to consider directed graphs. We denote these as $G = (V, \mathscr{A})$ where $\mathscr{A}$ 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 $\mathscr{A} = \{(4,6), (6,4), (4,5), (5,1)(1,2), (2,5), (2,3), (3,4)\}$.
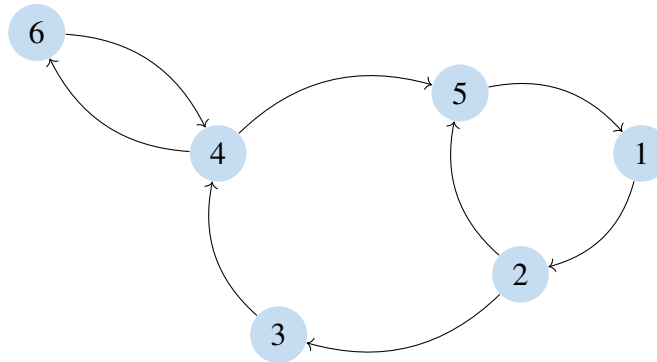
SETS   A finite network G is described by a finite set of vertices V and a finite set $\mathscr{A}$ of arcs. Each arc (i, j) has two key attributes, namely its tail $j \in V$ and its head $i \in V$.

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

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

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

3.2.7.2. Maximum Flow Problem

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

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

$$0 \le x_{ij} \le u_{ij} \quad \forall(i,j)\in\mathscr{A} \qquad (3.3)$$

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

SHORTEST PATH PROBLEM

Example 3.13: Max flow example
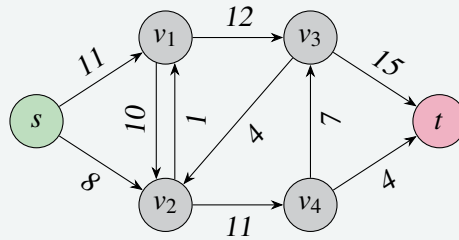
Example 3.14: Min Cost Network Flow
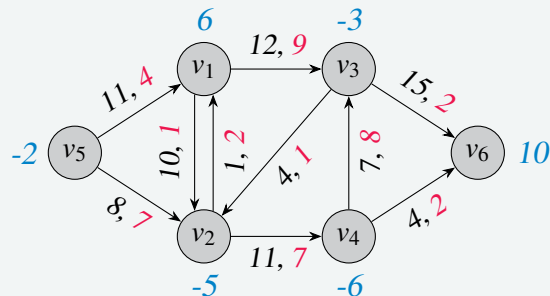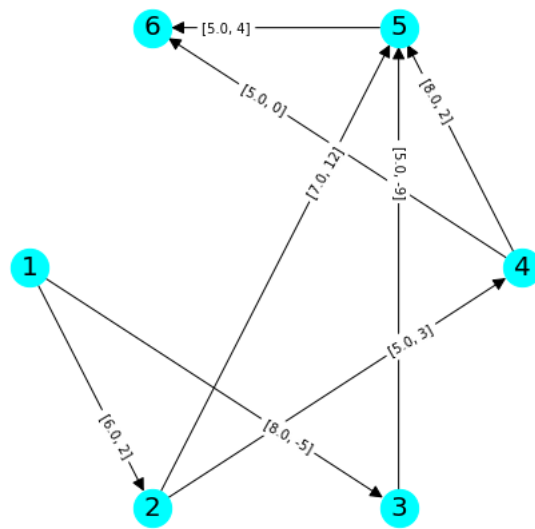
© **network-flow**[1]

Figure 3.1: network-flow
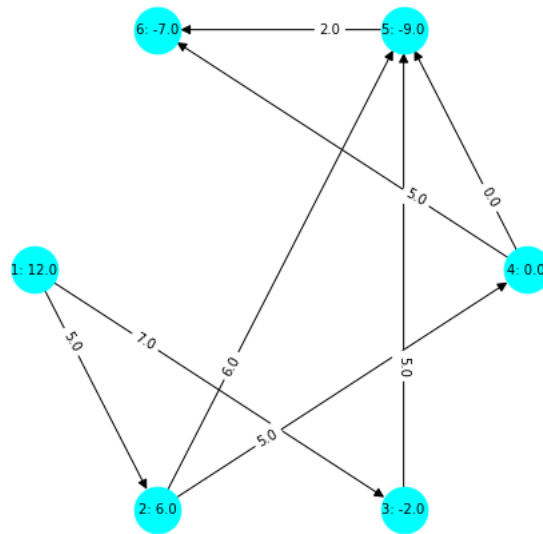
© **network-flow-solution**[2]

Figure 3.2: network-flow-solution

---

[1]**network-flow**, from **network-flow**. **network-flow**, **network-flow**.

[2]**network-flow-solution**, from **network-flow-solution**. **network-flow-solution**, **network-flow-solution**.

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

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

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

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

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

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

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:

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

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

$$\min \sum_{k=1}^{K} \sum_{e \in \mathscr{A}} c_e^k x_e^k$$

$$\sum_{e \in \mathscr{A} \,:\, t(e)=v} x_e^k - \sum_{e \in \mathscr{A} \,:\, h(e)=v} x_e^k = b_v^k, \;\; \text{for } v \in \mathscr{N}, \, k = 1, 2, \ldots, K;$$

$$\sum_{k=0}^{K} x_e^k \le u_e, \;\; \text{for } e \in \mathscr{A};$$

$$x_e^k \ge 0, \;\; \text{for } e \in \mathscr{A}, \, k = 1, 2, \ldots, K$$

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 probem. Nontheless, if the quanties in each flow are very large, then the LP solution will likely be very close to an integer valued solution.*

3.2.9. Modeling Tricks

> **Example 3.17: Minimizing a Maximum**
>
> *let.....*

> **Example 3.18: Minimizing an Absolute Value**

3.3 Other examples

Food manfacturing - GUROBI

Optimization Methods in Finance - Corneujols, 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.[1] 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

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

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

- $X + 2Y = 16$

- $5X + 3Y = 45$

---

[1]Special thanks to Joshua Emmanual and Christopher Griffin for sharing their content to help put this section together. Proper citations and referenes are forthcoming.

- $X = 0$

- $Y = 0$

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

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

$$X + 2Y \leq 16$$

X	Y
0	8
16	0

$$5X + 3Y \leq 45$$

X	Y
0	15
9	0
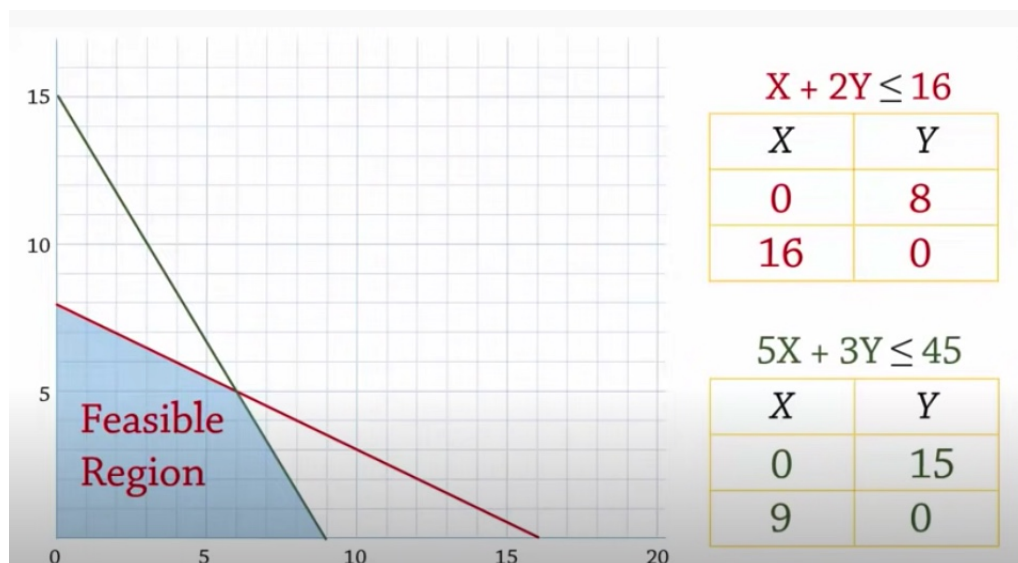
Notice that the feasible region is nonempty (it has points that satisfy all the inequalities) and also that it is bounded (the feasible points dont continue infinitly in any direction).

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).

Point	$Z = 2X + 5Y$
❶ (0, 0)	0
❷ (0, 8)	40
❸ (6, 5)	37
❹ (9, 0)	18
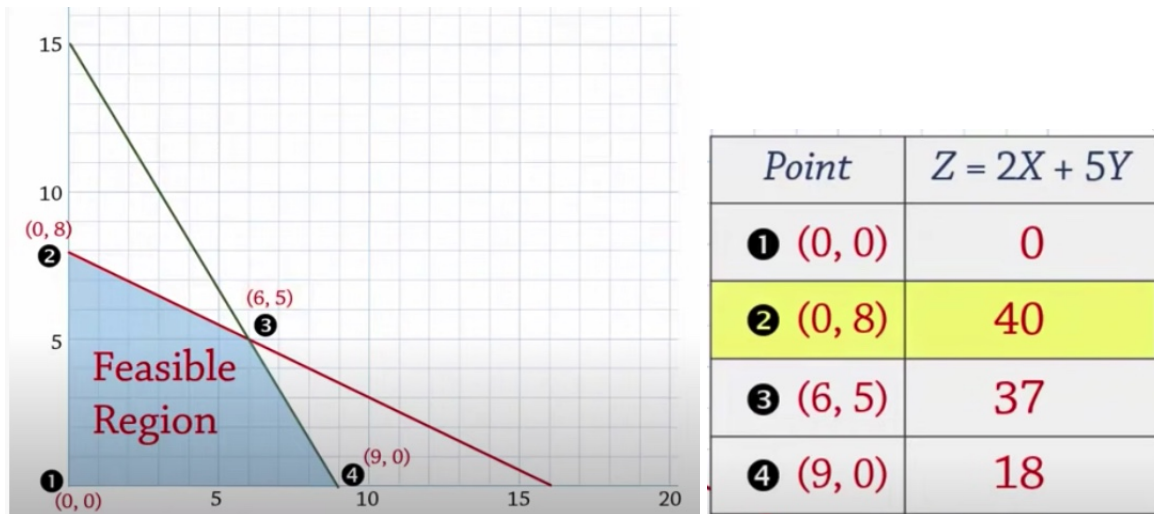
Theorem 4.1: Optimal Extreme Point

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 assumtions of either nonempty or bounded. We illustrate the idea first using the problem from Example 3.
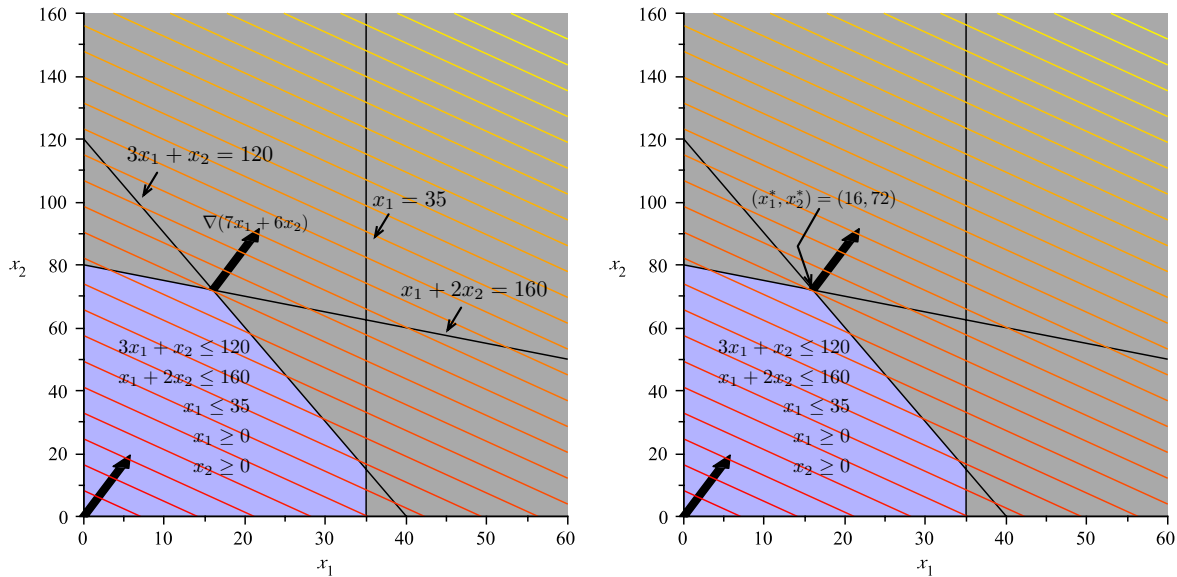
Figure 4.1: Feasible Region and Level Curves of the Objective Function: The shaded region in the plot is the feasible region and represents the intersection of the five inequalities constraining the values of x_1 and x_2. On the right, we see the optimal solution is the "last" point in the feasible region that intersects a level set as we move in the direction of increasing profit.

Example 4.2: Continuation of Example

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

$$3x_1 + x_2 \leq 120$$
$$x_1 + 2x_2 \leq 160$$

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 programing 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 infinitly 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:

$$\begin{cases} \max\ z(x_1,x_2) = 18x_1 + 6x_2 \\ s.t.\ 3x_1 + x_2 \leq 120 \\ \qquad x_1 + 2x_2 \leq 160 \\ \qquad x_1 \leq 35 \\ \qquad x_1 \geq 0 \\ \qquad x_2 \geq 0 \end{cases} \tag{4.1}$$

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.
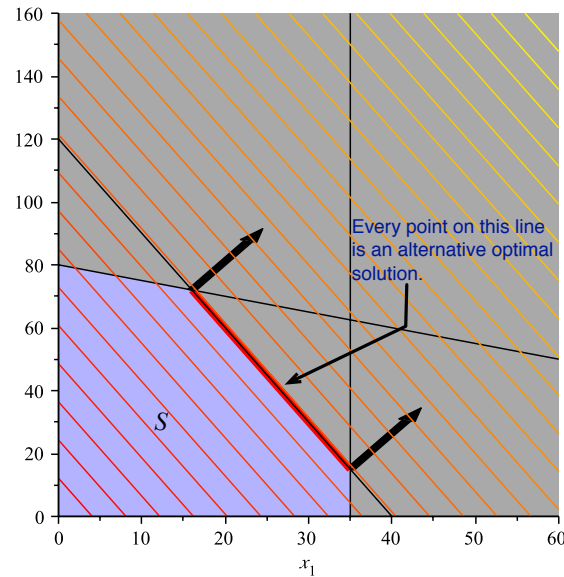
Figure 4.2: **An example of infinitely many alternative optimal solutions in a linear programming problem. The level curves for $z(x_1, x_2) = 18x_1 + 6x_2$ are *parallel* to one face of the polygon boundary of the feasible region. Moreover, this side contains the points of greatest value for $z(x_1, x_2)$ 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 value 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

onsider the following linear programming problem:

$$\begin{cases} \max\ z(x_1,x_2) = 3x_1 + 2x_2 \\ s.t.\ \dfrac{1}{40}x_1 + \dfrac{1}{60}x_2 \le 1 \\ \qquad \dfrac{1}{50}x_1 + \dfrac{1}{50}x_2 \le 1 \\ \qquad x_1 \ge 30 \\ \qquad x_2 \ge 20 \end{cases} \tag{4.1}$$

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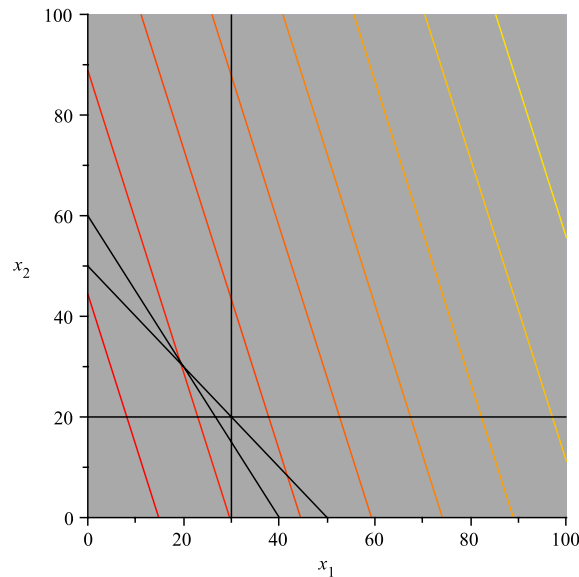
Figure 4.3: A Linear Programming Problem with no solution. The feasible region of the linear programming problem is empty; that is, there are no values for x_1 and x_2 that can simultaneously satisfy all the constraints. Thus, no solution exists.

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

$$
\begin{aligned}
\min \quad & Z = 5X + 7Y \\
\text{s.t.} \quad & X + 3Y \geq 6 \\
& 5X + 2Y \geq 10 \\
& Y \leq 4 \\
& X, Y \geq 0
\end{aligned}
$$
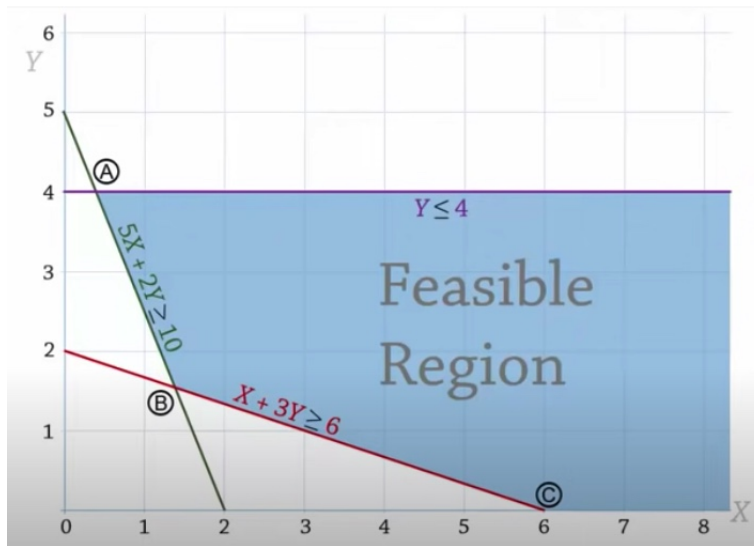
As you can see, the feasible region is *unbounded*. In particular, from any point in the feasible region, one can always find another feasible point by increasing the X coordinate (i.e., move to the right in the picture). However, this does not necessarily mean that the optimization problem is unbounded.

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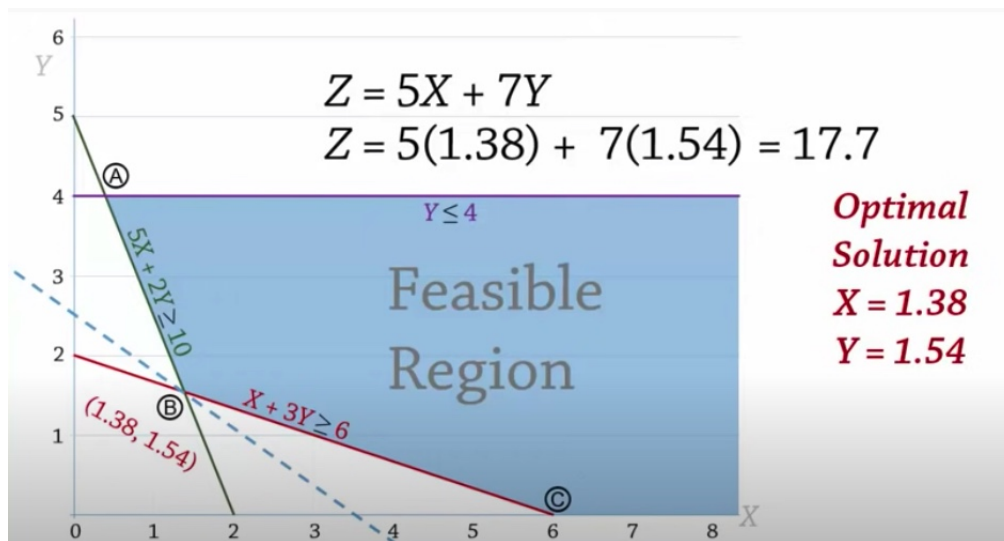
Consider however, if we consider a different problem where we try to maximize the objective

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

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

$$\begin{cases} \max \ z(x_1, x_2) = 2x_1 - x_2 \\ s.t. \ x_1 - x_2 \leq 1 \\ \qquad 2x_1 + x_2 \geq 6 \\ \qquad x_1, x_2 \geq 0 \end{cases} \qquad (4.1)$$

Solution. The feasible region and level curves of the objective function are shown in Figure 4.4.
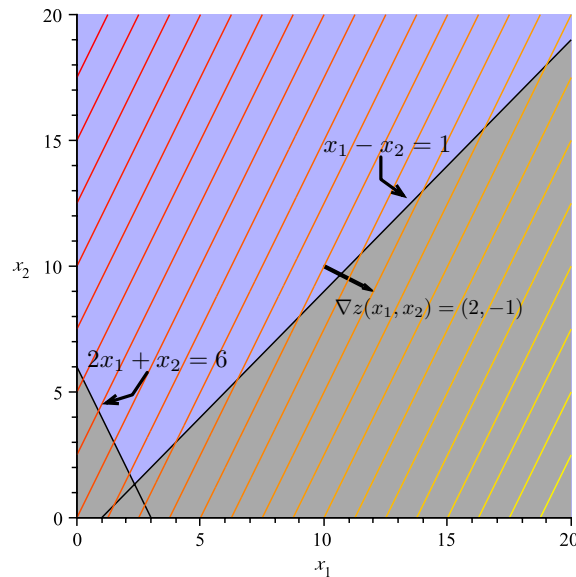
Figure 4.4: A Linear Programming Problem with Unbounded Feasible Region: Note that we can continue to make level curves of $z(x_1, x_2)$ corresponding to larger and larger values as we move down and to the right. These curves will continue to intersect the feasible region for any value of $v = z(x_1, x_2)$ 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**
>
> *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:*
>
> $$\begin{cases} \max \ z(x_1, x_2) = \dfrac{1}{2}x_1 - x_2 \\ s.t. \ x_1 - x_2 \leq 1 \\ \quad \ 2x_1 + x_2 \geq 6 \\ \quad \ x_1, x_2 \geq 0 \end{cases} \tag{4.2}$$

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.
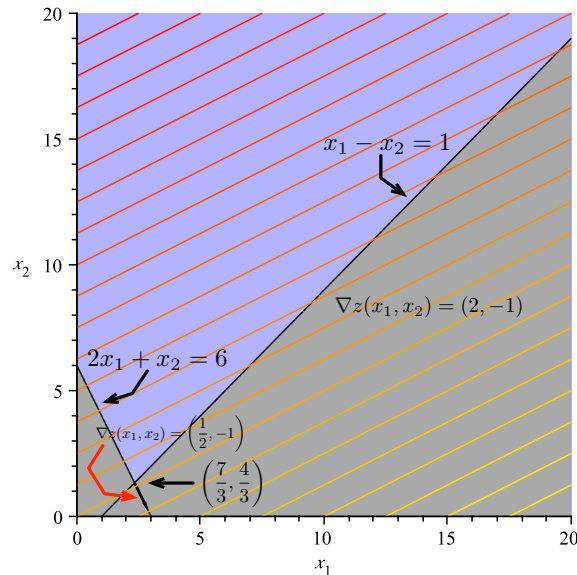
Figure 4.5: A Linear Programming Problem with Unbounded Feasible Region and Finite Solution: In this problem, the level curves of $z(x_1, x_2)$ increase in a more "southernly" direction that in Example 4.4–that is, *away* from the direction in which the feasible region increases without bound. The point 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.

Exercise 4.9

Does the following problem have a bounded solution? Why?

$$\begin{cases} \min \ z(x_1, x_2) = 2x_1 - x_2 \\ s.t. \ x_1 - x_2 \leq 1 \\ \qquad 2x_1 + x_2 \geq 6 \\ \qquad x_1, x_2 \geq 0 \end{cases} \tag{4.3}$$

[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.

**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 4.11

*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 **n** has n-elements and represents a point (or an arrow from the origin to the point, denoting a direction) in $\mathscr{R}^n$ 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 **b** is a **linear combination** of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ if $\mathbf{b} = \sum_{i=1}^{k} \lambda_i \mathbf{a}_i$ for $\lambda_1, \lambda_2, \cdots, \lambda_k \in \mathscr{R}$. If $\lambda_1, \lambda_2, \cdots, \lambda_k \in \mathscr{R}_{\geq 0}$ then **b** is a *non-negative linear combination* of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$.

Convex Combination: The vector **b** is a **convex combination** of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ if $\mathbf{b} = \sum_{i=1}^{k} \lambda_i \mathbf{a}_i$, for $\lambda_1, \lambda_2, \cdots, \lambda_k \in \mathscr{R}_{\geq 0}$ 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 $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ are *linearly independent* if the following linear combination $\sum_{i=1}^{k} \lambda_i \mathbf{a}_i = 0$ implies that $\lambda_i = 0$, $i = 1, 2, \cdots, k$. In $\mathscr{R}^2$ two vectors are only linearly dependent if they lie on the same line. Can you have three linearly independent vectors in $\mathscr{R}^2$?

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

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

Convex and Polyhedral Sets

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

Hyperplanes and Half-Spaces: A hyperplane in $\mathscr{R}^n$ divides $\mathscr{R}^n$ into 2 half-spaces (like a line does in $\mathscr{R}^2$). 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 $\mathscr{S} = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$, 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 $\mathscr{C} = \{\mathbf{x} : \mathbf{A}\mathbf{x} \leq 0\}$ is a polyhedral cone.

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

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

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

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

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 $\mathscr{R}^n$ 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 $\mathscr{C}$ is convex cone, then for any $\mathbf{x} \in \mathscr{C}$ we

have $\lambda \mathbf{x} \in \mathscr{C}$, $\lambda \geq 0$.

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

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

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

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.

$$\begin{aligned} \max \ z &= -5x_1 - x_2 \\ \text{s.t.} \ x_1 - 4x_2 + s_1 &= 0 \\ -x_1 + x_2 + s_2 &= 0 \\ -x_1 + 2x_2 + s_3 &= 0 \\ x_1 + x_2 &= 1 \\ x_1, x_2, s_1, s_2, s_3 &\geq 0. \end{aligned}$$
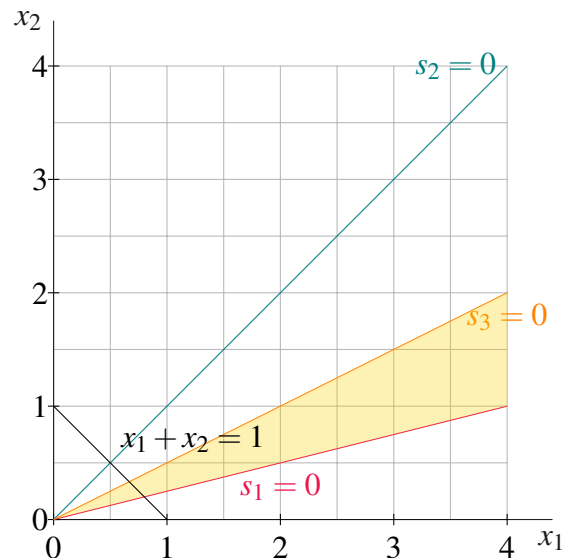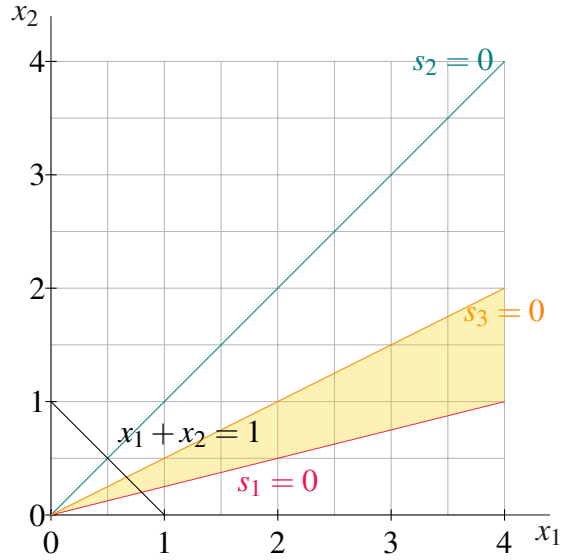
Magnifying for clarity, and removing the $s_2 = 0$ (teal) line, as it is redundant, and marking the extreme points of the new feasible region, (4/5, 1/5) and (2/3, 1/3), with red boxes, we have:
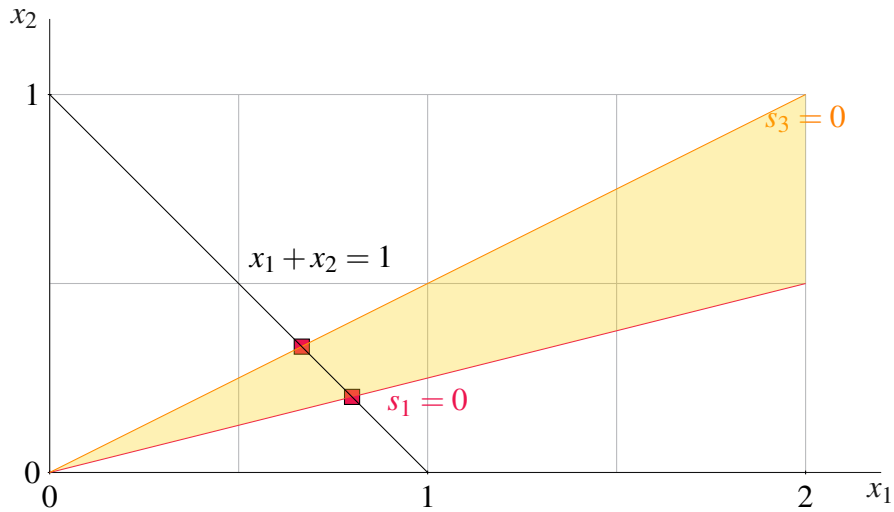
The extreme directions are thus (4/5, 1/5) and (2/3, 1/3).

Representation Theorem: Let $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x_k}$ be the set of extreme points of $\mathscr{S}$, and if $\mathscr{S}$ is unbounded, $\mathbf{d}_1, \mathbf{d}_2, \cdots \mathbf{d_l}$ be the set of extreme directions. Then any $\mathbf{x} \in \mathscr{S}$ 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 $\sum_{j=1}^{k} \lambda_j = 1$, $\lambda_j \geq 0$, $\forall j = 1, 2, \cdots, k$, and $\mu_j \geq 0$, $\forall j = 1, 2, \cdots, l$.

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

Represent point $(1/2, 1)$ as a convex combination of the extreme points of the above LP. Find λs 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

5.1 Excel

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

5.1.1. Excel Solver

5.1.2. 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

5.1.3. Links

Loan Example

Several Examples including TSP

6. 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 **all** 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/` 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/`.
>
> - 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, Juptyer 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 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.

7. 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.

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

Slicing

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

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

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

Stacking

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

$$A = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \qquad\qquad B = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$$

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

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

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()`.

$$\texttt{x} = \begin{bmatrix} \times & \times & \times & \times \end{bmatrix} \qquad\qquad \texttt{y} = \begin{bmatrix} * & * & * & * \end{bmatrix}$$

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

$$\texttt{np.vstack((x,y,x))} = \begin{bmatrix} \times & \times & \times & \times \\ * & * & * & * \\ \times & \times & \times & \times \end{bmatrix} \qquad \texttt{np.column\_stack((x,y,x))} = \begin{bmatrix} \times & * & \times \\ \times & * & \times \\ \times & * & \times \\ \times & * & \times \end{bmatrix}$$

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.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \qquad\qquad x = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix}$$

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

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

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`.

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

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

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

8. Plot Customization and Matplotlib Syntax Guide

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 8.1.

```
import numpy as np
from matplotlib import pyplot as plt

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

for i in xrange(8):
    plt.plot(x, i*y, colors[i], linewidth=18)

```
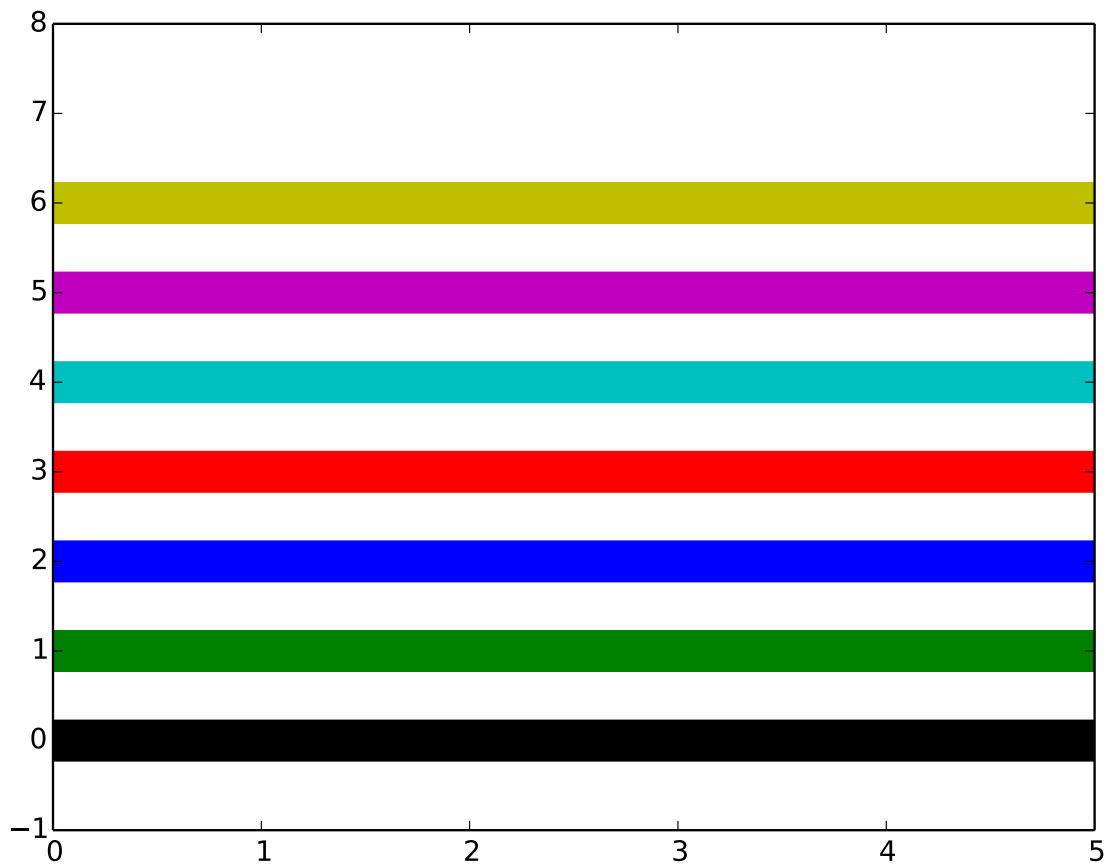
Figure 8.1: A display of all the built-in colors.

```
plt.ylim([-1, 8])
plt.savefig("colors.pdf", format='pdf')
plt.clf()
```

colors.py

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. Similarily, `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 8.2.

```
lw = np.linspace(.5, 15, 8)

for i in xrange(8):
    plt.plot(x, i*y, colors[i], linewidth=lw[i])

plt.ylim([-1, 8])
plt.show()
```

linewidth.py

Style

By default, plots are drawn with a solid line. The following are accepted format string characters to indicate line style.
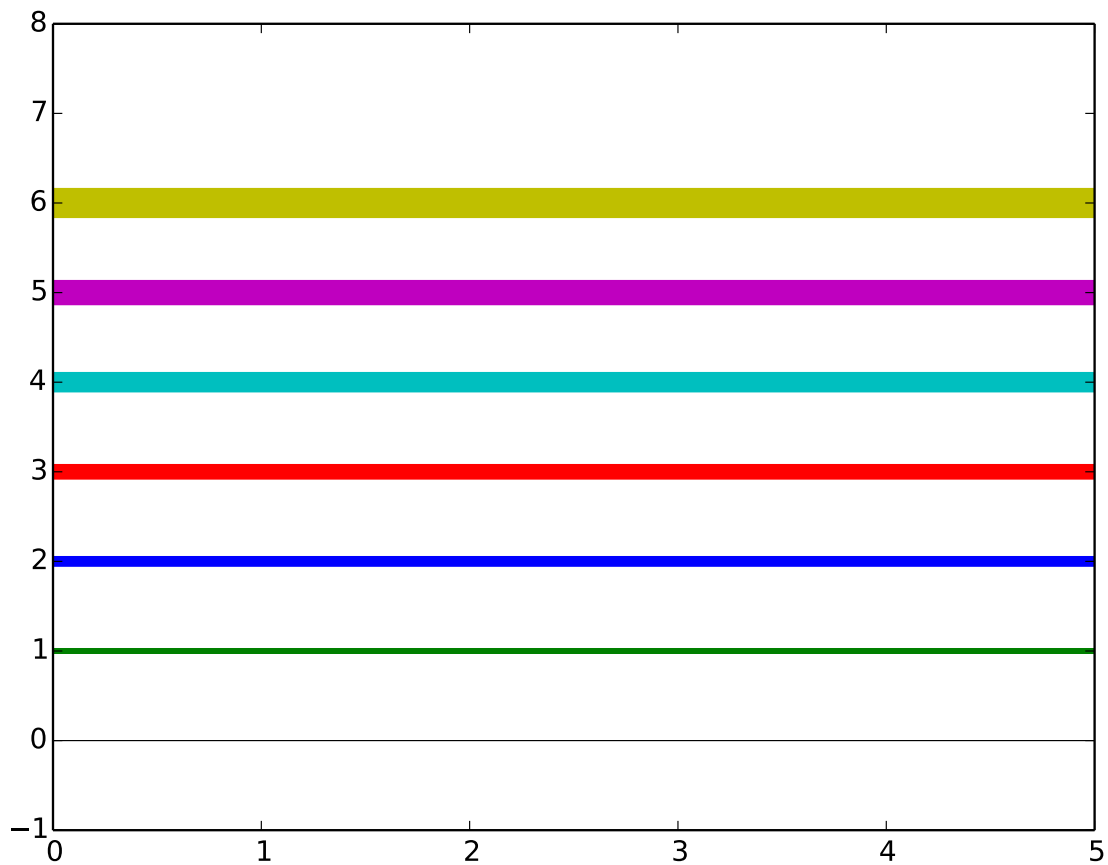
Figure 8.2: plot of varying linewidths.

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
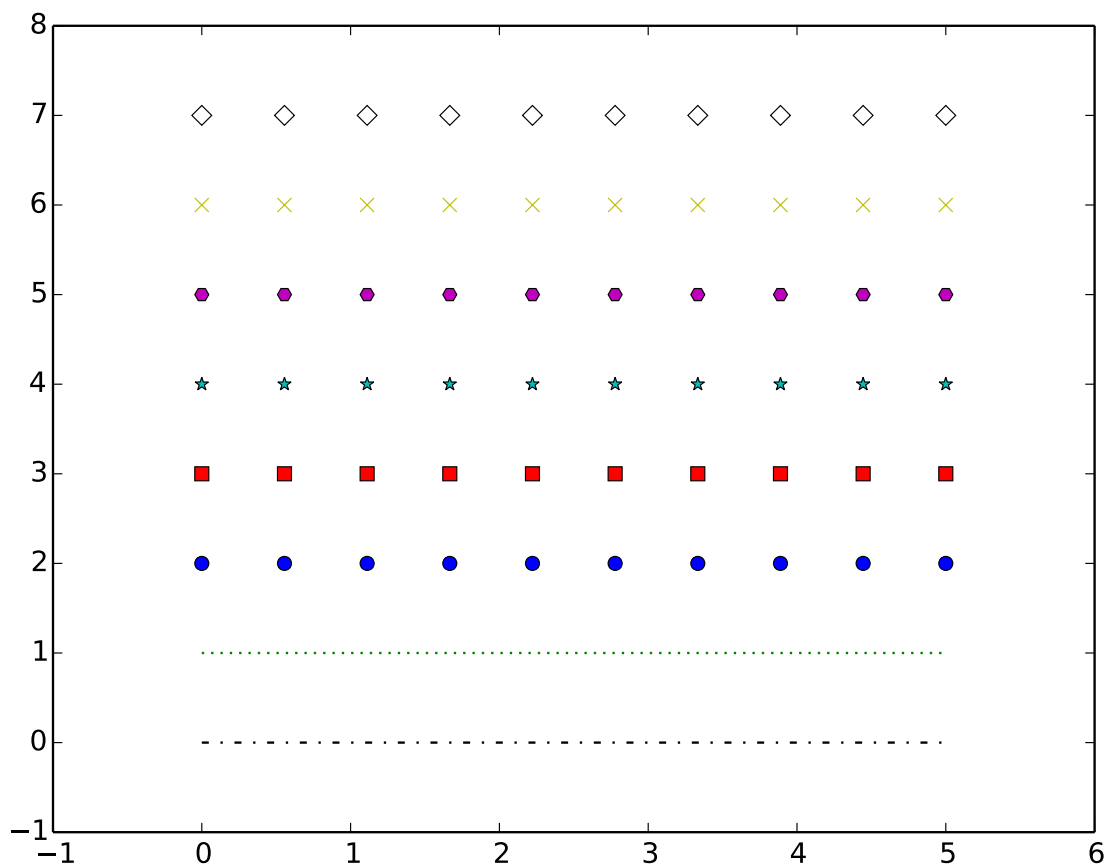
Figure 8.3: plot of varying linestyles.

The following displays how `linestyle` can be implemented. It is displayed in Figure 8.3.

```python
x = np.linspace(0, 5, 10)
y = np.ones(10)
ls = np.array(['-.', ':', 'o', 's', '*', 'H', 'x', 'D'])

for i in xrange(8):
    plt.plot(x, i*y, colors[i]+ls[i])

plt.axis([-1, 6, -1, 8])
plt.show()
```

linestyle.py

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 8.4.

```python
for i in xrange(8):
    plt.plot(x, i*y, colors[i]+ls[i])

plt.title("My Plot of Varying Linestyles", fontsize = 20, color = "gold")
plt.xlabel("x-axis", fontsize = 10, color = "darkcyan")
plt.ylabel("y-axis", fontsize = 10, color = "darkcyan")

plt.axis([-1, 6, -1, 8])
plt.show()
```

text.py

See `http://matplotlib.org` for Matplotlib documentation.

8.0.1. Jupyter Notebooks

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-J
ipynb

8.0.2. Reading and Writing

https://github.com/mathinmse/mathinmse.github.io/blob/master/Lecture-10B-Reading-and-Writin
ipynb
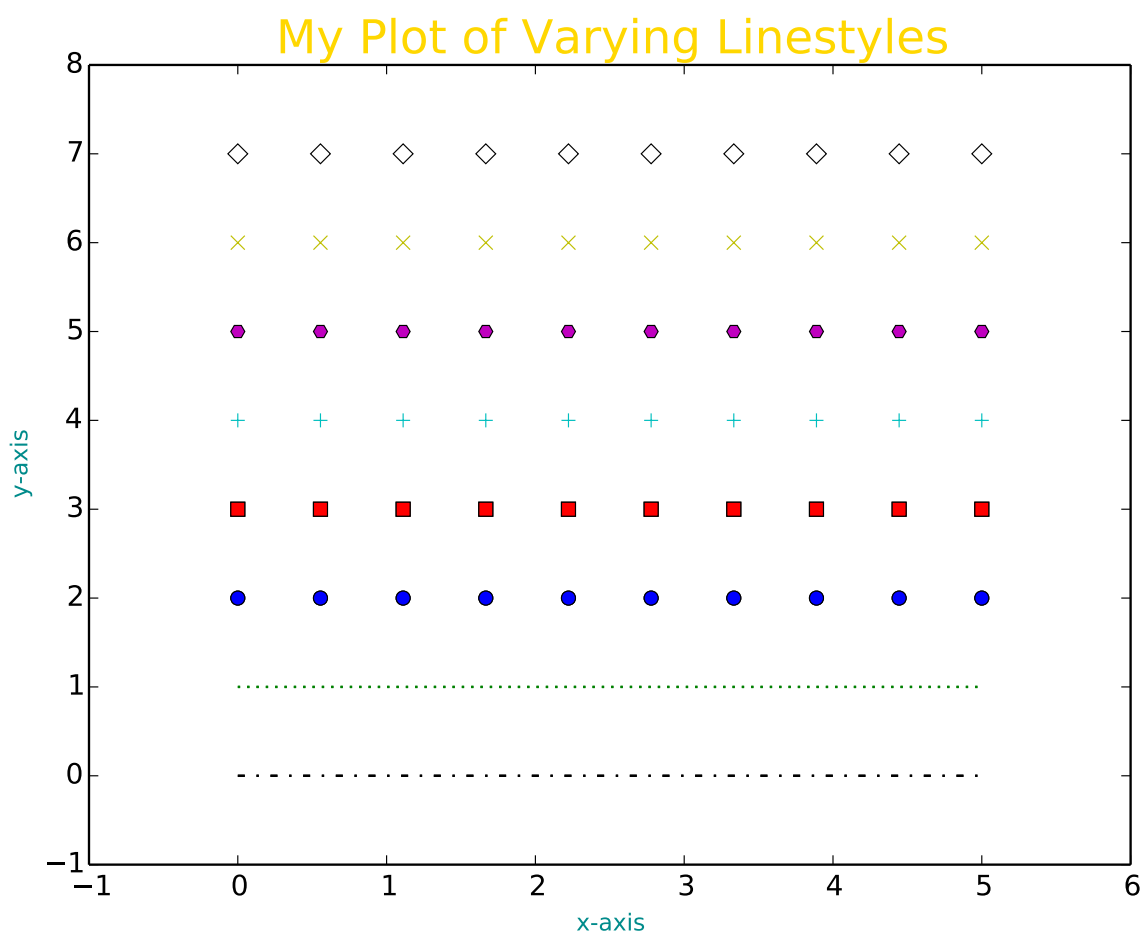
Figure 8.4: plot of varying linestyles using text labels.

8.0.3. Python Crash Course

```
https://github.com/rpmuller/PythonCrashCourse
```

8.0.4. Gurobi

Gurobi Log Tools

9. Simplex Method

> **Definition 9.1: Standard Form**
>
> *A linear program is in* standard form *if is written as*
>
> $$\max \ c^\top x$$
> $$s.t. \ Ax = b$$
> $$x \geq 0.$$

> **Definition 9.2: 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 9.3: 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 9.4: 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 9.5

Consider the problem

$$\max \quad Z = -5X - 7Y$$
$$\text{s.t.} \quad X + 3Y \geq 6$$
$$\qquad 5X + 2Y \geq 10$$
$$\qquad Y \leq 4$$
$$\qquad X, Y \geq 0$$
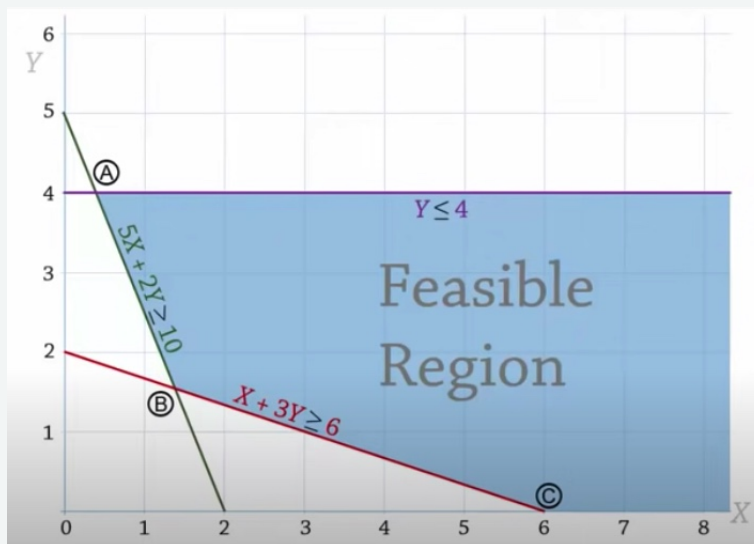
We begin by converting this problem to standard form.

$$\max \quad Z = -5X - 7Y + 0s_1 + 0s_2 + 0s_3$$
$$\text{s.t.} \quad X + 3Y - s_1 = 6$$
$$\qquad 5X + 2Y - s_2 = 10$$
$$\qquad Y + s_3 = 4$$
$$\qquad X, Y, s_1, s_2, s_3 \geq 0$$

Thus, we can write this problem in matrix form with

$$\max \begin{bmatrix} -5 \\ -7 \\ 0 \\ 0 \\ 0 \end{bmatrix}^{\top} \begin{bmatrix} X \\ Y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{9.1}$$

$$\begin{bmatrix} 1 & 3 & -1 & 0 & 0 \\ 5 & 2 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 10 \\ 4 \end{bmatrix} \tag{9.2}$$

$$(X, Y, s_1, s_2, s_3) \geq 0 \tag{9.3}$$

Definition 9.6: Basic feasible solution

Any basic solution in which all the variables are non-negative is a basic feasible solution.

Theorem 9.7: 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 9.8: Representation

Consider an LP in standard form, having bfs $b_1, \ldots, b_k$. 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 9.9: 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 unboundeness.

- 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 $i = 1, \ldots, k$, 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. ♠

10. 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. [6] Consider the following linear programming problem:

$$\text{maximize } 4x_1 + x_2 + 3x_3$$
$$\text{subject to} \quad x_1 + 4x_2 \leq 2$$
$$3x_1 - x_2 + x_3 \qquad\qquad \leq 4$$
$$x_1, x_2, x_3 \qquad\qquad \geq 0$$

Let σ^* 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 σ^* ?

In fact, there is. Let's multiply each of the constraints in our LP by a new non-negative scalar value y_i :

$$\text{maximize } 4x_1 + x_2 + 3x_3$$
$$\text{subject to } y_1 (x_1 + 4x_2 \quad \leq 2y_1$$
$$y_2 (3x_1 - x_2 + x_3) \leq 4y_2$$
$$x_1, x_2, x_3 \geq 0$$

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 $\left(x_1^*, x_2^*, x_3^*\right)$ for the original LP, we obtain the following upper bound on σ^* :

$$\sigma^* = 4x_1^* + x_2^* + 3x_3^* \leq 2y_1 + 4y_2.$$

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.

$$
\begin{aligned}
\text{minimize} \quad & 2y_1 + 4y_2 \\
\text{subject to} \quad & y_1 + 3y_2 \geq 4 \\
& 4y_1 - y_2 \geq 1 \\
& y_2 \qquad\quad \geq 3 \\
& y_1, y_2 \qquad \geq 0
\end{aligned}
$$

"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.

10.1 The Dual of Linear Program

Suppose that we have the following linear program in maximization standard form:

$$
\begin{aligned}
\text{maxim}&\text{ize} \quad x_1 + 2x_2 + x_3 + x_4 \\
\text{subject to} \quad & \\
& x_1 + 2x_2 + x_3 \leq 2 \\
& x_2 + x_4 \leq 1 \\
& x_1 + 2x_3 \leq 1 \\
& x_1 \geq 0 \\
& x_2 \geq 0 \\
& x_3 \geq 0
\end{aligned}
$$

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.

$$
\begin{aligned}
\text{maximize} \quad & c_1 x_1 + \ldots c_n x_n \\
\text{subject to} \quad & \\
& a_{1,1} x_1 + \ldots + a_{1,n} x_n \leq b_1 \\
& \vdots \\
& a_{m,1} x_1 + \ldots + a_{m,n} x_n \leq b_m \\
& x_1 \geq 0 \\
& \vdots \\
& x_n \geq 0
\end{aligned}
$$

For every choice of non-negative scaling factors $y_1, \ldots, y_m$, we can derive the inequality

$$
\begin{aligned}
y_1 \cdot (a_{1,1} x_1 + \ldots + a_{1,n} x_n) & \\
+ \cdots & \\
+ y_n \cdot (a_{m,1} x_1 + \ldots + a_{m,n} x_n) & \\
\leq y_1 b_1 + \cdots y_m b_m &
\end{aligned}
$$

which is true for every feasible solution $(x_1, \ldots, x_n)$ to the linear program (2). We can rewrite the inequality as

$$
\begin{aligned}
(a_{1,1} y_1 + \cdots a_{m,1} y_m) \cdot x_1 & \\
+ \cdots & \\
+ (a_{1,n} y_1 \cdots a_{m,n} y_m) \cdot x_n & \\
\leq y_1 b_1 + \cdots y_m b_m &
\end{aligned}
$$

So we get that a certain linear function of the x_i is always at most a certain value, for every feasible $(x_1, \ldots, x_n)$. 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 $(x_1, \ldots, x_n)$. We can achieve this if we choose the y_i such that

$$c_1 \le a_{1,1}y_1 + \cdots a_{m,1}y_m$$
$$\vdots$$
$$c_n \le a_{1,n}y_1 \cdots a_{m,n}y_m$$

Now we see that for every non-negative $(y_1, \ldots, y_m)$ that satisfies (3), and for every $(x_1, \ldots, x_n)$ that is feasible for (2),

$$c_1 x_1 + \ldots c_n x_n$$
$$\le (a_{1,1}y_1 + \cdots a_{m,1}y_m) \cdot x_1$$
$$+ \cdots$$
$$+ (a_{1,n}y_1 \cdots a_{m,n}y_m) \cdot x_n$$
$$\le y_1 b_1 + \cdots y_m b_m$$

Clearly, we want to find the non-negative values $y_1, \ldots, y_m$ such that the above upper bound is as strong as possible, that is we want to

$$
\begin{aligned}
\text{minimize} \quad & b_1 y_1 + \cdots b_m y_m \\
\text{subject to} \quad & \\
& a_{1,1}y_1 + \ldots + a_{m,1}y_m \ge c_1 \\
& \vdots \\
& a_{n,1}y_1 + \ldots + a_{m,n}y_m \ge c_n \\
& y_1 \ge 0 \\
& \vdots \\
& y_m \ge 0
\end{aligned}
$$

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

$$
\begin{aligned}
\text{maximize} \quad & \mathbf{c}^T \mathbf{x} \\
\text{subject to} \quad & \\
& A\mathbf{x} \le \mathbf{b} \\
& \mathbf{x} \ge 0
\end{aligned}
$$

is a linear program in maximization standard form, then its dual is the minimization linear program

$$
\begin{aligned}
\text{minimize} \quad & \mathbf{b}^T \mathbf{y} \\
\text{subject to} \quad & \\
& A^T \mathbf{y} \ge \mathbf{c} \\
& \mathbf{y} \ge 0
\end{aligned}
$$

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

$$\underset{l}{\text{minimize}} \quad \mathbf{c}^T \mathbf{y}$$
$$\text{subject to}$$
$$A\mathbf{y} \geq \mathbf{b}$$
$$\mathbf{y} \geq 0$$

in an equivalent way as

$$\underset{\text{maximize}}{\text{mubject to}} \quad -\mathbf{c}^T \mathbf{y}$$
$$-A\mathbf{y} \leq -\mathbf{b}$$
$$\mathbf{y} \geq 0$$

If we compute the dual of the above program we get

$$\underset{\text{minimize}}{\text{mubject to}} \quad -\mathbf{b}^T \mathbf{z}$$
$$-A^T \mathbf{z} \geq -\mathbf{c}$$
$$\mathbf{z} \geq 0$$

that is,

$$\text{maximize} \quad \mathbf{b}^T \mathbf{z}$$
$$\text{subject to}$$
$$A^T \mathbf{z} \leq \mathbf{c}$$
$$\mathbf{y} \geq 0$$

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.

We will return to the Strong Duality Theorem, and discuss its proof, later in the course.

11. Sensitivity Analysis

12. Multi-Objective Optimization

12.1 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.

$$P = \{(x,y) \in \mathbb{R}^2 :$$
$$80x + 20y \leq 960$$
$$12x + 10y \leq 200$$
$$x, y \geq 0\}$$
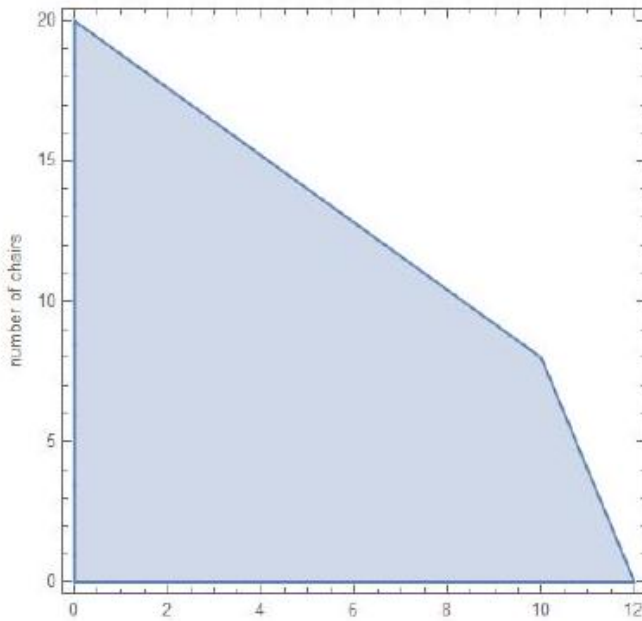
Suppose that each table will sells for $7000 while a chair goes for $1500. To increase profit we want to maximize

$$F(x,y) = 8000x + 2000y$$

over P. Having taken a linear programming class, the manager knows his way around these problems and begins the simplex method:

Maximize $8000x + 2000y$

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

Maximize $4x + y$

s.t. $4x + y + s_1 = 48$

$$\frac{f}{2000} \qquad \begin{aligned} 6x + 5y + s_2 &= 100 \\ x, y &\geq 0 \end{aligned}$$

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 this from the tableau? From the original formulation?

Is the manager's solution really the best?
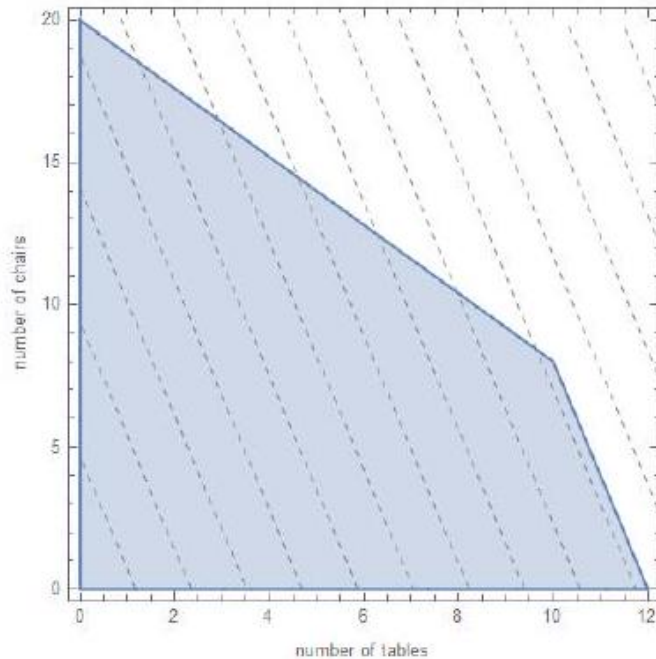
number of tables Having fired the prior manager for producing no chairs, a new and more competent manager is hired. This one knows that Dalbergia stevensonii (the tree which produces their preferred rosewood) is a threatened species and decides that she doesn't want to waste any more rosewood than is 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:

$$\text{Minimize } 10x + 2y$$
$$\text{s.t.} \quad 80x + 20y = 960$$
$$x \in [10, 12]$$
$$y \in [0, 8].$$

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:

$$\text{Minimize } 10x + 2y$$
$$\text{s.t.} \quad 8000x + 2000y \geq (\alpha)96000$$
$$80x + 20y \leq 960$$
$$12x + 10y \leq 200$$
$$x, y \geq 0$$

where $\alpha = 0.7$ This new constraint limits us to solutions which offered at least 70% of maximum possible revenue.
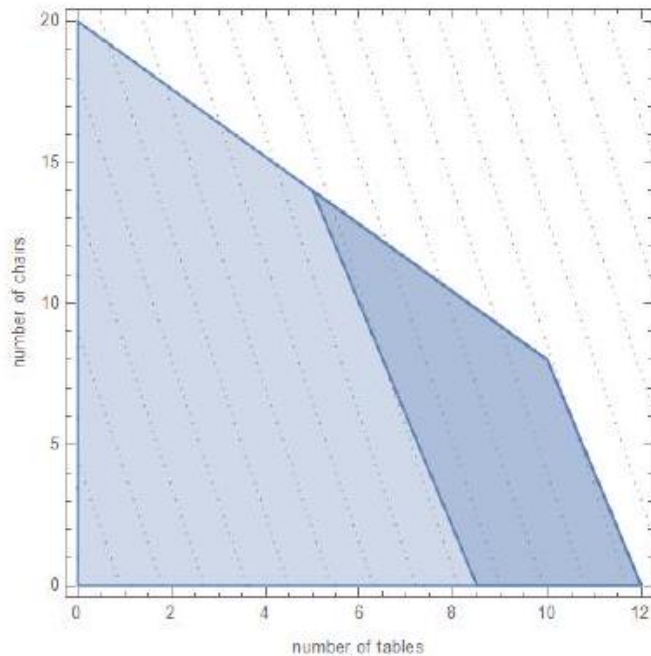
The strategy is the called the Benchmark or Rollover method because we choose a benchmark for one of our objectives (revenue in this case), roll that benchmark into the constraints, and optimize for the second objective (waste).

Notice that if we set α 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 $f_i : P \to \mathbb{R}$ 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

$\to f_j(\bar{\mathbf{x}}) \geq f_j(\mathbf{x})$ 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?
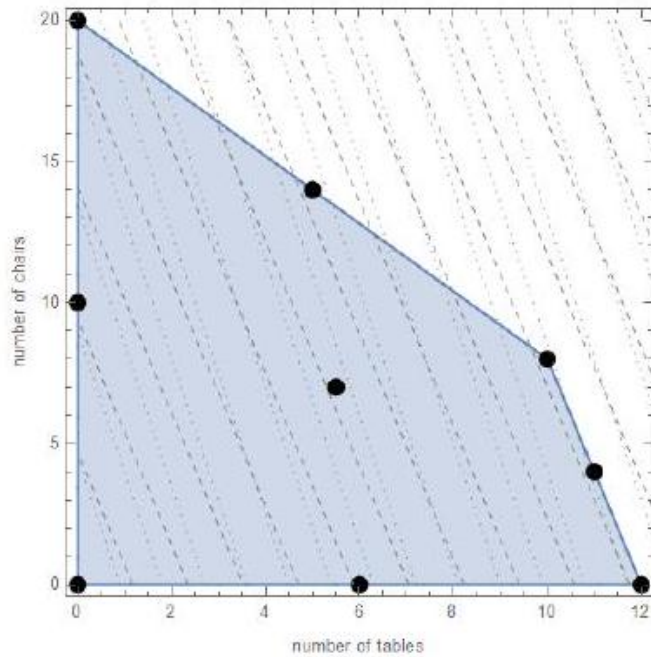
number of tables The rollover method is generalized in Goal Programming

By varying α, it is possible to generate many distinct efficient solutions.

However, this method can generate inefficient solutions if the underlying model is poorly constructed.
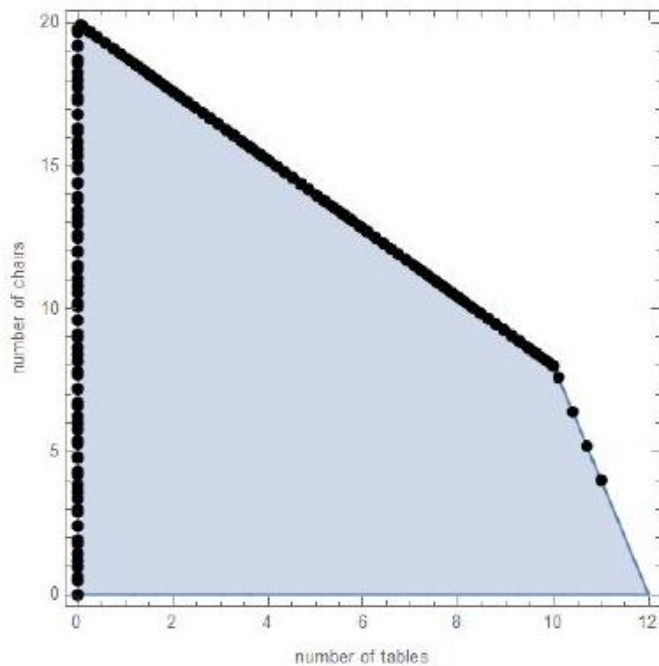
number of tables It is more common to see a Pareto frontier plotted with respect to its objectives.
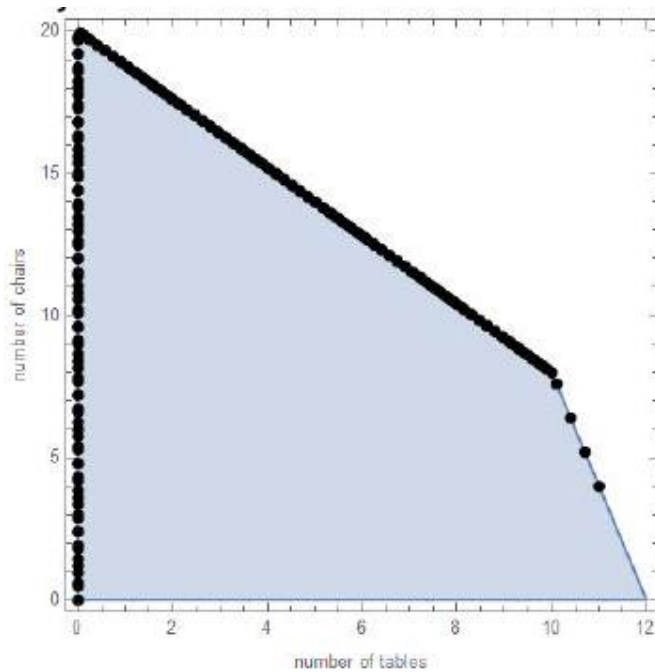
number of table
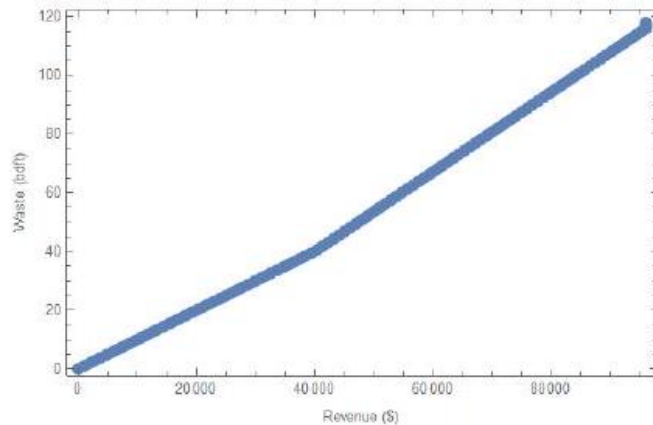
One of the owners of our manufactory decides to explore possible planning himself; he implements the multi-objective method that he remembers, Scalarization by picking some arbitrary constant $\lambda \in [0, 1]$ and combining his two objectives like so:

$$\begin{aligned}
\text{Minimize} \quad & \lambda(8000x + 2000y) + (1 - \lambda)(10x + 2y) \\
\text{s.t.} \quad & 80x + 20y \leq 960 \\
& 12x + 10y \leq 200 \\
& x, y \geq 0
\end{aligned}$$

What is the benefit of this method?

Where does it fall short?

12.2 What points will the Scalarization method find if we vary λ?
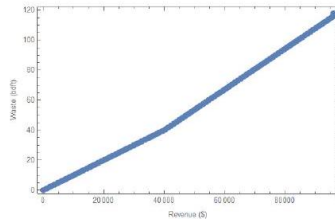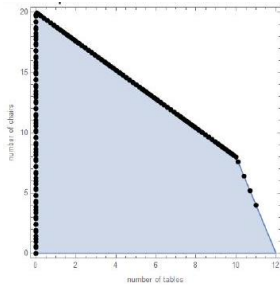
These are all nice ideas, but the problem presented above is neither difficult nor practical.

What are some areas that a Pareto frontier would be actually useful?

12.3 Political Redistricting [3]
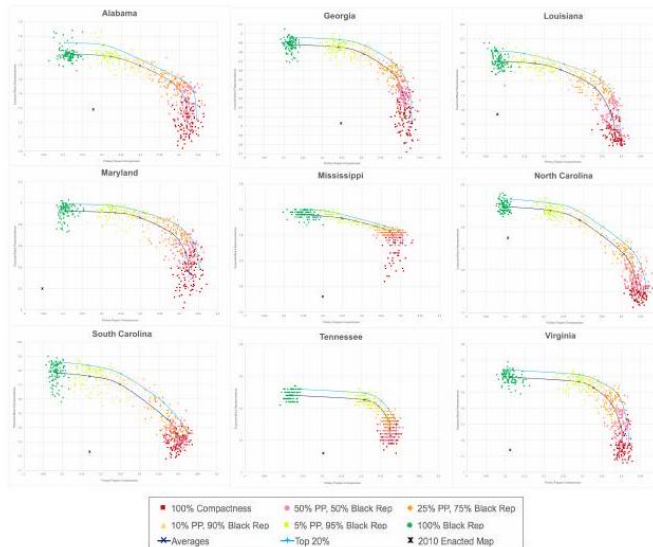
12.4 Portfolio Optimization [5]

12.5 Simulated Portfolio Optimization based on Efficient Frontier
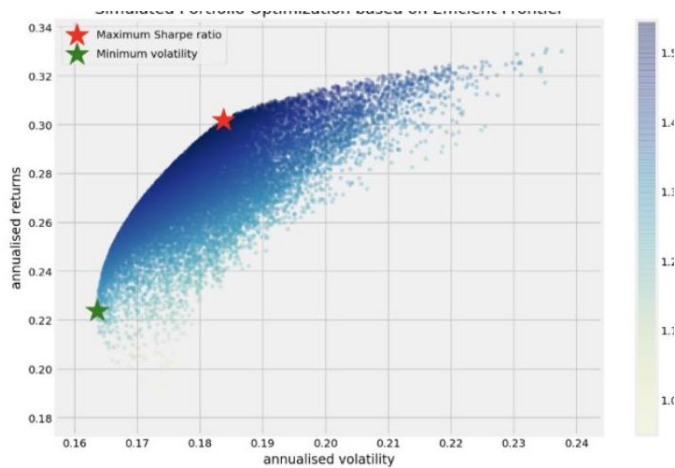
12.6 Aircraft Design [1]
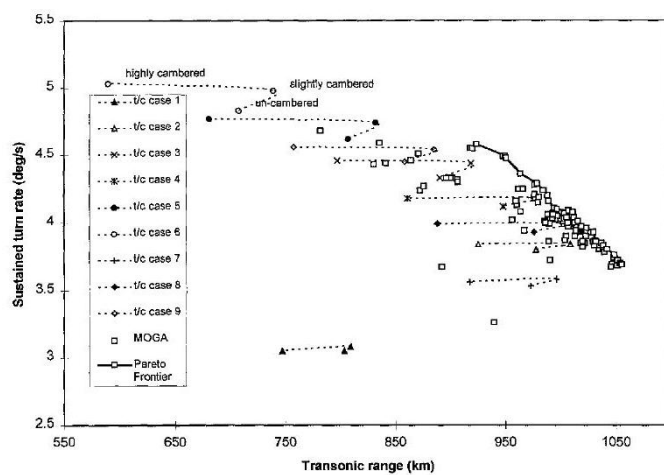
12.7 Vehicle Dynamics [4]
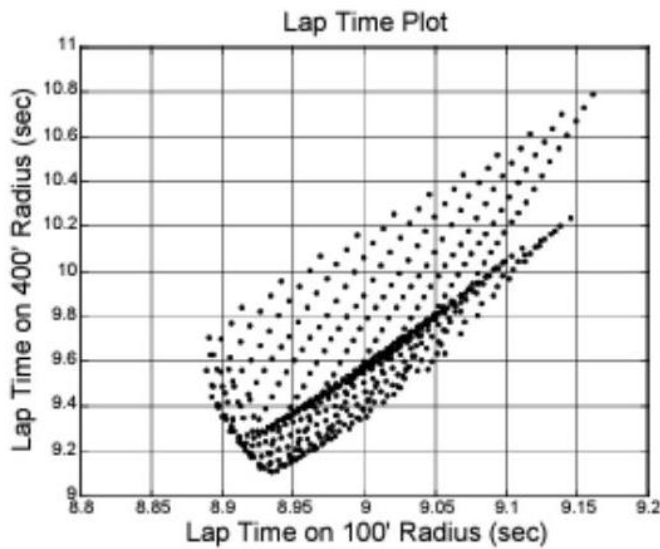
Figure 7: Grid Search Results in the Performance Space

12.8 Sustainable Constriction [2]
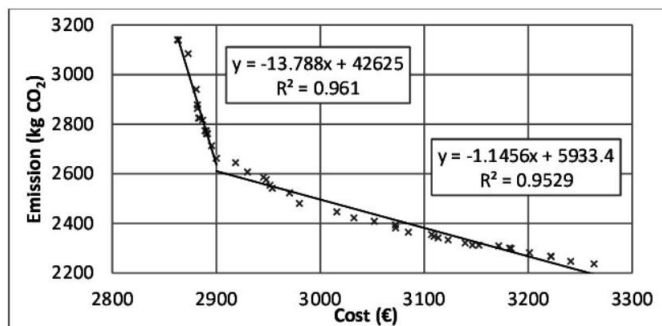
12.9 References

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 $00a59$ d$200c2c74$ 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_optimiza 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 II

Discrete Algorithms

13. 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 tth person is the one we should marry. Note that the probability that the tth person is not the best candidate is $\frac{t-1}{t}$ and the probability that they are is $\frac{1}{t}$. If the tth 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 tth 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,

$$V(t-1) = \frac{t-1}{t}\max\{0, V(t)\} + \frac{1}{t}\max\left\{\frac{t}{N}, V(t)\right\} = \max\left\{\frac{t-1}{t}V(t) + \frac{1}{N}, V(t)\right\}. \tag{13.1}$$

Notice that (13.1) 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, $P(\text{t is best overall}|t \text{ is best out of the first } t) = \frac{t}{N}$ is strictly

101

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 (13.1) 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 $t = N, N-1, \ldots, 1$.

If $N = 4$, we have

$$V(4) = 0,$$

$$V(3) = \max\left\{\frac{3}{4}V(4) + \frac{1}{4}, 0\right\} = .25,$$

$$V(2) = \max\left\{\frac{2}{3}V(3) + \frac{1}{4}, .25\right\} = .4166,$$

$$V(1) = \max\left\{\frac{1}{4}, .4166\right\} = .4166.$$

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 13.1: W

ite a function that accepts a number of candidates N. Calculate the expected values of choosing candidate t for $t = 0, 1, \ldots, N-1$.
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 13.2: W

ite a function that takes in an integer M and runs your function from Problem 13 for each $N = 3, 4, \ldots, M$. 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.

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

$$\mathbf{c} = \begin{bmatrix} c_0 & c_1 & \cdots & c_T \end{bmatrix}^\mathsf{T},$$

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

$$\sum_{t=0}^{T} \beta^t u(c_t).$$
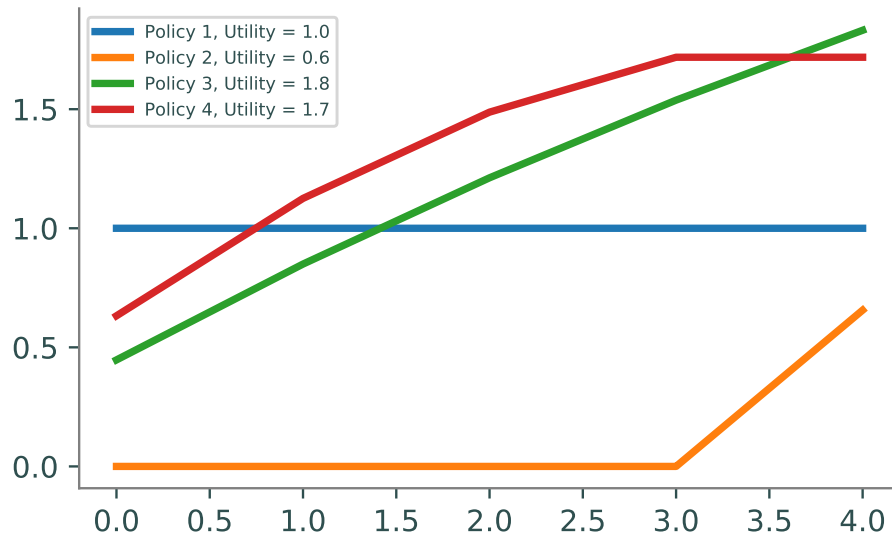
Figure 13.1: Plots for various policies with $u(x) = \sqrt{x}$ **and** $\beta = 0.9$**. Policy 1 eats all of the cake in the first step while policy 2 eats all of the cake in the last step. Their difference in utility demonstrate the effect of the discount factor on waiting to eat. Policy 3 eats the 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.

$$\max_{\mathbf{c}} \sum_{t=0}^{T} \beta^t u(c_t) \tag{13.2}$$

$$\text{subject to } \sum_{t=0}^{T} c_t = W$$

$$c_t \geq 0.$$

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.

$$V(a,b,W) = \max_{\mathbf{c}} \sum_{t=a}^{b} \beta^t u(c_t)$$

$$\text{subject to } \sum_{t=a}^{b} c_t = W$$

$$c_t \geq 0.$$

$V(0,T,W)$ gives how much utility we gain in T days and is the same as Equation 13.2.

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.

$$V(t,T,W_t) = \max_{W_{t+1}} \left(u(W_t - W_{t+1}) + \beta V(t,T-1,W_{t+1}) \right) \tag{13.3}$$

where $u(W_t - W_{t+1})$ is the value gained from eating $W_t - W_{t+1}$ cake at time t.

Let $\mathbf{w} = \begin{bmatrix} 0 & \frac{1}{N} & \cdots & \frac{N-1}{N} & 1 \end{bmatrix}^{\mathsf{T}}$. 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 $w = [0,.25,.5,.75,1]^{\mathsf{T}}$, 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 13.3

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 β. 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 13.3 is a new value function problem with $a = t, b = T - 1$, and $W = W_{t+1}$, making 13.3 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 (13.3),

$$A_{iT} = V(0, 0, w_i) = \max_{w_j} \left(u(w_i - w_j) + \beta V(0, -1, w_j) \right) = u(w_i). \tag{13.4}$$

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 13.4: W

ite a function that accepts a stopping time T, a number of equal sized pieces N that divides the cake, a discount factor β, 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 (13.4) as follows,

$$A_{i(T-1)} = V(0,1,w_i) = \max_{w_j} \left(u(w_i - w_j) + \beta V(0,0,w_j) \right) = \max_{w_j} \left(u(w_i - w_j) + \beta A_{jT} \right). \tag{13.5}$$

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 $t = T-1$,

$$CV_{ij}^{T-1} = u(w_i - w_j) + \beta A_{jT}. \tag{13.6}$$

The largest entry in the ith 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 tth 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 3rd column of A. Remember that A's column index begins at 0, so the 3rd 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 ith row of CV^t. Generalizing (13.5) and (13.6) shows

$$CV_{ij}^t = u(w_i - w_j) + \beta A_{j(t+1)}. \qquad A_{it} = \max_j \left(CV_{ij}^t \right). \qquad (13.7)$$

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 13.2: 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 13.5: C

mplete your function from Problem 13 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 (13.7),

- 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,

$$P_{it} = w_i - w_j.$$

$$\text{where } j = \{ min\{j\} \mid CV_{ij}^t \geq CV_{ik}^t \, \forall k \in [0, 1, \ldots, N] \}$$

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 entries 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 $\mathbf{c} = \begin{bmatrix} .25 & .25 & .25 & .25 \end{bmatrix}$.

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 entires of A, we can see that the optimal policy does give the maximum value.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \sqrt{0.25} & \sqrt{0.25} & \sqrt{0.25} & \sqrt{0.25} \\ \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} & \sqrt{0.5} \\ \sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.5} + \beta\sqrt{0.25} & \sqrt{0.75} \\ \cancel{\sqrt{0.25} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} + \beta^3\sqrt{0.25}} & \sqrt{0.5} + \beta\sqrt{0.25} + \beta^2\sqrt{0.25} & \sqrt{0.5} + \beta\sqrt{0.5} & \sqrt{1} \end{bmatrix}$$

Problem 13.6: M

dify your function from Problem 13 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 13.7: W

ite 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 β, and the utility function u.*

14. Graph Algorithms

14.1 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.

14.2 Graphs

14.2.1. Drawing Graphs

Example 14.1

Here is a portion of a housing development from Missoula, Montana[a]. 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.

[a]Same Beebe. http://www.flickr.com/photos/sbeebe/2850476641/
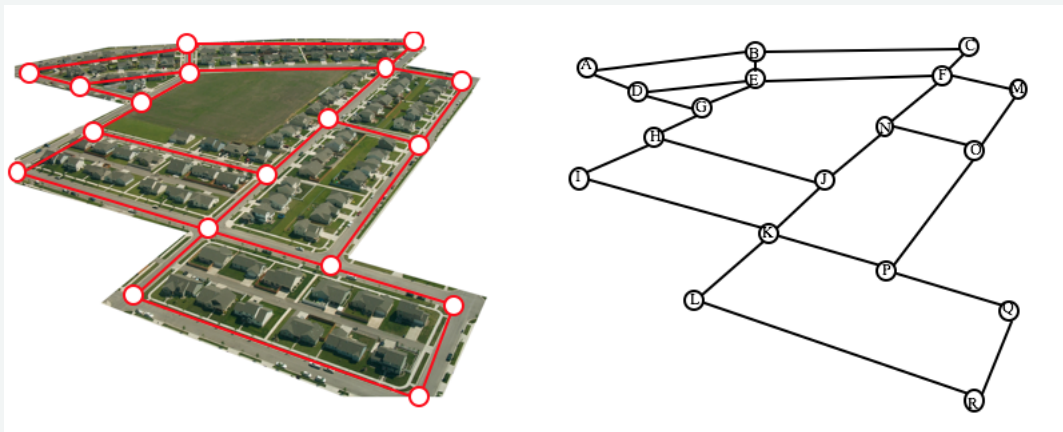
This type of simplified picture is called a **graph**.

Definition 14.2: 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.
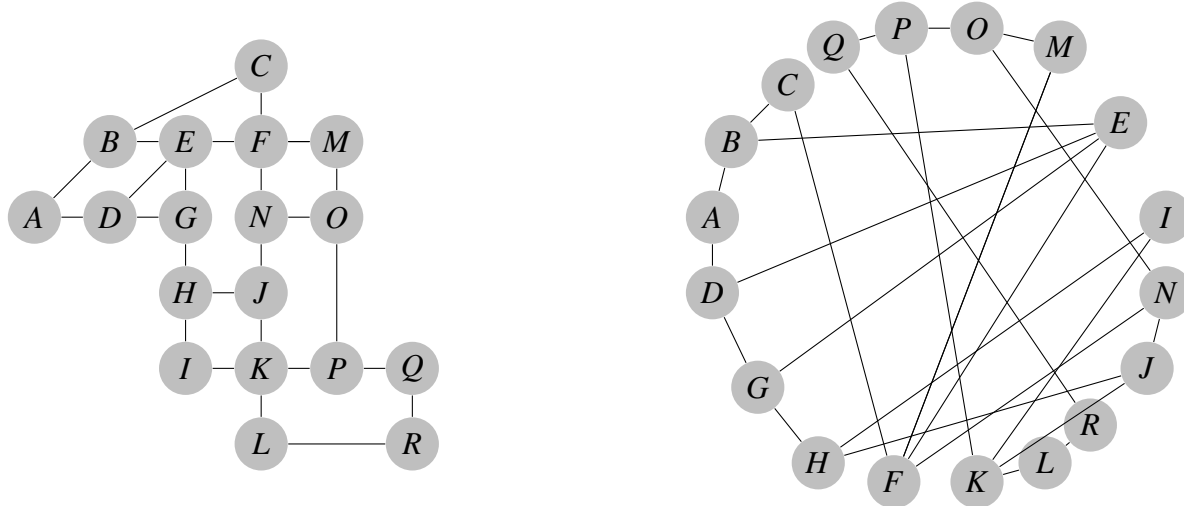
You probably already noticed that we are using the term graph differently than you may have used the term in the past to describe the graph of a mathematical function.

Example 14.3

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:
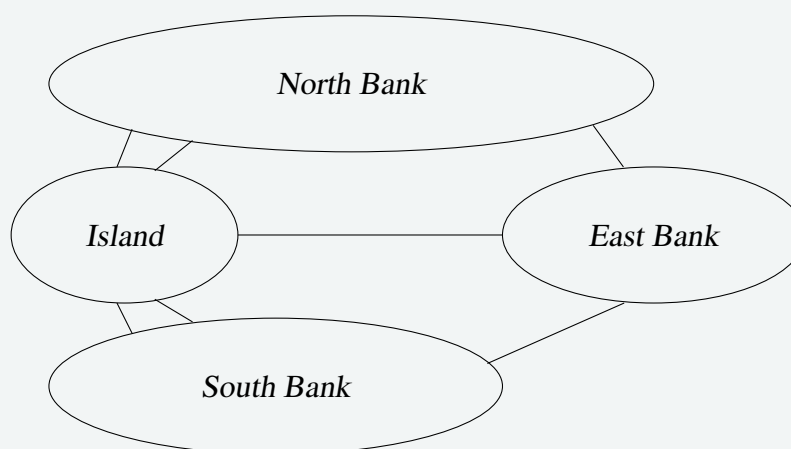
Since the size of each land mass it is not relevant to the question of bridge crossings, each can be shrunk down to a vertex representing the location:
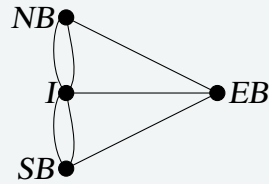
Notice that in this graph there are two edges connecting the north bank and island, corresponding to the two bridges in the original drawing. Depending upon the interpretation of edges and vertices appropriate to a scenario, it is entirely possible and reasonable to have more than one edge connecting 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.

14.3 Definitions

While we loosely defined some terminology earlier, we now will try to be more specific.

Definition 14.4: 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 14.5: 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.
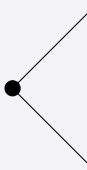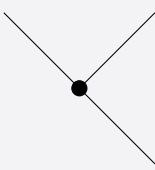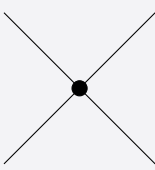
Definition 14.6: Loop

A loop is a special type of edge that connects a vertex to itself. Loops are not used much in street network graphs.
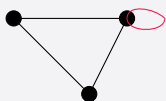
Definition 14.7: Degree of a vertex

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.

Degree 0	Degree 1	Degree 2	Degree 3	Degree 4
•	╱	⟨	⟨	⤬

Definition 14.8: 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.
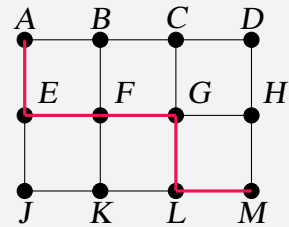
Definition 14.9: Circuit (a.k.a. cycle)

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.
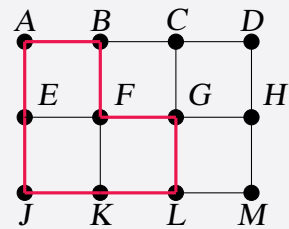
Definition 14.10: Connected

*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.*
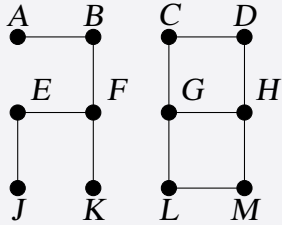
Definition 14.11: Weights

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 14.12

The graph below shows 5 cities. The weights on the edges represent the airfare for a one-way flight between the cities.

    a. *How many vertices and edges does the graph have?*

    b. *Is the graph connected?*

    c. *What is the degree of the vertex representing LA?*

    d. *If you fly from Seattle to Dallas to Atlanta, is that a path or a circuit?*

    e. *If you fly from LA to Chicago to Dallas to LA, is that a path or a circuit?*
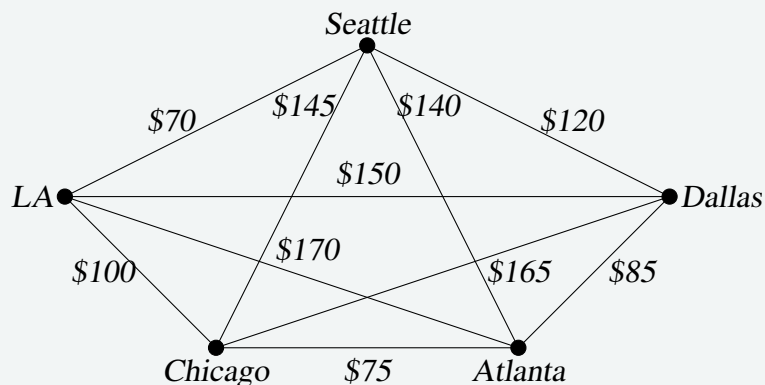
14.4 Shortest Path

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-stra) 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 14.13

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.
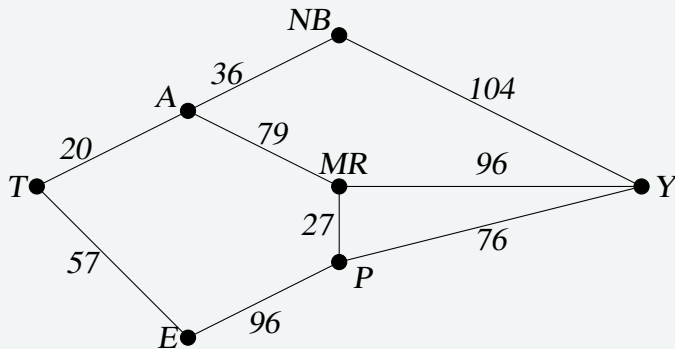
Step 1: Mark the ending vertex with a distance of zero. The distances will be recorded in [brackets] after the vertex name.
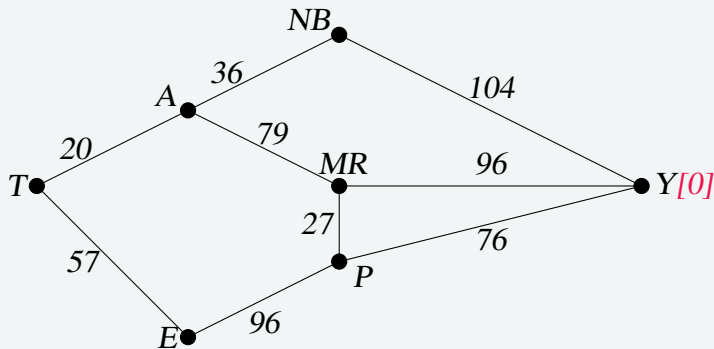
Step 2: For each vertex leading to Y, we calculate the distance to the end. For example, NB is a distance of 104 from the end, and MR is 96 from the end. Remember that distances in this case refer to the travel time in minutes.
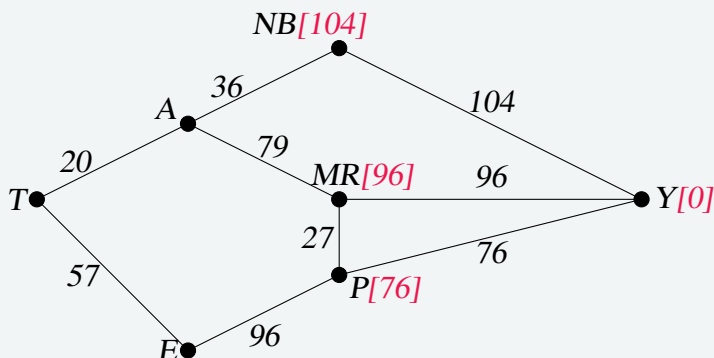
Step 3 & 4: We mark Y as visited, and mark the vertex with the smallest recorded distance as current. At this point, P will be designated current. Back to step 2.

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.
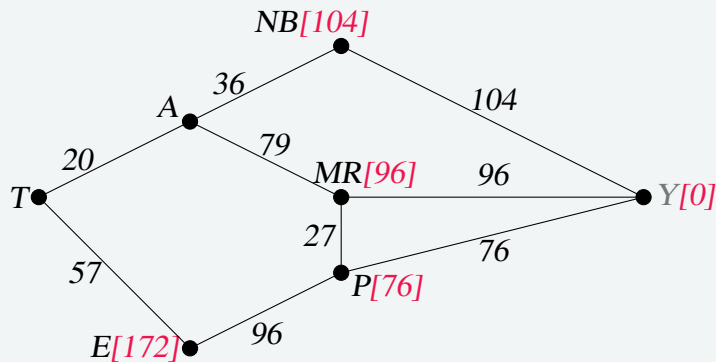
Step 3 & 4 (#2): We mark P as visited, and designate the vertex with the smallest recorded distance as current: MR. Back to step 2.

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.
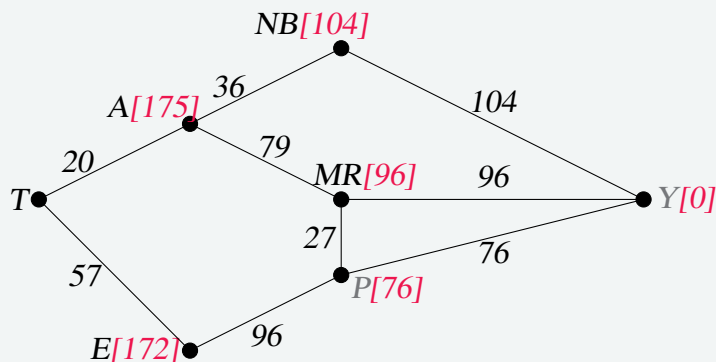
Step 3 & 4 (#3): We mark MR as visited, and designate the vertex with smallest recorded distance as current: NB. Back to step 2.

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.
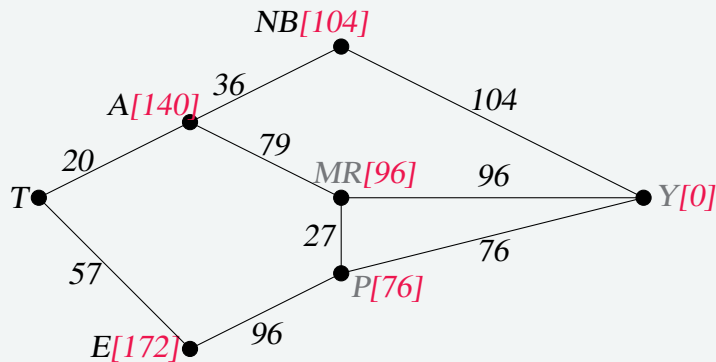
Step 3 & 4 (#4): We mark NB as visited, and designate A as current, since it now has the shortest distance.

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.
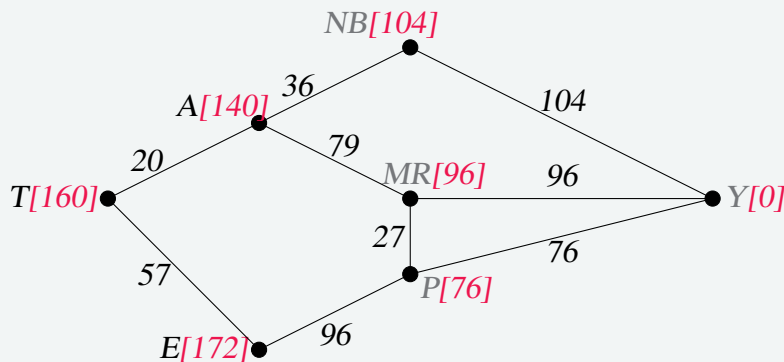
Step 3 & 4 (#5): We mark A as visited, and designate E as current.

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 Tacp,a 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[1]. 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.

---

[1] 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 14.14: Dijkstra's algorithm example

We would like to find a shortest path in the graph from node a to node g.
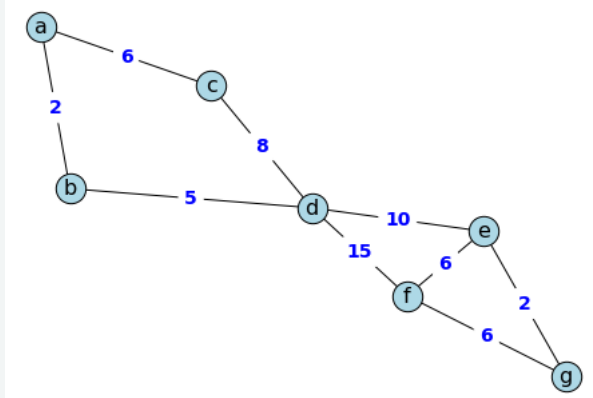
We will initialize our algorithm at node 'a'.

current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

Example 14.15: Test figure
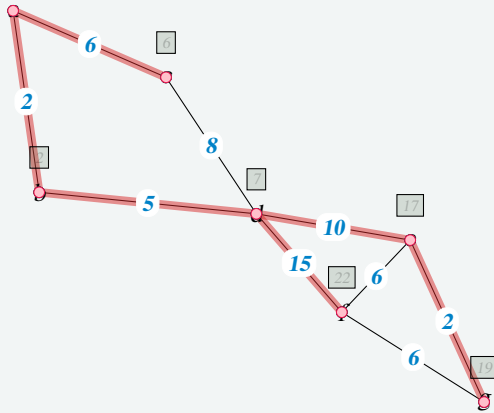
Example 14.16

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	Denver [19]	Dallas [25]	Chicago	Atlanta	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 (#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 14.17

Find the shortest path between vertices A and G in the graph below.
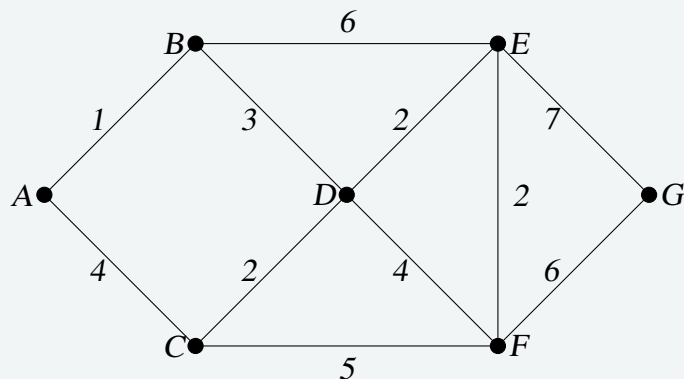
14.5 Spanning Trees

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.
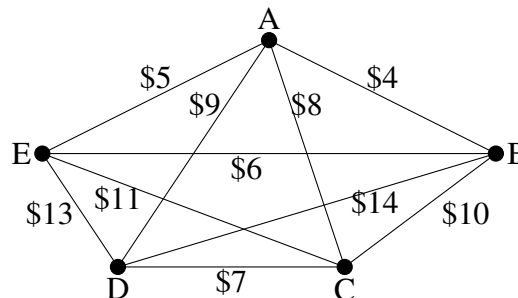
In this case, we don't need to find a circuit, or even a specific path; all we need to do is make sure we can make a call from any office to any other. In other words, we need to be sure there is a path from any vertex to any other vertex.

Definition 14.18: 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.
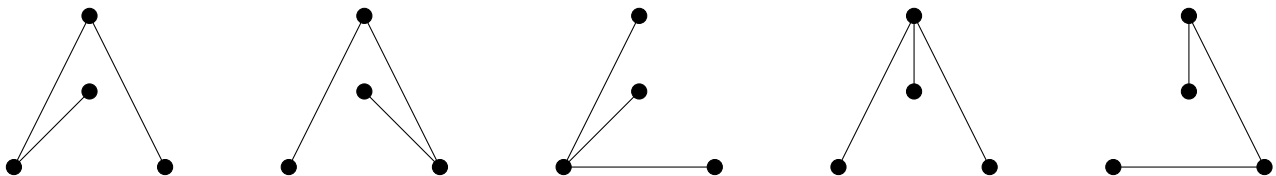
Usually we have a starting graph to work from, like in the phone example above. In this case, we form our spanning tree by finding a **subgraph** – a new graph formed using all the vertices but only some of the edges from the original graph. No edges will be created where they didn't already exist.

Of course, any random spanning tree isn't really what we want. We want the **minimum cost spanning tree (MCST)**.

Definition 14.19: 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 14.20

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*
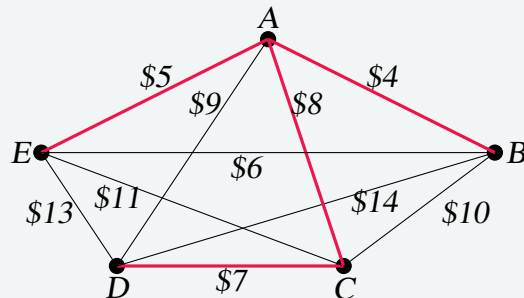
At this point we stop – every vertex is now connected, so we have formed a spanning tree with cost $24 thousand a year.

Remarkably, Kruskal's algorithm is both optimal and efficient; we are guaranteed to always produce the optimal MCST.

Example 14.21

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,
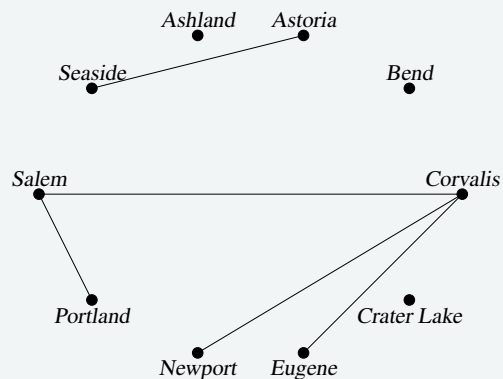
Newport to Salem	*reject*
Corvallis to Portland	*reject*
Eugene to Newport	*reject*
Portland to Astoria	*reject*
Ashland to Crater Lake	*108 miles*
Eugene to Portland	*reject*
Newport to Portland	*reject*
Newport to Seaside	*reject*
Salem to Seaside	*reject*
Bend to Eugene	*128 miles*
Bend to Salem	*reject*
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*
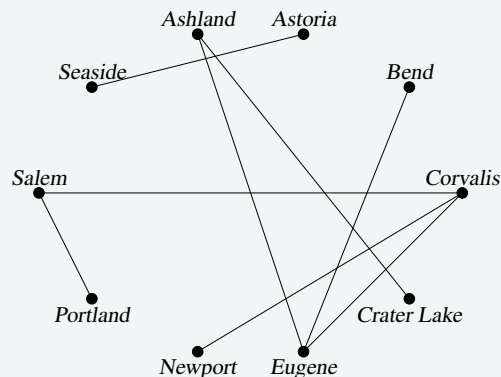
This connects the graph. The total length of cable to lay would be 695 miles.

Exercise 14.22

Find a minimum cost spanning tree on the graph below using Kruskal's algorithm.
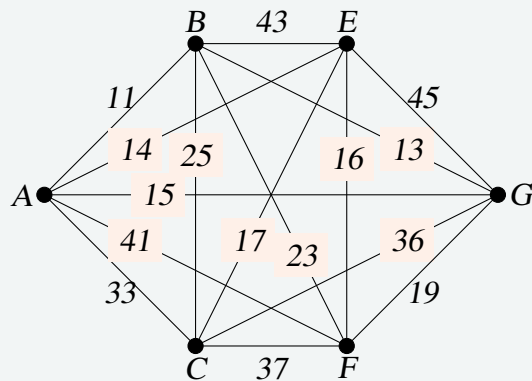
14.6 Exercise Answers

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: ABEGFCDFEDBCA

4.

   This graph can be eulerized by duplicating the edge BC, as shown. One possible Euler circuit on the eulerized graph is ACDBCBA.
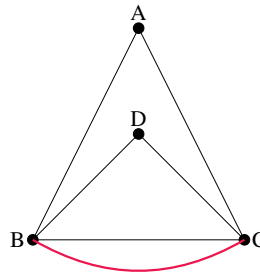
5. At each step, we look for the nearest location we haven't already visited.
   From B the nearest computer is E with time 24.
   From E, the nearest computer is D with time 11.
   From D the nearest is A with time 12.
   From A the nearest is C with time 34.
   From C, the only computer we haven't 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 RNNA 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
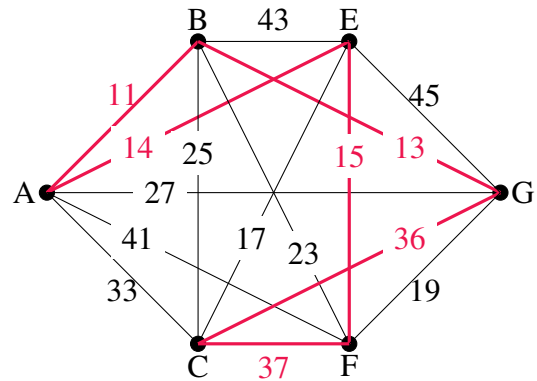
7.

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.
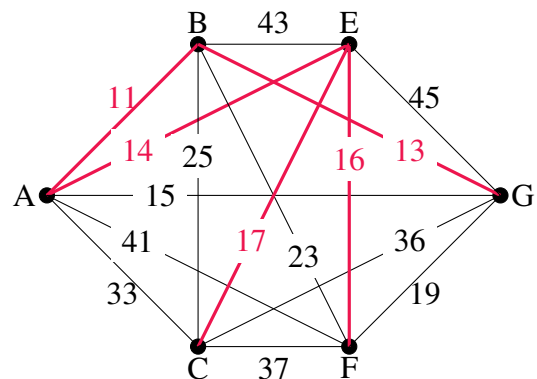
14.7 Additional Exercises

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.
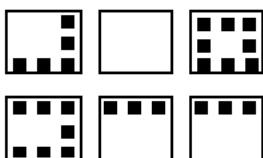
2. Suppose that a town has 7 bridges as pictured below. Create a graph that could be used to determine if there is a path that crosses all bridges once.
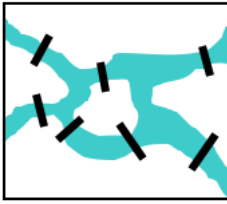
3. The table below shows approximate driving times (in minutes, without traffic) between five cities in the Dallas area. Create a weighted graph representing this data.

	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[2]. 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.
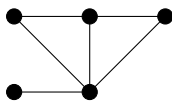
6. Find the degree of each vertex in the graph below.

---
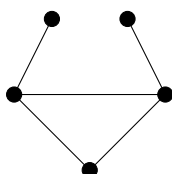
[2]Cheapest fares found when retrieved Sept. 1, 2009 for travel Sept. 22, 2009

7. Which of these graphs are connected?
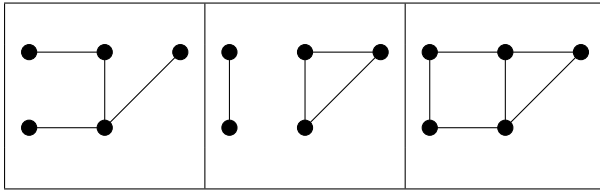
8. Which of these graphs are connected?
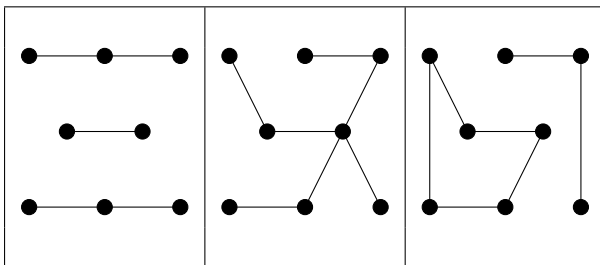
9. Travel times by rail for a segment of the Eurail system is shown below with travel times in hours and minutes. Find path with shortest travel time from Bern to Berlin by applying Dijkstra's algorithm.
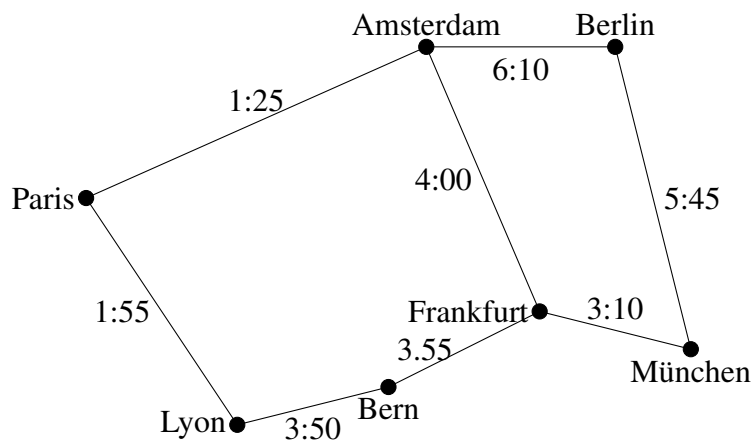
10. Using the graph from the previous problem, find the path with shortest travel time from Paris to München.

11. Does each of these graphs have an Euler circuit? If so, find it.
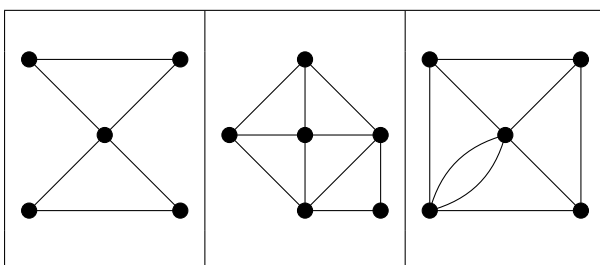
12. Does each of these graphs have an Euler circuit? If so, find it.
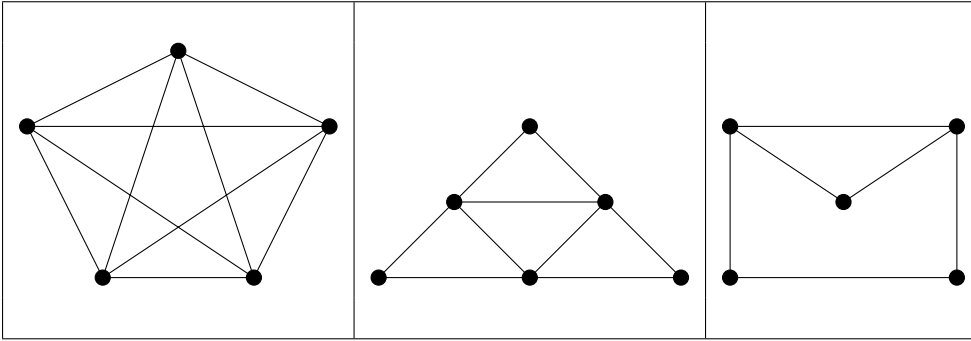
13. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.
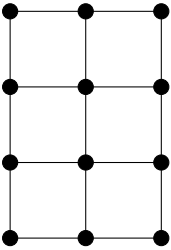
14. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.
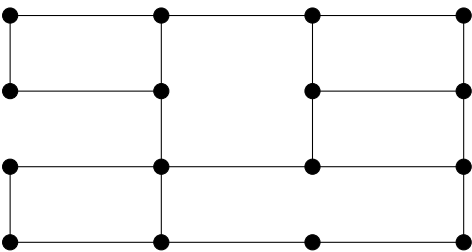
15. The maintenance staff at an amusement park need to patrol the major walkways, shown in the graph below, collecting litter. Find an efficient patrol route by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.
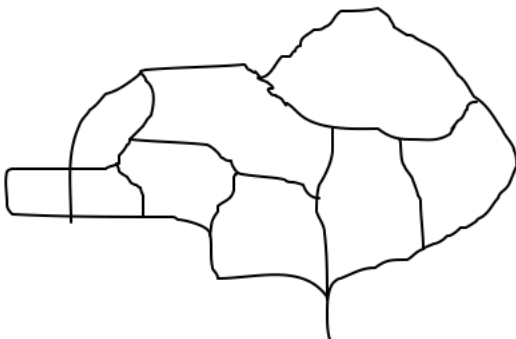
16. After a storm, the city crew inspects for trees or brush blocking the road. Find an efficient route for the neighborhood below by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.
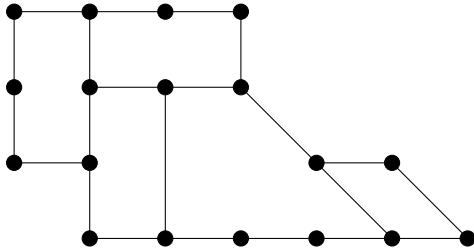
17. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.
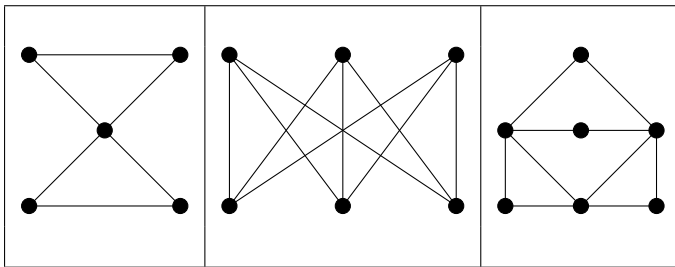
18. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.
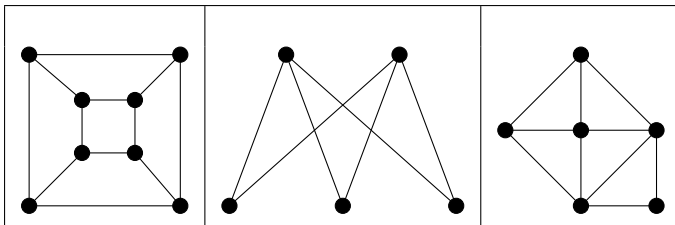
19. A company needs to deliver product to each of their 5 stores around the Dallas, TX area. Driving distances between the stores are shown below. Find a route for the driver to follow, returning to the distribution center in Fort Worth:

   (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:

   (a) Using Nearest Neighbor starting in Seattle

   (b) Using Repeated Nearest Neighbor

   (c) 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:

   (a) Using Nearest Neighbor starting at building A

   (b) Using Repeated Nearest Neighbor

   (c) Using Sorted Edges
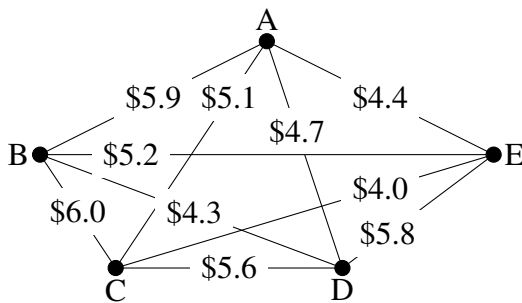
22. A tourist wants to visit 7 cities in Israel. Driving distances, in kilometers, between the cities are shown below[3]. Find a route for the person to follow, returning to the starting city:

   (a) Using Nearest Neighbor starting in Jerusalem

   (b) Using Repeated Nearest Neighbor

   (c) Using Sorted Edges

	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

[3]From http://www.ddtravel-acc.com/Israel-cities-distance.htm

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?
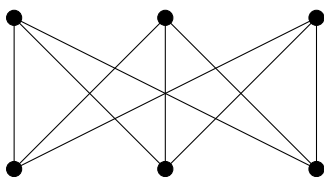
29. Eulerize this graph in the most efficient way possible, considering the weights of the edges.
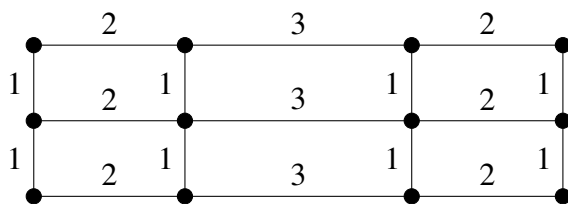
30. Eulerize this graph in the most efficient way possible, considering the weights of the edges.
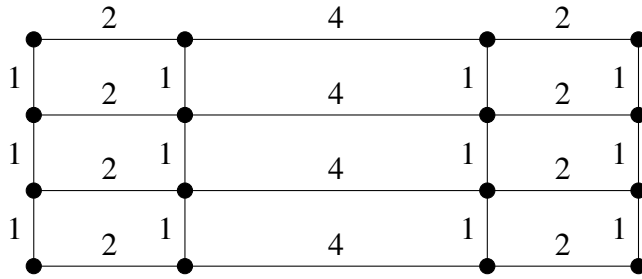
31. Eulerize this graph in the most efficient way possible, considering the weights of the edges.
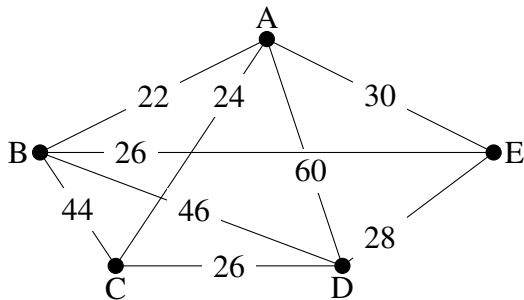
32. Eulerize this graph in the most efficient way possible, considering the weights of the edges.
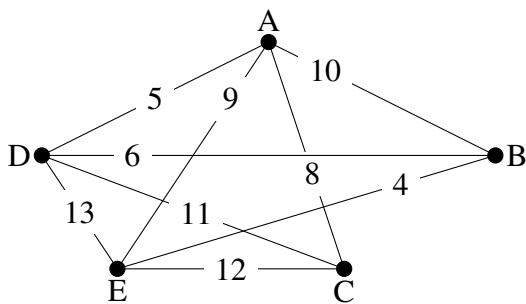
Explorations

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	B	C	D	E	F	G	H	I
A		X	X			X	X		
B			X		X				
C					X				
D					X				X
E							X		X
F								X	X
G								X	
H									X

(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 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?

14.7.1. 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 III

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:

$$AH = A \begin{bmatrix} x_1 & x_2 & x_3 & \cdots \\ y_1 & y_2 & y_3 & \cdots \end{bmatrix} = A \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \cdots \end{bmatrix} = \begin{bmatrix} A\mathbf{x}_1 & A\mathbf{x}_2 & A\mathbf{x}_3 & \cdots \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{x}'_1 & \mathbf{x}'_2 & \mathbf{x}'_3 & \cdots \end{bmatrix} = \begin{bmatrix} x'_1 & x'_2 & x'_3 & \cdots \\ y'_1 & y'_2 & y'_3 & \cdots \end{bmatrix} = H'.$$

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 $[a, b]^\mathsf{T}$ 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 θ radians has the following matrix representation:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

A negative value of θ performs a clockwise rotation. Choosing $\theta = \frac{\pi}{2}$ produces the rotation in Figure A.1.
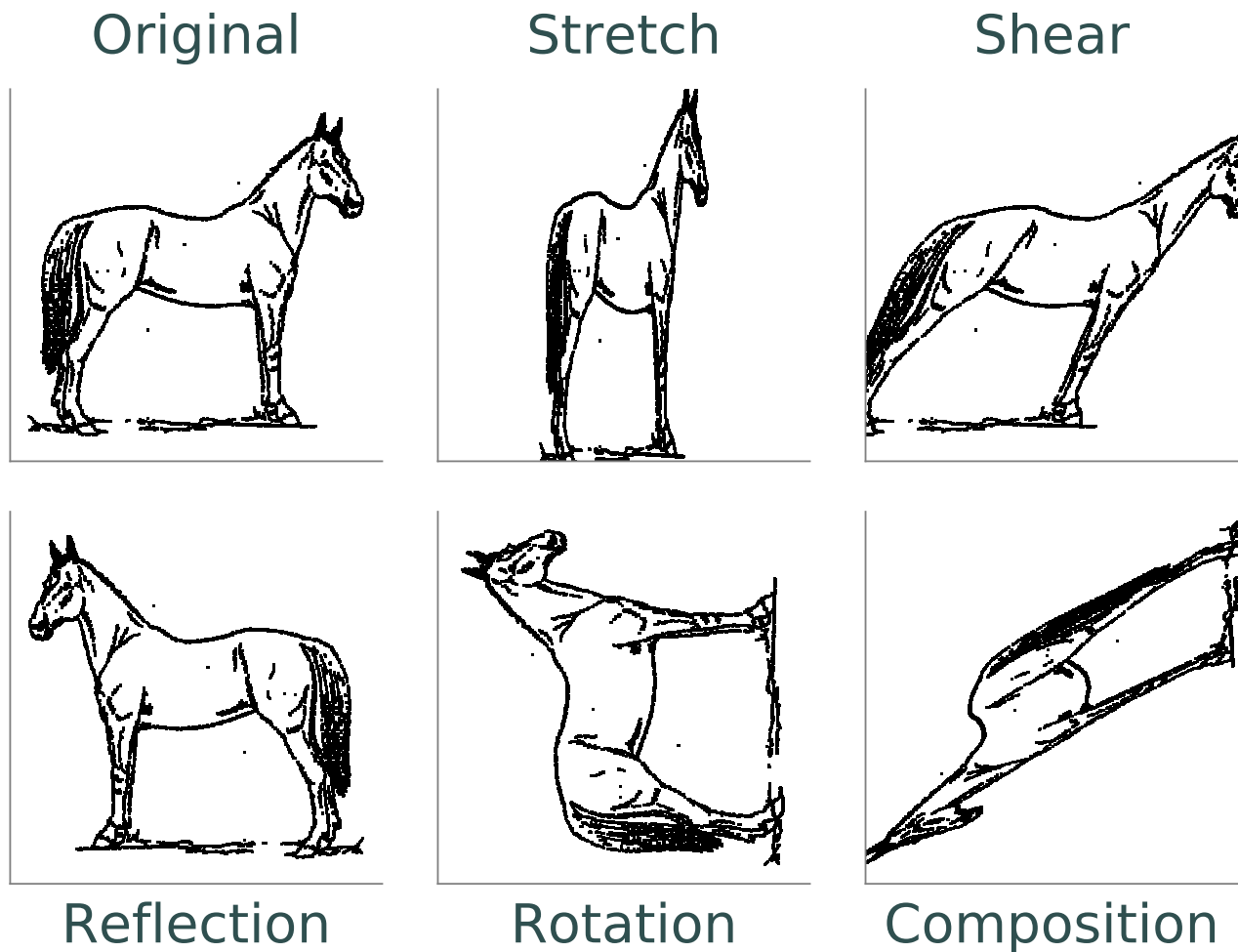
Figure A.1: The points stored in `horse.npy` under various linear transformations.

Problem A.1: Implement linear transformations.

rite 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 θ 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 $A\mathbf{x} + b$. 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 $\mathbf{b} = \left[\frac{3}{4}, \frac{1}{2}\right]^{\mathsf{T}}$, 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.

Original   Translation
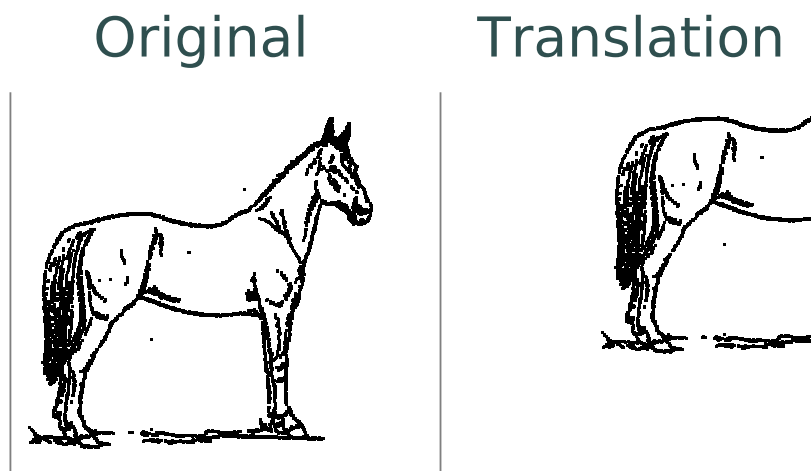
Affine transformations include all compositions of stretches, shears, rotations, reflections, and translations. For example, if S represents a shear and R a rotation, and if b is a vector, then $RS\mathbf{x} + b$ shears, then rotates, then translates $\mathbf{x}$.

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 ω (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 θ radians, then $\mathbf{p}(t) = R(t\omega)\mathbf{p}(0)$.
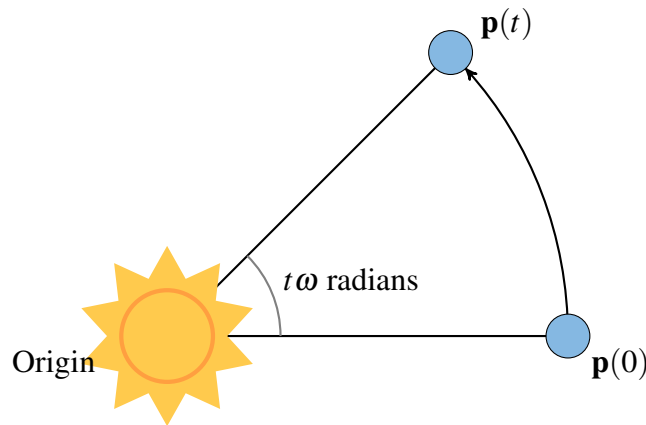
Composing the rotation with a translation shifts the center of rotation away from the origin, yielding more complicated motion.

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 ω_e and ω_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.

1. *Compute $\mathbf{p}_e(t)$ by rotating the initial vector $\mathbf{p}_e(0)$ counterclockwise about the origin by $t\omega_e$ radians.*

2. *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.*

3. *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 ω_e and ω_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`*).*
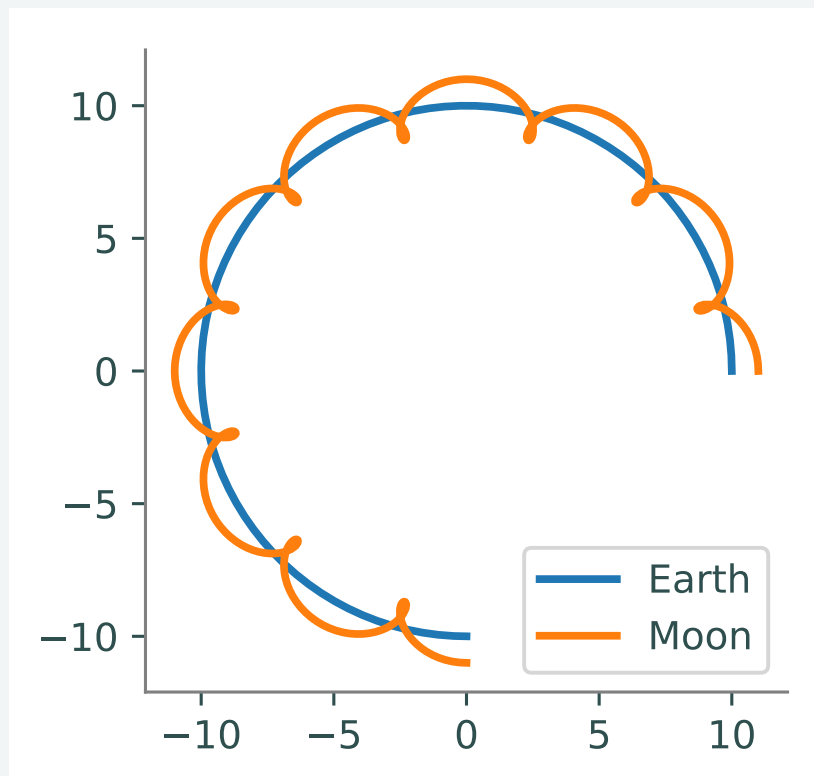
Timing Matrix Operations

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.

```python
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, ..., 2^{12} and plot *n* against the execution time. The result is displayed below on the left.

```python
>>> 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()
```
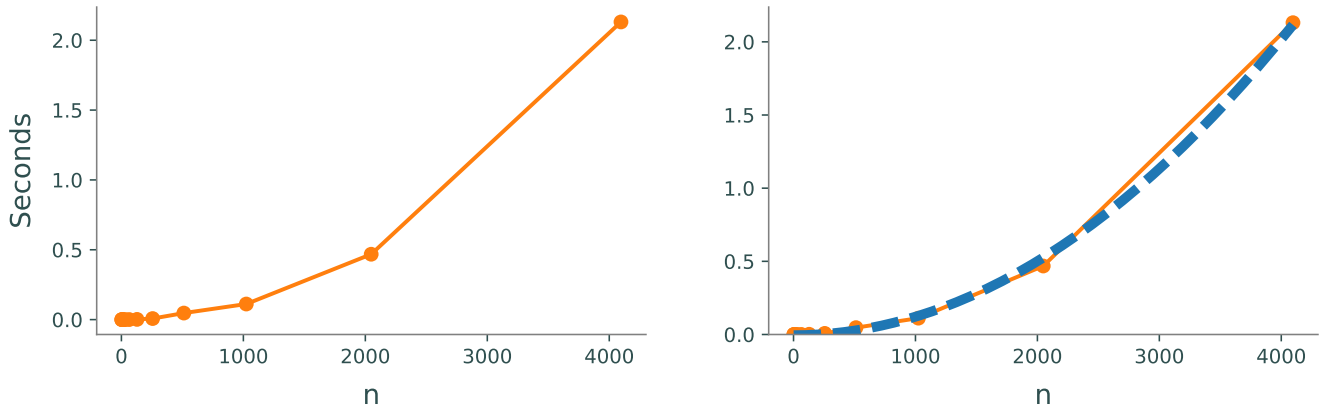
The figure on the left shows that the execution time for `random_matrix(n)` increases quadratically in n. In fact, the blue dotted line in the figure on the right is the parabola $y = an^2$, which fits nicely over the timed observations. Here a is a small constant, but it is much less significant than the exponent on the n. To represent this algorithm's growth, we ignore a altogether and write `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 `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

et 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:

$$y_i = \sum_{k=1}^{n} a_{ik} x_k \qquad\qquad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

These formulas are implemented below **without** using NumPy arrays or operations.

```python
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.
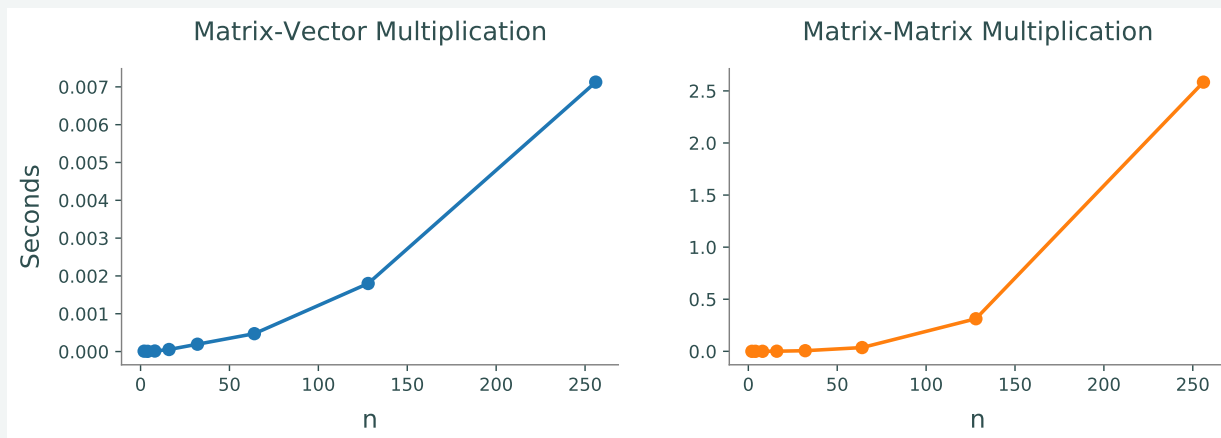
Logarithmic Plots

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, …— 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, … 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 = plot.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()
```
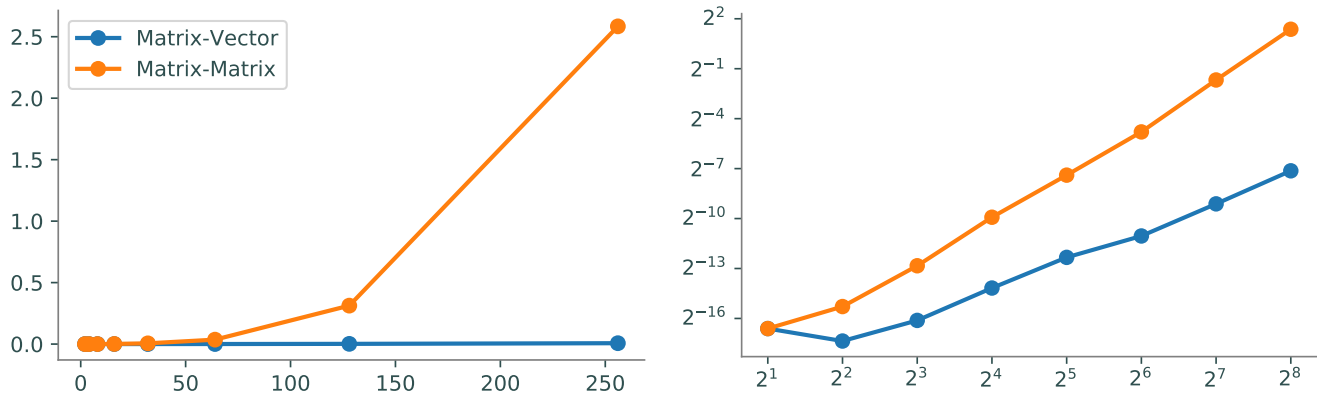
Figure A.5

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:

1. Iterating through loops is very expensive. Many of NumPy's operations are implemented in C, which are much faster than Python loops.

2. Arrays are designed specifically for matrix operations, while Python lists are general purpose.

3. 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 $A\mathbf{x} = b$, 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 $A\mathbf{x} = b$ on paper is reducing the augmented matrix $[A \mid b]$ 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.

$$
\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 4 & 2 & 3 \\ 4 & 7 & 8 & 9 \end{array}\right]
\longrightarrow
\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 4 & 7 & 8 & 9 \end{array}\right]
\longrightarrow
\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 3 & 4 & 5 \end{array}\right]
\longrightarrow
\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 0 & 3 & 3 \end{array}\right]
$$

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 $E_3 E_2 E_1 A = U$.

$$
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{array}\right]
\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{array}\right]
\left[\begin{array}{ccc} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]
\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 4 & 2 & 3 \\ 4 & 7 & 8 & 9 \end{array}\right]
=
\left[\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 0 & 3 & 3 \end{array}\right]
$$

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.

$$\begin{bmatrix} 10^{-15} & 1 \\ -1 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 10^{-15} & 1 \\ 0 & 10^{15} \end{bmatrix}$$

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 $E_1, \ldots, E_k$. 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:

$$E_k \ldots E_2 E_1 A = U \quad \longrightarrow \quad A = (E_k \ldots E_2 E_1)^{-1} U$$
$$= E_1^{-1} E_2^{-1} \ldots E_k^{-1} U$$
$$= LU.$$

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 5

1: **procedure** LU DECOMPOSITION(A)
2:     $m, n \leftarrow \text{shape}(A)$ ▷ Store the dimensions of A.
3:     $U \leftarrow \text{copy}(A)$ ▷ Make a copy of A with `np.copy()`.
4:     $L \leftarrow I_m$ ▷ The $m \times m$ identity matrix.
5:     **for** $j = 0 \ldots n-1$ **do**
6:         **for** $i = j+1 \ldots m-1$ **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**
>
> rite a function that finds the LU decomposition of a square matrix. You may assume that the de-
> composition exists and requires no row swaps.

Forward and Backward Substitution

If $PA = LU$ and $A\mathbf{x} = b$, then $LU\mathbf{x} = PA\mathbf{x} = Pb$. 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:

$$
\begin{bmatrix}
u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\
0 & u_{22} & u_{23} & \cdots & u_{2n} \\
0 & 0 & u_{33} & \cdots & u_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & u_{nn}
\end{bmatrix}
\begin{bmatrix}
x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n
\end{bmatrix}
=
\begin{bmatrix}
y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n
\end{bmatrix},
$$

$$
\begin{aligned}
u_{nn}x_n &= y_n, & x_n &= \frac{1}{u_{nn}}y_n, \\
u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1}, & x_{n-1} &= \frac{1}{u_{n-1,n-1}}\left(y_{n-1} - u_{n-1,n}x_n\right), \\
&\;\;\vdots & &\;\;\vdots \\
\sum_{j=k}^{n} u_{kj}x_j &= y_k, & x_k &= \frac{1}{u_{kk}}\left(y_k - \sum_{j=k+1}^{n} u_{kj}x_j\right).
\end{aligned}
\tag{2.2}
$$

> **Problem B.3: Program back and forward substitution.**
>
> *rite a function that, given A and b, solves the square linear system $A\mathbf{x} = b$. Use the function from Problem B to compute L and U, then use (2.1) and (2.2) to solve for $\mathbf{y}$, then $\mathbf{x}$. 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.[1]

```
>>> 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
det()	The determinant of a square matrix.
eig()	The eigenvalues and eigenvectors of a square matrix.
inv()	The inverse of an invertible matrix.
norm()	The norm of a vector or matrix norm of a matrix.
solve()	The solution to $A\mathbf{x} = b$ (the system need not be square).

This library also includes routines for computing matrix decompositions.

```
>>> from scipy import linalg as la

# Make a random matrix and a random vector.
```

---

[1] SciPy modules like `linalg` are really *packages*, which are not initialized when SciPy is imported alone.

```
>>> 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.

rite 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 $A\mathbf{x} = b$ with each of the following approaches:

1. Invert *A* with `la.inv()` and left-multiply the inverse to *b*.

2. Use `la.solve()`.

3. Use `la.lu_factor()` and `la.lu_solve()` to solve the system with the LU decomposition.

4. 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
`bsr_matrix`	Block Sparse Row	Specialized structure.
`coo_matrix`	Coordinate Format	Conversion among sparse formats.
`csc_matrix`	Compressed Sparse Column	Column-based operations and slicing.
`csr_matrix`	Compressed Sparse Row	Row-based operations and slicing.
`dia_matrix`	Diagonal Storage	Specialized structure.
`dok_matrix`	Dictionary of Keys	Element access, incremental construction.
`lil_matrix`	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()
```
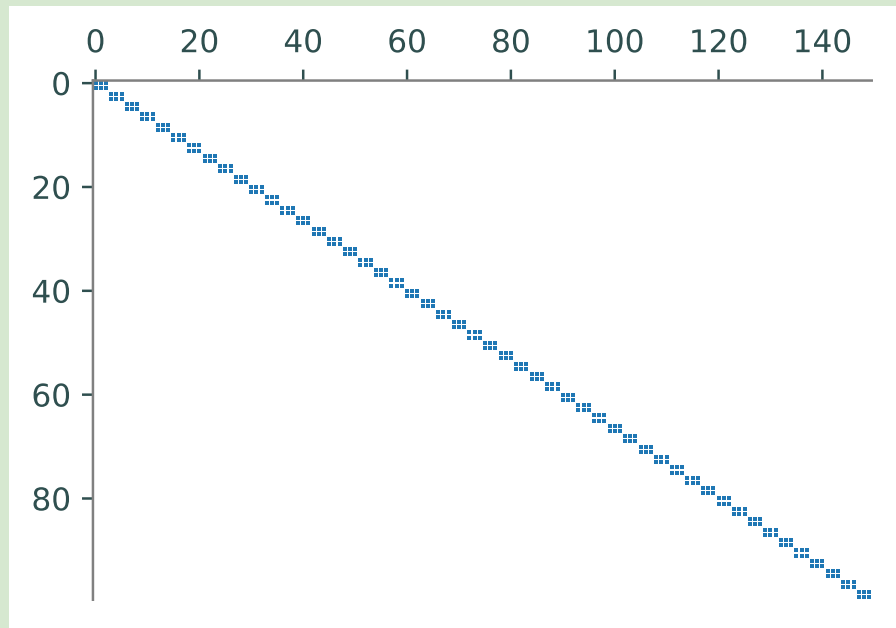

Problem B.5: Construct a large sparse matrix.

et I be the $n \times n$ identity matrix, and define

$$A = \begin{bmatrix} B & I & & & \\ I & B & I & & \\ & I & \ddots & \ddots & \\ & & \ddots & \ddots & I \\ & & & I & B \end{bmatrix}, \qquad B = \begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -4 \end{bmatrix},$$

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`.

```python
# 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 | numpy | scipy.sparse |
> |---|---|---|
> | Component-wise Addition | `A + B` | `A + B` |
> | Scalar Multiplication | `2 * A` | `2 * A` |
> | Component-wise Multiplication | `A * B` | `A.multiply(B)` |
> | Matrix Multiplication | `A.dot(B), A @ B` | `A * B, A.dot(B), A @ B` |
>
> 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().

rite 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 $A\mathbf{x} = b$ with each of the following approaches:

1. *Convert A to CSR format and use `scipy.sparse.linalg.spsolve()` (`spla.spsolve()`).*

2. *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 system, 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 5 uses two loops to compute the LU decomposition. With a little vectorization, the process can be reduced to a single loop.

Algorithm 6

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** $k = 0 \ldots n - 1$ **do**
6:         $L_{k+1:,k} \leftarrow U_{k+1:,k}/U_{k,k}$
7:         $U_{k+1:,k:} \leftarrow U_{k+1:,k:} - L_{k+1:,k}U_{k,k:}^\mathsf{T}$
8:     **return** L, U

Note that step 7 is an *outer product*, not the regular dot product ($\mathbf{x}\mathbf{y}^\mathsf{T}$ instead of the usual $\mathbf{x}^\mathsf{T}\mathbf{y}$). 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 kth 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[2] 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.

$$
\begin{bmatrix}
\times & \times & \times & \times \\
0 & 2 & \times & \times \\
0 & 4 & \times & \times \\
0 & -6 & \times & \times
\end{bmatrix}
\longrightarrow
\begin{bmatrix}
\times & \times & \times & \times \\
0 & -6 & \times & \times \\
0 & 4 & \times & \times \\
0 & 2 & \times & \times
\end{bmatrix}
\longrightarrow
\begin{bmatrix}
\times & \times & \times & \times \\
0 & -6 & \times & \times \\
0 & 0 & \times & \times \\
0 & 0 & \times & \times
\end{bmatrix}
$$

---

[2]*Complete pivoting* involves row and column swaps, but doing both operations is usually considered overkill.

A row swap is equivalent to left-multiplying by a type II elementary matrix, also called a *permutation 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 $P = P_k \ldots P_2 P_1$ yields $PA = LU$. The complete algorithm is given below.

Algorithm 7

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:     $P \leftarrow [0, 1, \ldots, n-1]$ ▷ See tip 2 below.
6:     **for** $k = 0 \ldots n-1$ **do**
7:         Select $i \geq k$ that maximizes $|U_{i,k}|$
8:         $U_{k,k:} \leftrightarrow U_{i,k:}$ ▷ Swap the two rows.
9:         $L_{k,:k} \leftrightarrow L_{i,:k}$ ▷ Swap the two rows.
10:         $P_k \leftrightarrow P_i$ ▷ Swap the two entries.
11:         $L_{k+1:,k} \leftarrow U_{k+1:,k}/U_{k,k}$
12:         $U_{k+1:,k:} \leftarrow U_{k+1:,k:} - L_{k+1:,k}U_{k,k:}^\mathsf{T}$
13:     **return** L, U, P

The following tips may be helpful for implementing this algorithm:

1. Since NumPy arrays are mutable, use `np.copy()` to reassign the rows of an array simultaneously.

2. 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 $[0, 1, \ldots, n]$. 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.]]
>>> 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 $LU\mathbf{a}_i = \mathbf{p}_i$ 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 kth 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 $\mathbf{z}^\mathsf{T}A\mathbf{z} > 0$ for all nonzero vectors $\mathbf{z}$. In addition, A is called *Hermitian* if $A = A^\mathsf{H} = \overline{A^\mathsf{T}}$. If A is Hermitian positive definite, it has a *Cholesky Decomposition* $A = U^\mathsf{H}U$ 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 8

1: **procedure** CHOLESKY DECOMPOSITION(A)
2: $\quad n, n \leftarrow \text{shape}(A)$
3: $\quad U \leftarrow \texttt{np.triu}(A)$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ ▷ Get the upper-triangular part of A.
4: $\quad$ **for** $i = 0 \ldots n-1$ **do**
5: $\qquad$ **for** $j = i+1 \ldots n-1$ **do**
6: $\qquad\quad U_{j,j:} \leftarrow U_{j,j:} - U_{i,j:}\overline{U_{ij}}/U_{ii}$
7: $\qquad U_{i,i:} \leftarrow U_{i,i:}/\sqrt{U_{ii}}$
8: $\quad$ **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.
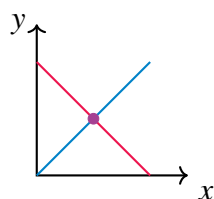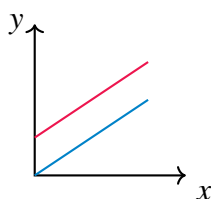
One Solution   No Solutions   Infinitely Many Solutions
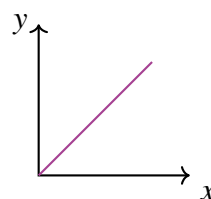
In the first diagram, there is a unique point of intersection, which means that there is only one (unique) solution to the two equations. In the second, there are no points of intersection and no solution. When no solution exists, this means that the two lines are parallel and they never intersect. The 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

$$x + y = 3$$
$$y - x = 5$$

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, $x = -1, y = 4$.
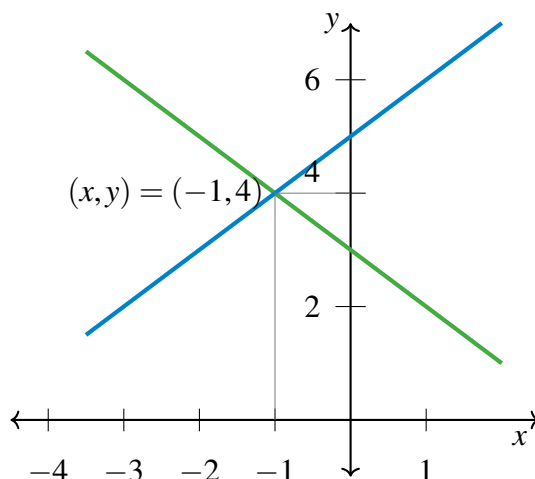
In the above example, we investigated the intersection point of two equations in two variables, x and y. Now we will consider the graphical solutions of three equations in two variables.

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.
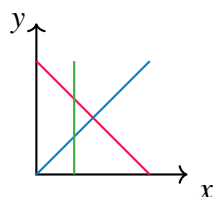
No Solution
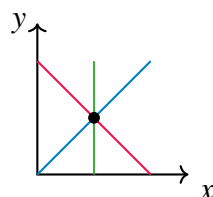
One Solution

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.
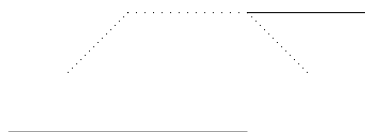
Notice how these two planes intersect in a line. This means that the points (x,y,z) on this line satisfy both equations in the system. Since the line contains infinitely many points, this system has infinitely many solutions.

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.
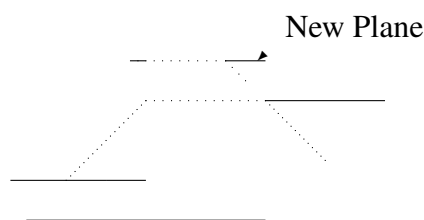
New Plane

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.

New Plane
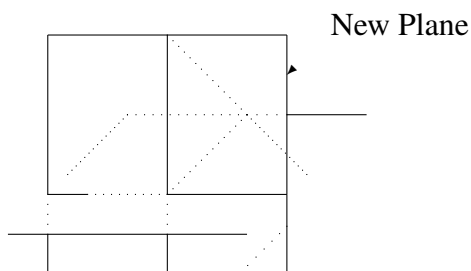
In this case, the three planes have a single point of intersection. Can you think of other types of solutions possible? Another is that the three planes could intersect in a line, resulting in infinitely many solutions, as in the following diagram.
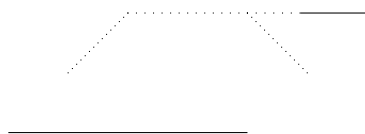
We have now seen how three equations in three variables can have no solution, a unique solution, or intersect in a line resulting in infinitely many solutions. It is also possible that the three equations graph the same plane, which also leads to infinitely many solutions.

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

<div>

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.*

</div>

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,

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m$$

where a_{ij} and b_j are real numbers. The above is a system of m equations in the n variables, $x_1, x_2 \cdots, x_n$. Written more simply in terms of summation notation, the above can be written in the form

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \ i = 1, 2, 3, \cdots, m$$

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*

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = 0$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = 0$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = 0$$

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 $x_1, \cdots, x_n$ which solve all of the equations. If such a set of values exists, we call $(x_1, \cdots, x_n)$ 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.

$$x + y = 3$$
$$y - x = 5$$

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.

1. *Interchange the order in which the equations are listed.*

2. *Multiply any equation by a nonzero number.*

3. *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

$$x + y = 7$$
$$2x - y = 8$$

has the same solution as the system

$$x + y = 7$$
$$-3y = -6$$

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.

$$x + y = (5) + (2) = 7$$
$$2x - y = 2(5) - (2) = 8$$

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

$$E_1 = b_1$$
$$E_2 = b_2$$
(3.1)

Then the following systems have the same solution set as 3.1:

1.
$$E_2 = b_2$$
$$E_1 = b_1$$
(3.2)

2.
$$E_1 = b_1$$
$$kE_2 = kb_2$$
(3.3)

   for any scalar k, provided $k \neq 0$.

3.
$$E_1 = b_1$$
$$E_2 + kE_1 = b_2 + kb_1$$
(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,

$$E_1 = x+y, \quad b_1 = 7$$
$$E_2 = 2x-y, \quad b_2 = 8$$

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

$$E_1 = b_1$$
$$E_2 + (-2)E_1 = b_2 + (-2)b_1$$

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

$$
\begin{aligned}
E_1 &= x + 3y + 6z, & b_1 &= 25 \\
E_2 &= 2x + 7y + 14z, & b_2 &= 58 \\
E_3 &= 2y + 5z, & b_3 &= 19
\end{aligned}
$$

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

$$x + 3y + 6z = 25$$
$$2x + 7y + 14z = 58$$
$$2y + 5z = 19$$

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, $\begin{bmatrix} 1 & 3 & 6 & | & 25 \end{bmatrix}$ 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

$$a_{11}x_1 + \cdots + a_{1n}x_n = b_1$$
$$\vdots$$
$$a_{m1}x_1 + \cdots + a_{mn}x_n = b_m$$

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*

1. *Switch two rows.*

2. *Multiply a row by a nonzero number.*

3. *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

$$x + 3y + 6z = 25$$
$$y + 2z = 8$$
$$z = 3$$

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*

1. *All nonzero rows are above any rows of zeros.*

2. *Each leading entry of a row is in a column to the right of the leading entries of any row above it.*

3. *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*

1. *All nonzero rows are above any rows of zeros.*

2. *Each leading entry of a row is in a column to the right of the leading entries of any rows above it.*

3. *Each leading entry of a row is equal to 1.*

4. *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.

$$\left[\begin{array}{ccccc|c} 1 & 0 & 6 & 5 & 8 & 2 \\ 0 & 0 & 1 & 2 & 7 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right], \left[\begin{array}{ccc|c} 1 & 3 & 5 & 4 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right], \left[\begin{array}{ccc|c} 1 & 0 & 6 & 0 \\ 0 & 1 & 4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

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.

$$\left[\begin{array}{ccccc|c} 1 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right], \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right], \left[\begin{array}{ccc|c} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

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 formof a matrix.
A **pivot column** is a column that contains a pivot position.

For example consider the following.

Example C.18: Pivot Position

Let

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 6 \\ 4 & 4 & 4 & 10 \end{bmatrix}$$

Where are the pivot positions and pivot columns of the augmented matrix A?

Solution. The row-echelon form of this matrix is

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & \frac{3}{2} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

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:

$$\begin{bmatrix} \boxed{1} & 2 & 3 & 4 \\ 3 & \boxed{2} & 1 & 6 \\ 4 & 4 & 4 & 10 \end{bmatrix}$$

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 **??**.

Reduced Row-Echelon Form Algorithmrrefalgorithm

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.

1. 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.

2. Use row operations to make the entries below the first pivot position (in the first pivot column) equal to zero.

3. 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.

4. 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.

5. 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.

Example C.19: Finding Row-Echelon Form and Reduced Row-Echelon Form of a Matrix

Let
$$A = \begin{bmatrix} 0 & -5 & -4 \\ 1 & 4 & 3 \\ 5 & 10 & 7 \end{bmatrix}$$

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 **??**.

1. 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.

$$\begin{bmatrix} \boxed{1} & 4 & 3 \\ 0 & -5 & -4 \\ 5 & 10 & 7 \end{bmatrix}$$

2. 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

$$\begin{bmatrix} 1 & 4 & 3 \\ 0 & -5 & -4 \\ 0 & 10 & 8 \end{bmatrix}$$

3. Now ignore the top row. Apply steps 1 and 2 to the smaller matrix

$$\begin{bmatrix} -5 & -4 \\ 10 & 8 \end{bmatrix}$$

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

$$\begin{bmatrix} -5 & -4 \\ 0 & 0 \end{bmatrix}$$

Our original matrix now looks like

$$\begin{bmatrix} 1 & 4 & 3 \\ 0 & -5 & -4 \\ 0 & 0 & 0 \end{bmatrix}$$

We can see that there are no more rows to modify.

4. 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

$$\begin{bmatrix} 1 & 4 & 3 \\ 0 & 1 & \frac{4}{5} \\ 0 & 0 & 0 \end{bmatrix}$$

This matrix is now in row-echelon form.

5. 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

$$\begin{bmatrix} 1 & 0 & -\frac{1}{5} \\ 0 & 1 & \frac{4}{5} \\ 0 & 0 & 0 \end{bmatrix}$$

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

$$2x + 4y - 3z = -1$$
$$5x + 10y - 7z = -2$$
$$3x + 6y + 5z = 9$$

Solution. The augmented matrix for this system is

$$\begin{bmatrix} 2 & 4 & -3 & -1 \\ 5 & 10 & -7 & -2 \\ 3 & 6 & 5 & 9 \end{bmatrix}$$

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

$$\begin{bmatrix} 2 & 4 & -3 & -1 \\ 0 & 0 & 1 & 1 \\ 3 & 6 & 5 & 9 \end{bmatrix}$$

Now, replace the third row with -3 times the first row plus to 2 times the third row. This yields

$$\begin{bmatrix} 2 & 4 & -3 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 21 \end{bmatrix}$$

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

$$\begin{bmatrix} 2 & 4 & -3 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 20 \end{bmatrix}$$

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

$$3x - y - 5z = 9$$
$$y - 10z = 0 \qquad\qquad (3.8)$$
$$-2x + y = -6$$

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

$$x - 5z = 3$$
$$y - 10z = 0$$

or

$$x = 3 + 5z$$
$$y = 10z$$

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

$$x = 3 + 5t$$
$$y = 10t$$
$$z = t$$

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

$$x = 3 + 5(4) = 23$$
$$y = 10(4) = 40$$
$$z = 4$$

♠

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

$$x + 2y - z + w = 3$$
$$x + y - z + w = 1$$
$$x + 3y - z + w = 5$$

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.

$$\begin{bmatrix} 1 & 2 & -1 & 1 & 3 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.9}$$

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

$$x = -1 + s - t$$
$$y = 2$$
$$z = s$$
$$w = t$$

It is customary to write this solution in the form

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} -1 + s - t \\ 2 \\ s \\ t \end{bmatrix} \tag{3.10}$$

♠

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.

$$\begin{bmatrix} 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

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

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

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.

$$\begin{bmatrix} 1 & 0 & 0 & | & 5 \\ 0 & 1 & 0 & | & 0 \\ 0 & 0 & 1 & | & 2 \end{bmatrix}$$

- *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.

$$\begin{bmatrix} 1 & 0 & 0 & | & 5 \\ 0 & 1 & 2 & | & -3 \\ 0 & 0 & 0 & | & 0 \end{bmatrix}$$

or

$$\begin{bmatrix} 1 & 0 & 0 & | & 5 \\ 0 & 1 & 0 & | & -3 \end{bmatrix}$$

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 $x_1, x_2, \cdots, x_n$ 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

$$x+2y-z+w=3$$
$$x+y-z+w=1$$
$$x+3y-z+w=5$$

Solution. Recall from the solution of Example C.22 that the row-echelon form of the augmented matrix of this system is given by

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

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

$$x=-1+s-t$$
$$y=2$$
$$z=s$$
$$w=t$$

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

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = 0$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = 0$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = 0$$

*Then, $x_1 = 0, x_2 = 0, \cdots, x_n = 0$ 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 $x_1, \cdots, x_n$ 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

$$2x + y - z = 0$$
$$x + 2y - 2z = 0$$

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

$$x = 0$$
$$y - z = 0$$

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

$$x = 0$$
$$y = z = t$$
$$z = t$$

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,

$$x = 0$$
$$y = 0 + t$$
$$z = 0 + t$$

can be written as

$$\left[\begin{array}{c} x \\ y \\ z \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ 0 \end{array}\right] + t \left[\begin{array}{c} 0 \\ 1 \\ 1 \end{array}\right]$$

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 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.

$$x + 4y + 3z = 0$$
$$3x + 12y + 9z = 0$$

Find the basic solutions to this system.

<u>**Solution.**</u> The augmented matrix of this system and the resulting reduced row-echelon form are

$$\begin{bmatrix} 1 & 4 & 3 & | & 0 \\ 3 & 12 & 9 & | & 0 \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} 1 & 4 & 3 & | & 0 \\ 0 & 0 & 0 & | & 0 \end{bmatrix}$$

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

$$x = -4s - 3t$$
$$y = s$$
$$z = t$$

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 $X_1, \cdots, X_n, V$ be column matrices. Then V is said to be a **linear combination** of the columns $X_1, \cdots, X_n$ if there exist scalars, $a_1, \cdots, a_n$ such that*

$$V = a_1 X_1 + \cdots + a_n X_n$$

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

$$\begin{bmatrix} \boxed{1} & 0 & -1 \\ 0 & \boxed{1} & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

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*
>
> 1. *the system has a unique solution if $r = n$*
>
> 2. *the system has infinitely many solutions if $r < n$*

We will not present a formal proof of this, but consider the following discussions.

1. *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!

2. *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.

3. *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

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.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 2 & 8 & 7 \\ 6 & -9 & 1 & 2 \end{bmatrix} \tag{4.1}$$

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 3×4 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 **Row**man **C**atholic.

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

$$A = \begin{bmatrix} A_1 & A_2 & \cdots & A_n \end{bmatrix}$$

Therefore, A_j is the jth 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 ith row and j^{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 ith row and the jth column of matrix A by a_{ij}. Then, we can write A in terms of its entries, as $A = \begin{bmatrix} a_{ij} \end{bmatrix}$. 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 2×3 zero matrix is $0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

Note there is a 2×3 zero matrix, a 3×4 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 $A = \begin{bmatrix} a_{ij} \end{bmatrix}$ and $B = \begin{bmatrix} b_{ij} \end{bmatrix}$, $a_{ij} = b_{ij}$ 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 3×2 while the other has size 2×3.

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 $A = \begin{bmatrix} a_{ij} \end{bmatrix}$ and $B = \begin{bmatrix} b_{ij} \end{bmatrix}$ be two $m \times n$ matrices. Then $A + B = C$ where C is the $m \times n$ matrix $C = \begin{bmatrix} c_{ij} \end{bmatrix}$ 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 2×3. 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*

$$A + B = B + A \tag{4.2}$$

- *Associative Law of Addition*

$$(A + B) + C = A + (B + C) \tag{4.3}$$

- *Existence of an Additive Identity*

There exists a zero matrix 0 such that
$$A + 0 = A \tag{4.4}$$

- *Existence of an Additive Inverse*

There exists a matrix $-A$ such that
$$A + (-A) = 0 \tag{4.5}$$

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 $A = \begin{bmatrix} a_{ij} \end{bmatrix}$ and k is a scalar, then $kA = \begin{bmatrix} ka_{ij} \end{bmatrix}$.

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 ith row of a column matrix, or the ith 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*

$$X = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}$$

*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

$$a_{11}x_1 + \cdots + a_{1n}x_n = b_1$$
$$\vdots$$
$$a_{m1}x_1 + \cdots + a_{mn}x_n = b_m$$

*We can express this system in **vector form** which is as follows:*

$$x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

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,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = x_1 \begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \end{bmatrix} + x_3 \begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \end{bmatrix}$$

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 $A = \begin{bmatrix} a_{ij} \end{bmatrix}$ 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.

$$1\begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + 2\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} + 0\begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix} + 1\begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 8 \\ 2 \\ 5 \end{bmatrix}$$

♠

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

$$a_{11}x_1 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{m1}x_1 + \cdots + a_{mn}x_n = b_m$$

*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 $x_1, x_2, \cdots, x_n$ 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 3×4 matrix by a 4×1 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

$$\overset{\text{these must match!}}{(m \times \quad \overbrace{n) (n} \times p \quad)} = m \times p$$

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 $B_1, ..., B_p$ 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 kth 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 2×3 and the second matrix has size 3×3. The inside numbers are equal, so A and B are conformable matrices. According to the above discussion AB will be a 2×3 matrix. Definition D.16 gives us a way to calculate each column of AB, as follows.

$$
\left[
\overbrace{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}}^{\text{First column}},
\overbrace{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}}^{\text{Second column}},
\overbrace{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}}^{\text{Third column}}
\right]
$$

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

Solution. In this case we are multiplying a matrix of size 3×1 by a matrix of size 1×4. The inside numbers match so the product is defined. Note that the product will be a matrix of size 3×4. Using Definition D.16, we can compute this product as follows

$$
\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 2 & 1 & 0 \end{bmatrix} =
\left[
\overbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \end{bmatrix}}^{\text{First column}},
\overbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 2 \end{bmatrix}}^{\text{Second column}},
\overbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \end{bmatrix}}^{\text{Third column}},
\overbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} 0 \end{bmatrix}}^{\text{Fourth column}}
\right]
$$

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 2×2 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 2×1 zero vector given by $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$. The result would be the 2×1 zero vector. Therefore, it is always the case that $A0 = 0$, for an appropriately sized zero matrix or vector.

D.1.4. The ij^{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^{th} entry of AB? It is the entry in the ith row and the jth 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 jth 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^{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^{th} entry of a product of matrices.

Definition D.21: The ij^{th} Entry of a Product

Let $A = \begin{bmatrix} a_{ij} \end{bmatrix}$ be an $m \times n$ matrix and let $B = \begin{bmatrix} b_{ij} \end{bmatrix}$ 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

$$(AB)_{ij} = \begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{bmatrix} \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{nj} \end{bmatrix} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

In other words, to find the (i, j)-entry of the product AB, or $(AB)_{ij}$, you multiply the ith row of A, on the left by the jth column of B. To express AB in terms of its entries, we write $AB = \begin{bmatrix} (AB)_{ij} \end{bmatrix}$.

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 3×3 matrix. We can first compute AB:

$$\begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 2 \\ 7 \end{bmatrix}, & \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \end{bmatrix}, & \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{bmatrix}$$

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 3×3 matrix as desired. Thus, the $(3,2)$-entry equals 42.

Now using Definition D.21, we can find that the $(3,2)$-entry equals

$$\sum_{k=1}^{2} a_{3k}b_{k2} = a_{31}b_{12} + a_{32}b_{22}$$
$$= 2 \times 3 + 6 \times 6 = 42$$

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 2×2. 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,

$$A(rB + sC) = r(AB) + s(AC) \tag{4.6}$$

$$(B + C)A = BA + CA \tag{4.7}$$

$$A(BC) = (AB)C \tag{4.8}$$

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 3×2 matrix became a 2×3 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

$$A^T = \begin{bmatrix} a_{ij} \end{bmatrix}^T = \begin{bmatrix} a_{ji} \end{bmatrix}$$

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 $A = \begin{bmatrix} a_{ij} \end{bmatrix}$, $A^T = \begin{bmatrix} a_{ji} \end{bmatrix}$. 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 2×3 matrix, while A^T is a 3×2 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

1. $\left(A^T\right)^T = A$

2. $(AB)^T = B^T A^T$

3. $(rA + sB)^T = rA^T + sB^T$

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 1×1 identity matrix, the second is the 2×2 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 δ_{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 $I_m A = A$.

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,

$$x + y = 1$$
$$x + 2y = 0$$

and

$$z + w = 0$$
$$z + 2w = 1$$

Writing the augmented matrix for these two systems gives

$$\begin{bmatrix} 1 & 1 & | & 1 \\ 1 & 2 & | & 0 \end{bmatrix}$$

for the first system and

$$\begin{bmatrix} 1 & 1 & | & 0 \\ 1 & 2 & | & 1 \end{bmatrix} \tag{4.9}$$

for the second.

Let's solve the first system. Take -1 times the first row and add to the second to get

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

Now take -1 times the second row and add to the first to get

$$\begin{bmatrix} 1 & 0 & | & 2 \\ 0 & 1 & | & -1 \end{bmatrix}$$

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

$$\begin{bmatrix} 1 & 1 & | & 1 & 0 \\ 1 & 2 & | & 0 & 1 \end{bmatrix}$$

and row reduced until we obtained

$$\begin{bmatrix} 1 & 0 & | & 2 & -1 \\ 0 & 1 & | & -1 & 1 \end{bmatrix}$$

and read off the inverse as the 2×2 matrix on the right side.

This exploration motivates the following important algorithm.

Matrix Inverse Algorithmmatrixinversionalgorithm 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

$$[A|I] = \begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 3 & 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$

Now we row reduce, with the goal of obtaining the 3×3 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

$$\begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -2 & 0 & -1 & 1 & 0 \\ 0 & -5 & -7 & -3 & 0 & 1 \end{bmatrix}$$

Then take 5 times the second row and add to -2 times the third row.

$$\begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -10 & 0 & -5 & 5 & 0 \\ 0 & 0 & 14 & 1 & 5 & -2 \end{bmatrix}$$

Next take the third row and add to -7 times the first row. This yields

$$\begin{bmatrix} -7 & -14 & 0 & -6 & 5 & -2 \\ 0 & -10 & 0 & -5 & 5 & 0 \\ 0 & 0 & 14 & 1 & 5 & -2 \end{bmatrix}$$

Now take $-\frac{7}{5}$ times the second row and add to the first row.

$$\begin{bmatrix} -7 & 0 & 0 & 1 & -2 & -2 \\ 0 & -10 & 0 & -5 & 5 & 0 \\ 0 & 0 & 14 & 1 & 5 & -2 \end{bmatrix}$$

Finally divide the first row by -7, the second row by -10 and the third row by 14 which yields

$$\begin{bmatrix} 1 & 0 & 0 & -\frac{1}{7} & \frac{2}{7} & \frac{2}{7} \\ 0 & 1 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & 0 & 1 & \frac{1}{14} & \frac{5}{14} & -\frac{1}{7} \end{bmatrix}$$

Notice that the left hand side of this matrix is now the 3×3 identity matrix I_3. Therefore, the inverse is the 3×3 matrix on the right hand side, given by

$$\begin{bmatrix} -\frac{1}{7} & \frac{2}{7} & \frac{2}{7} \\ \frac{1}{2} & -\frac{1}{2} & 0 \\ \frac{1}{14} & \frac{5}{14} & -\frac{1}{7} \end{bmatrix}$$

♠

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.

<u>Solution.</u> Write the augmented matrix $[A|I]$

$$\begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 2 & 2 & 4 & 0 & 0 & 1 \end{bmatrix}$$

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.

$$\begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -2 & 0 & -1 & 1 & 0 \\ 0 & -2 & 0 & -2 & 0 & 1 \end{bmatrix}$$

Next add -1 times the second row to the third row.

$$\begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & -2 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 \end{bmatrix}$$

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

$$\left(A^{-1}\right)AX = A^{-1}B$$
$$\left(A^{-1}A\right)X = A^{-1}B$$
$$IX = A^{-1}B$$
$$X = A^{-1}B$$

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.

$$x + z = 1$$
$$x - y + z = 3$$
$$x + y - z = 2$$

Solution. First, we can write the system of equations in matrix form

$$AX = \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = B \tag{4.10}$$

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:*
>
>   (a) *The determinant of a product of matrices is the product of the determinants.*
>
>   (b) *The determinant of a matrix is equal to the determinant of its transpose.*

E.1.1. Cofactors and 2×2 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 2×2 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 2×2 matrix.

Example E.2: A Two by Two Determinant

Find $\det(A)$ *for the matrix* $A = \begin{bmatrix} 2 & 4 \\ -1 & 6 \end{bmatrix}$.

<u>Solution.</u> From Definition E.1,

$$\det(A) = (2)(6) - (-1)(4) = 12 + 4 = 16$$

♠

The 2×2 determinant can be used to find the determinant of larger matrices. We will now explore how to find the determinant of a 3×3 matrix, using several tools including the 2×2 determinant.

We begin with the following definition.

Definition E.3: The ij^{th} Minor of a Matrix

*Let A be a 3×3 matrix. The ij^{th} **minor** of A, denoted as $minor(A)_{ij}$, is the determinant of the 2×2 matrix which results from deleting the ith row and the jth column of A.*
In general, if A is an $n \times n$ matrix, then the ij^{th} minor of A is the determinant of the $n-1 \times n-1$ matrix which results from deleting the ith row and the jth 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 $minor(A)_{12}$ and $minor(A)_{23}$.

<u>Solution.</u> First we will find $minor(A)_{12}$. By Definition E.3, this is the determinant of the 2×2 matrix which results when you delete the first row and the second column. This minor is given by

$$minor(A)_{12} = \det \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Using Definition E.1, we see that

$$\det \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix} = (4)(1) - (3)(2) = 4 - 6 = -2$$

Therefore $minor(A)_{12} = -2$.

Similarly, $minor(A)_{23}$ is the determinant of the 2×2 matrix which results when you delete the second row and the third column. This minor is therefore

$$minor(A)_{23} = \det \begin{bmatrix} 1 & 2 \\ 3 & 2 \end{bmatrix} = -4$$

Finding the other minors of A is left as an exercise. ♠

The ij^{th} minor of a matrix A is used in another important definition, given next.

Definition E.5: The ij^{th} Cofactor of a Matrix

Suppose A is an $n \times n$ matrix. The ij^{th} **cofactor**, denoted by $cof(A)_{ij}$ is defined to be

$$cof(A)_{ij} = (-1)^{i+j} minor(A)_{ij}$$

It is also convenient to refer to the cofactor of an entry of a matrix as follows. If a_{ij} is the ij^{th} entry of the matrix, then its cofactor is just $cof(A)_{ij}$.

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 $cof(A)_{12}$ and $cof(A)_{23}$.

Solution. We will use Definition E.5 to compute these cofactors.

First, we will compute $cof(A)_{12}$. Therefore, we need to find $minor(A)_{12}$. This is the determinant of the 2×2 matrix which results when you delete the first row and the second column. Thus $minor(A)_{12}$ is given by

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

Then,

$$cof(A)_{12} = (-1)^{1+2} minor(A)_{12} = (-1)^{1+2}(-2) = 2$$

Hence, $cof(A)_{12} = 2$.

Similarly, we can find $cof(A)_{23}$. First, find $minor(A)_{23}$, which is the determinant of the 2×2 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,

$$\text{cof}(A)_{23} = (-1)^{2+3}\, minor\,(A)_{23} = (-1)^{2+3}\,(-4) = 4$$

♠

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 3×3 matrix.

Definition E.7: The Determinant of a Three By Three Matrix

*Let A be a 3×3 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 ith row (column) is known as **expanding along the ith 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.

$$\det(A) = 1\overbrace{(-1)^{1+1}\begin{vmatrix} 3 & 2 \\ 2 & 1 \end{vmatrix}}^{\text{cof}(A)_{11}} + 4\overbrace{(-1)^{2+1}\begin{vmatrix} 2 & 3 \\ 2 & 1 \end{vmatrix}}^{\text{cof}(A)_{21}} + 3\overbrace{(-1)^{3+1}\begin{vmatrix} 2 & 3 \\ 3 & 2 \end{vmatrix}}^{\text{cof}(A)_{31}}$$

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 2×2 and 3×3 matrices. It turns out that the method used to calculate the determinant of a 3×3 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 4×4 matrix is the determinant of the 3×3 matrix you obtain when you delete the ith row and the jth column. Just as with the 3×3 determinant, we can compute the determinant of a 4×4 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 3×3 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 3×3 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 2×2 and 3×3 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

$$\det(A) = \sum_{j=1}^{n} a_{ij} cof(A)_{ij} = \sum_{i=1}^{n} a_{ij} cof(A)_{ij}$$

The first formula consists of expanding the determinant along the ith row and the second expands the determinant along the jth 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 col-

umn. This gives

$$\det(A) = \quad 1 \begin{vmatrix} 2 & 6 & 7 \\ 0 & 3 & 33.7 \\ 0 & 0 & -1 \end{vmatrix} + 0(-1)^{2+1} \begin{vmatrix} 2 & 3 & 77 \\ 0 & 3 & 33.7 \\ 0 & 0 & -1 \end{vmatrix} +$$

$$0(-1)^{3+1} \begin{vmatrix} 2 & 3 & 77 \\ 2 & 6 & 7 \\ 0 & 0 & -1 \end{vmatrix} + 0(-1)^{4+1} \begin{vmatrix} 2 & 3 & 77 \\ 2 & 6 & 7 \\ 0 & 3 & 33.7 \end{vmatrix}$$

and the only nonzero term in the expansion is

$$1 \begin{vmatrix} 2 & 6 & 7 \\ 0 & 3 & 33.7 \\ 0 & 0 & -1 \end{vmatrix}$$

Now find the determinant of this 3×3 matrix, by expanding along the first column to obtain

$$\det(A) = 1 \times \left(2 \times \begin{vmatrix} 3 & 33.7 \\ 0 & -1 \end{vmatrix} + 0(-1)^{2+1} \begin{vmatrix} 6 & 7 \\ 0 & -1 \end{vmatrix} + 0(-1)^{3+1} \begin{vmatrix} 6 & 7 \\ 3 & 33.7 \end{vmatrix} \right)$$

$$= 1 \times 2 \times \begin{vmatrix} 3 & 33.7 \\ 0 & -1 \end{vmatrix}$$

Next use Definition E.1 to find the determinant of this 2×2 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 2×2 or a 3×3 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

$$
X = A^{-1}B = \frac{1}{\det(A)}\mathrm{adj}(A)B
$$

Let x_i be the ith entry of X and b_j be the jth entry of B. Then this equation becomes

$$
x_i = \sum_{j=1}^{n} \left[a_{ij}\right]^{-1} b_j = \sum_{j=1}^{n} \frac{1}{\det(A)}\mathrm{adj}(A)_{ij} b_j
$$

where $\mathrm{adj}(A)_{ij}$ is the ij^{th} entry of $\mathrm{adj}(A)$.

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 ith 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 $X = [x_1, \cdots, x_n]^T$. Then Cramer's rule says

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where A_i is the matrix obtained by replacing the ith 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

$$\begin{bmatrix} 1 & 2 & 1 \\ 3 & 2 & 1 \\ 2 & -3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

<u>Solution.</u> 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

$$\mathbb{R}^n = \left\{ (x_1 \cdots x_n) : x_j \in \mathbb{R} \text{ for } j = 1, \cdots, n \right\}$$

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 $0 = (0, \cdots, 0)$ 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,

$$\mathbb{R}^2 = \left\{ (x_1, x_2) : x_j \in \mathbb{R} \text{ for } j = 1, 2 \right\}$$

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.
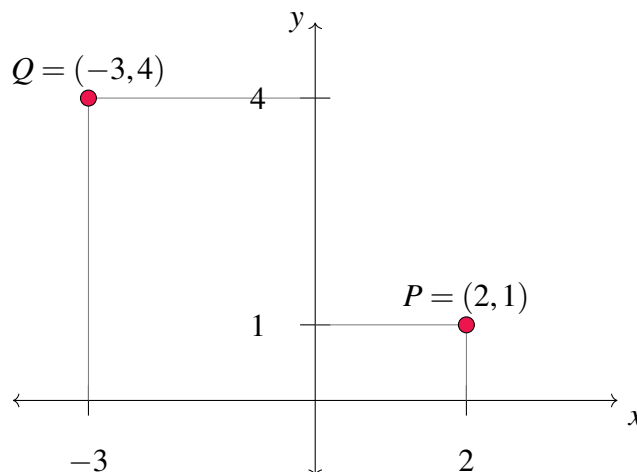
Hence, every element in $\mathbb{R}^2$ is identified by two components, x and y, in the usual manner. The coordinates x, y (or x_1, x_2) uniquely determine a point in the plan. Note that while the definition uses x_1 and x_2 to label the coordinates and you may be used to x and y, these notations are equivalent.

Now suppose $n = 3$. You may have previously encountered the 3-dimensional coordinate system, given by

$$\mathbb{R}^3 = \left\{ (x_1, x_2, x_3) : x_j \in \mathbb{R} \text{ for } j = 1, 2, 3 \right\}$$

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 $P = (p_1, \cdots, p_n)$ be the coordinates of a point in $\mathbb{R}^n$. Then the vector $\overrightarrow{0P}$ with its tail at $0 = (0, \cdots, 0)$ 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 $P = (p_1, \cdots, p_n) \in \mathbb{R}^n$ and $\overrightarrow{0P} = [p_1 \cdots p_n]^T \in \mathbb{R}^n$.

This definition is illustrated in the following picture for the special case of $\mathbb{R}^3$.

$$P = (p_1, p_2, p_3)$$

$$\overrightarrow{0P} = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}^T$$
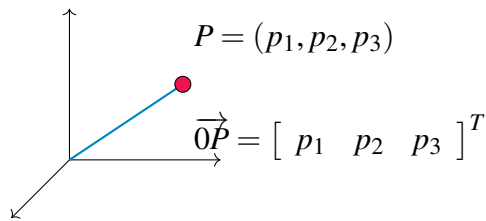
Thus every point P in $\mathbb{R}^n$ determines its position vector $\overrightarrow{0P}$. Conversely, every such position vector $\overrightarrow{0P}$ which has its tail at 0 and point at P determines the point P of $\mathbb{R}^n$.

Now suppose we are given two points, P, Q whose coordinates are $(p_1, \cdots, p_n)$ and $(q_1, \cdots, q_n)$ 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.

$$\overrightarrow{0P} = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}^T$$
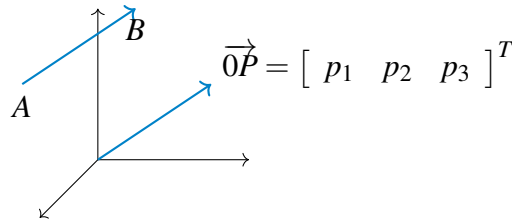
After moving it around, it is regarded as the same vector. Each vector, $\overrightarrow{0P}$ and $\overrightarrow{AB}$ has the same length (or magnitude) and direction. Therefore, they are equal.

Consider now the general definition for a vector in $\mathbb{R}^n$.

Definition F.2: Vectors in $\mathbb{R}^n$

Let $\mathbb{R}^n = \left\{ (x_1, \cdots, x_n) : x_j \in \mathbb{R} \text{ for } j = 1, \cdots, n \right\}$. 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, $\mathbf{p} = \overrightarrow{0P}$. 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

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 & 3 \end{bmatrix}^T$$

Notice that two vectors $\mathbf{u} = [u_1 \cdots u_n]^T$ and $\mathbf{v} = [v_1 \cdots v_n]^T$ are equal if and only if all corresponding components are equal. Precisely,

$$\mathbf{u} = \mathbf{v} \text{ if and only if}$$
$$u_j = v_j \text{ for all } j = 1, \cdots, n$$

Thus $\begin{bmatrix} 1 & 2 & 4 \end{bmatrix}^T \in \mathbb{R}^3$ and $\begin{bmatrix} 2 & 1 & 4 \end{bmatrix}^T \in \mathbb{R}^3$ but $\begin{bmatrix} 1 & 2 & 4 \end{bmatrix}^T \neq \begin{bmatrix} 2 & 1 & 4 \end{bmatrix}^T$ 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

$$\mathbf{i} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$$

$$\mathbf{j} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$$

$$\mathbf{k} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$$

We can write any vector $\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T$ 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

$$\mathbf{u} + \mathbf{v} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

$$= \begin{bmatrix} u_1 + v_1 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

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*

$$\mathbf{u} + 0 = \mathbf{u} \tag{6.1}$$

- *The Existence of an Additive Inverse*

$$\mathbf{u} + (-\mathbf{u}) = 0$$

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 $\mathbf{u}_1, \cdots, \mathbf{u}_n$ if there exist scalars, $a_1, \cdots, a_n$ such that*

$$\mathbf{v} = a_1\mathbf{u}_1 + \cdots + a_n\mathbf{u}_n$$

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 $\mathbf{u} + \mathbf{v}$.
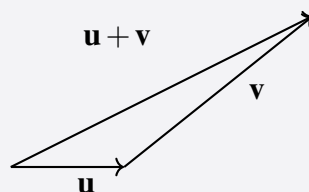
This definition is illustrated in the following picture in which $\mathbf{u}+\mathbf{v}$ is shown for the special case $n=3$.
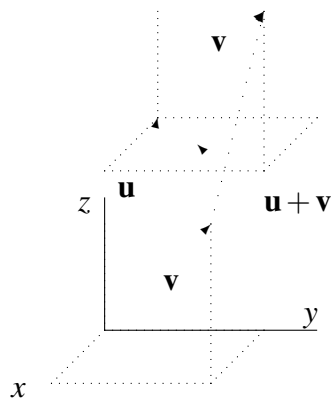
Notice the parallelogram created by $\mathbf{u}$ and $\mathbf{v}$ in the above diagram. Then $\mathbf{u}+\mathbf{v}$ is the directed diagonal of the parallelogram determined by the two vectors $\mathbf{u}$ and $\mathbf{v}$.

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 $\mathbf{v}$.
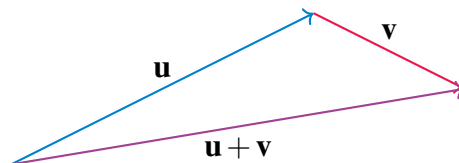
Next consider $\mathbf{u}-\mathbf{v}$. This means $\mathbf{u}+(-\mathbf{v})$. From the above geometric description of vector addition, $-\mathbf{v}$ is the vector which has the same length but which points in the opposite direction to $\mathbf{v}$. Here is a picture.
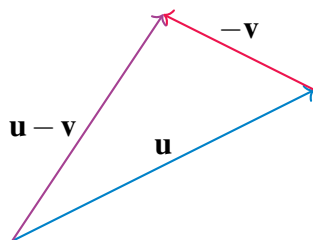
F.4 Length of a Vector

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

$$d(P,Q) = \sqrt{(P-Q)^2} \tag{6.1}$$

Consider now the case for $n = 2$, demonstrated by the following picture.
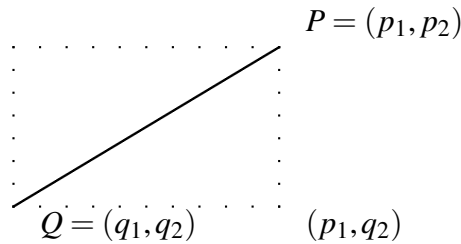
There are two points $P = (p_1, p_2)$ and $Q = (q_1, q_2)$ in the plane. The distance between these points is shown in the picture as a solid line. Notice that this line is the hypotenuse of a right triangle which is half of the rectangle shown in dotted lines. We want to find the length of this hypotenuse 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

$$\left(|p_1 - q_1|^2 + |p_2 - q_2|^2\right)^{1/2} = \left((p_1 - q_1)^2 + (p_2 - q_2)^2\right)^{1/2} \tag{6.2}$$

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).

$$P = (p_1, p_2, p_3)$$

$$(p_1, p_2, q_3)$$

$$Q = (q_1, q_2, q_3) \qquad (p_1, q_2, q_3)$$
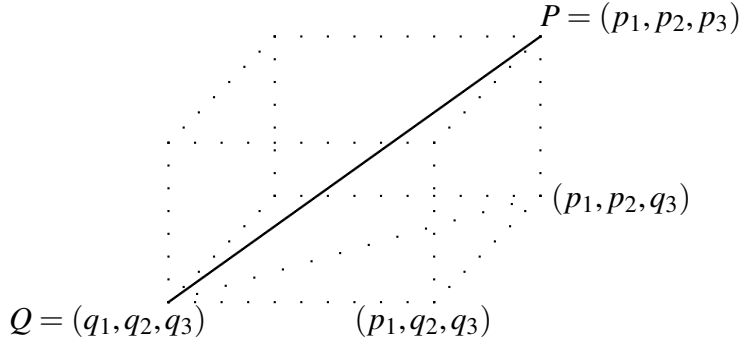
Here, we need to use Pythagorean Theorem twice in order to find the length of the solid line. First, by the Pythagorean Theorem, the length of the dotted line joining (q_1, q_2, q_3) and (p_1, p_2, q_3) equals

$$\left((p_1 - q_1)^2 + (p_2 - q_2)^2 \right)^{1/2}$$

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

$$\left(\left(\left((p_1 - q_1)^2 + (p_2 - q_2)^2 \right)^{1/2} \right)^2 + (p_3 - q_3)^2 \right)^{1/2}$$

$$= \left((p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2 \right)^{1/2} \tag{6.3}$$

This discussion motivates the following definition for the distance between points in $\mathbb{R}^n$.

Definition F.10: Distance Between Points

Let $P = (p_1, \cdots, p_n)$ and $Q = (q_1, \cdots, q_n)$ 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

$$2p_1 + 2p_2 + 2p_3 = -9 \qquad (6.4)$$

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 $0 = (0, \cdots, 0)$ and its tip at the point $U = (u_1, \cdots, u_n)$. 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

$$\mathbf{v} = \begin{bmatrix} 1 & -3 & 4 \end{bmatrix}^T$$

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 **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 **u**, we divide **v** by $\sqrt{26}$. The result is

$$\begin{aligned} \mathbf{u} &= \frac{1}{\|\mathbf{v}\|}\mathbf{v} \\ &= \frac{1}{\sqrt{26}} \begin{bmatrix} 1 & -3 & 4 \end{bmatrix}^T \\ &= \begin{bmatrix} \frac{1}{\sqrt{26}} & -\frac{3}{\sqrt{26}} & \frac{4}{\sqrt{26}} \end{bmatrix}^T \end{aligned}$$

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 **p** from 0 to P. The length of **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 $\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T$ and we multiply **u** by a scalar k. By Definition F.5, $k\mathbf{u} = \begin{bmatrix} ku_1 & ku_2 & ku_3 \end{bmatrix}^T$. Then, by using Definition F.10, the length of this vector is given by

$$\sqrt{\left((ku_1)^2 + (ku_2)^2 + (ku_3)^2 \right)} = |k|\sqrt{u_1^2 + u_2^2 + u_3^2}$$

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 **u** and k**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 **u** *and* **v** *drawn below.*

Draw $-$**u**, 2**v**, *and* $-\frac{1}{2}$**v**.

Solution.

In order to find $-$**u**, we preserve the length of **u** and simply reverse the direction. For 2**v**, we double the length of **v**, while preserving the direction. Finally $-\frac{1}{2}$**v** is found by taking half the length of **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 **u** *and* **v**

Sketch a picture of **u** $+ 2$**v**, **u** $- \frac{1}{2}$**v**.

Solution. The two vectors are shown below.
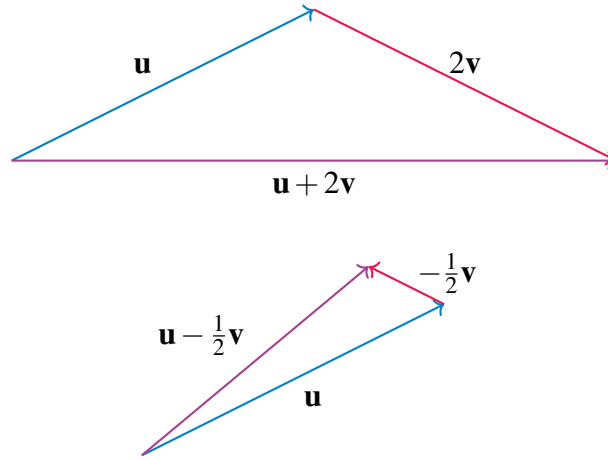
F.6 The Dot Product

> **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 **u**, **v** be two vectors in $\mathbb{R}^n$. Then we define the **dot product** **u** • **v** as*
>
> $$\mathbf{u} \bullet \mathbf{v} = \sum_{k=1}^{n} u_k v_k$$

The dot product **u** • **v** is sometimes denoted as $(\mathbf{u}, \mathbf{v})$ where a comma replaces •. 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

$$|\mathbf{u} \bullet \mathbf{v}| \leq \|\mathbf{u}\| \|\mathbf{v}\| \tag{6.1}$$

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 $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$

$$\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\| \tag{6.2}$$

and equality holds if and only if one of the vectors is a non-negative scalar multiple of the other. Also

$$|\|\mathbf{u}\| - \|\mathbf{v}\|| \leq \|\mathbf{u} - \mathbf{v}\| \tag{6.3}$$

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 θ 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 θ gives the included angle.
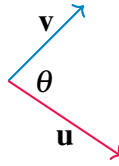
Proposition F.25: The Dot Product and the Included Angle

Let $\mathbf{u}$ and $\mathbf{v}$ be two vectors in $\mathbb{R}^n$, and let θ be the included angle. Then the following equation holds.

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos \theta$$

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,

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos \theta$$

Hence,

$$\cos\theta = \frac{\mathbf{u}\bullet\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

First, we can compute $\mathbf{u}\bullet\mathbf{v}$. By Definition F.19, this equals

$$\mathbf{u}\bullet\mathbf{v} = (2)(3)+(1)(4)+(-1)(1) = 9$$

Then,

$$\|\mathbf{u}\| = \sqrt{(2)(2)+(1)(1)+(1)(1)} = \sqrt{6}$$
$$\|\mathbf{v}\| = \sqrt{(3)(3)+(4)(4)+(1)(1)} = \sqrt{26}$$

Therefore, the cosine of the included angle equals

$$\cos\theta = \frac{9}{\sqrt{26}\sqrt{6}} = 0.7205766...$$

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 θ

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$$

to obtain $\cos \theta = -\frac{1}{2}$ and since we choose included angles between 0 and π 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 ($90°$).

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 λ 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 λ_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 **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
>
> $$AX = \lambda X \qquad (7.1)$$
>
> for some scalar λ. Then λ is called an **eigenvalue** of the matrix A and X is called an **eigenvector** of A associated with λ, or a λ-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

$$AX - \lambda X = 0$$
$$\text{or}$$
$$(A - \lambda I)X = 0$$

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.

$$
\begin{aligned}
X &= IX \\
&= \left((\lambda I - A)^{-1} (\lambda I - A) \right) X \\
&= (\lambda I - A)^{-1} \left((\lambda I - A) X \right) \\
&= (\lambda I - A)^{-1} 0 \\
&= 0
\end{aligned}
$$

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

$$\det(\lambda I - A) = 0 \qquad (7.2)$$

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 λ 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 λ of A is the number of times λ 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.

1. *First, find the eigenvalues λ of A by solving the equation $\det(xI - A) = 0$.*

2. *For each λ, 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 λ 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

$$\det\left(x\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}\right) = 0$$

$$\det\begin{bmatrix} x+5 & -2 \\ 7 & x-4 \end{bmatrix} = 0$$

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 λ. 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 $(2I - A)X = 0$.

$$\left(2\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix}\right)\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 7 & -2 \\ 7 & -2 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The augmented matrix for this system and corresponding reduced row-echelon form are given by

$$\begin{bmatrix} 7 & -2 & | & 0 \\ 7 & -2 & | & 0 \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} 1 & -\frac{2}{7} & | & 0 \\ 0 & 0 & | & 0 \end{bmatrix}$$

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 $((-3)I - A)X = 0$.

$$\left((-3) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -5 & 2 \\ -7 & 4 \end{bmatrix} \right) \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 2 & -2 \\ 7 & -7 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The augmented matrix for this system and corresponding reduced row-echelon form are given by

$$\left[\begin{array}{cc|c} 2 & -2 & 0 \\ 7 & -7 & 0 \end{array} \right] \rightarrow \cdots \rightarrow \left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0 & 0 & 0 \end{array} \right]$$

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 3×3 matrix.

Example G.7: Find the Eigenvalues and Eigenvectors

Find the eigenvalues and eigenvectors for the matrix

$$A = \begin{bmatrix} 5 & -10 & -5 \\ 2 & 14 & 2 \\ -4 & -8 & 6 \end{bmatrix}$$

<u>Solution.</u> 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.

$$(x-5)\left(x^2 - 20x + 100\right) = 0$$

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

$$\begin{bmatrix} 0 & 10 & 5 & | & 0 \\ -2 & -9 & -2 & | & 0 \\ 4 & 8 & -1 & | & 0 \end{bmatrix}$$

The reduced row-echelon form is

$$\begin{bmatrix} 1 & 0 & -\frac{5}{4} & | & 0 \\ 0 & 1 & \frac{1}{2} & | & 0 \\ 0 & 0 & 0 & | & 0 \end{bmatrix}$$

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 eigen-value $\lambda = 10$. 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 eigen-values 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 $z^\top M z$ is strictly positive for every non-zero column vector z of n real numbers. Here $z^\top$ denotes the `transpose` of z.[1]hen 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.

Positive semi-definite matrices are defined similarly, except that the above scalars $z^\top M z$ 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 $\langle z, w \rangle = z^\top M w$ 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.[2]Some authors use more general definitions of definiteness, including some non-symmetric real matrices, or non-Hermitian complex ones.

---

[1]W

[2]

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 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 z with real entries a and b, one has

$$z^\top z = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a^2 + b^2.$$

Seen as a complex matrix, for any non-zero column vector $\mathbf{z}$ with complex entries a and b one has

$$\mathbf{z}^* \mathbf{z} = \begin{bmatrix} \bar{a} & \bar{b} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \bar{a}a + \bar{b}b = |a|^2 + |b|^2$$

.

Either way, the result is positive since 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 z with entries a, b and c, we have

$$z^\top M z = \left(z^\top M\right) z \tag{7.1}$$

$$= \begin{bmatrix}(2a-b) & (-a+2b-c) & (-b+2c)\end{bmatrix} \tag{7.2}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{7.3}$$

$$= (2a-b)a + (-a+2b-c)b + (-b+2c)c \tag{7.4}$$

$$= 2a^2 - ba - ab + 2b^2 - cb - bc + 2c^2 \tag{7.5}$$

$$= 2a^2 - 2ab + 2b^2 - 2bc + 2c^2 \tag{7.6}$$

$$= a^2 + a^2 - 2ab + b^2 + b^2 - 2bc + c^2 + c^2 \tag{7.7}$$

$$= a^2 + (a-b)^2 + (b-c)^2 + c^2 \tag{7.8}$$

This result is a sum of squares, and therefore non-negative; and is zero only if $a = b = c = 0$, that is, when **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 **z**, the condition $\mathbf{z}^\top A^\top A\mathbf{z} = (A\mathbf{z})^\top (A\mathbf{z}) = \|A\mathbf{z}\|^2 > 0$, since the invertibility of matrix A means that $A\mathbf{z} \neq 0$.

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 $\begin{bmatrix} -1 & 1 \end{bmatrix} N \begin{bmatrix} -1 & 1 \end{bmatrix}^\top = -2 < 0.$

G.2.2. Eigenvalues

<div style="border:1px solid #800">

Theorem G.14: Eigenvalue Characterizations

Let M be an n × 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.*

</div>

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 $Q : \mathbb{R}^n \to \mathbb{R}$ such that $Q(x) = x^\top M x$ for all x. M can be assumed symmetric by replacing it with $\frac{1}{2}\left(M + M^\top\right)$.

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 $x^\top M x + x^\top b + c$ 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.[3] If $M \geq N > 0$ then $N^{-1} \geq M^{-1} > 0$.[4] Moreover, by the min-max theorem, the kth largest eigenvalue of M is greater than the kth 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.[5]

G.2.4.3. Addition

If M and N are positive definite, then the sum $M + N$ is also positive definite.[6]

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 $M = LL^*$.

---

[3], p. 397
[4], Corollary 7.7.4(a)
[5], Observation 7.1.3
[6]

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,[7] 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 $M^{\frac{1}{2}} > N^{\frac{1}{2}} > 0$.

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 α 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 $z^*Mz \geq 0$ for all complex z, and in particular for $z = [v, 0]^\top$. Then

---

[7], Theorem 7.2.6 with $k = 2$

$$\begin{bmatrix} v^* & 0 \end{bmatrix} \begin{bmatrix} A & B \\ B^* & D \end{bmatrix} \begin{bmatrix} v \\ 0 \end{bmatrix} = v^* A v \geq 0.$$

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 $x_1, \ldots, x_n$ 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 $x_1, \ldots, x_n$ 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 IV

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 ∞ 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

$$\mathbb{N} = \{1, 2, \cdots\}$$

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 $n = 1, 2, \cdots$, 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.

1. ***Base Case:** Show S_1 is true.*

2. *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 $n = 1, 2, \cdots$.

We can use this procedure to solve the following examples.

Example A.9: Proving by Induction

Prove by induction that $\sum_{k=1}^{n} k^2 = \dfrac{n(n+1)(2n+1)}{6}$.

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

$$\sum_{k=1}^{1} k^2 = \frac{1(1+1)(2(1)+1)}{6}$$
$$= \frac{6}{6}$$
$$= 1$$

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.

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \tag{1.1}$$

We want to show that this is true for $n + 1$.

Suppose we add $(n+1)^2$ to both sides of equation 1.1.

$$\sum_{k=1}^{n+1} k^2 = \sum_{k=1}^{n} k^2 + (n+1)^2$$
$$= \frac{n(n+1)(2n+1)}{6} + (n+1)^2$$

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 $n = 1, 2, \cdots$. ♠

Consider another example.

Example A.10: Proving an Inequality by Induction

Show that for all $n \in \mathbb{N}$, $\dfrac{1}{2} \cdot \dfrac{3}{4} \cdots \dfrac{2n-1}{2n} < \dfrac{1}{\sqrt{2n+1}}$.

<u>Solution.</u> 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 $\dfrac{1}{\sqrt{2n+3}}$. 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 Inteface 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 to long too find Incumbent solution:
It might be better to focus on trying to find heuristic solutions faster. You can do this w
```

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`

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=4OifjG2kIJQ

A.10 Contributors

lyryx

advancing learning[1]

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.

---

[1]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.

Contact Lyryx Today!

info@lyryx.com

lyryx
advancing learning

A First Course in Linear Algebra
an Open Text

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 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
Brigham Young University

R. Evans
Brigham Young University

J. Grout
Drake University

J. Humpherys
Brigham Young University

T. Jarvis
Brigham Young University

J. Whitehead
Brigham Young University

J. Adams
Brigham Young University

J. Bejarano
Brigham Young University

Z. Boyd
Brigham Young University

M. Brown
Brigham Young University

A. Carr
Brigham Young University

C. Carter
Brigham Young University

T. Christensen
Brigham Young University

M. Cook
Brigham Young University

R. Dorff
Brigham Young University

B. Ehlert
Brigham Young University

M. Fabiano
Brigham Young University

K. Finlinson
Brigham Young University

J. Fisher
Brigham Young University

R. Flores
Brigham Young University

R. Fowers
Brigham Young University

A. Frandsen
Brigham Young University

R. Fuhriman
Brigham Young University

S. Giddens
Brigham Young University

C. Gigena
Brigham Young University

M. Graham
Brigham Young University

F. Glines
Brigham Young University

C. Glover
Brigham Young University

M. Goodwin
Brigham Young University

R. Grout
Brigham Young University

D. Grundvig
Brigham Young University

E. Hannesson
Brigham Young University

J. Hendricks
Brigham Young University

A. Henriksen
Brigham Young University

I. Henriksen
Brigham Young University

C. Hettinger
Brigham Young University

S. Horst
Brigham Young University

K. Jacobson
Brigham Young University

J. Leete
Brigham Young University

J. Lytle
Brigham Young University

R. McMurray
Brigham Young University

S. McQuarrie
Brigham Young University

D. Miller
Brigham Young University

J. Morrise
Brigham Young University

M. Morrise
Brigham Young University

A. Morrow
Brigham Young University

R. Murray
Brigham Young University

J. Nelson
Brigham Young University

E. Parkinson
Brigham Young University

M. Probst
Brigham Young University

M. Proudfoot
Brigham Young University

D. Reber
Brigham Young University

H. Ringer
Brigham Young University

C. Robertson
Brigham Young University

M. Russell
Brigham Young University

R. Sandberg
Brigham Young University

C. Sawyer
Brigham Young University

M. Stauffer
Brigham Young University

J. Stewart
Brigham Young University

S. Suggs
Brigham Young University

A. Tate
Brigham Young University

T. Thompson
Brigham Young University

M. Victors
Brigham Young University

J. Webb
Brigham Young University

R. Webb
Brigham Young University

J. West
Brigham Young University

A. Zaitzeff
Brigham Young University

A.10.1. 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`