

# Engineering Mathematics



# **Engineering Mathematics**

Rico A. R. Picone

© 2025 Rico A. R. Picone

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the author.

## Contents

<b>1</b>	<b>Mathematics</b>	<b>1</b>
1.1	Truth	1
1.2	The Foundations of Mathematics	8
1.3	Problems	13
<b>2</b>	<b>Mathematical Reasoning, Logic, and Set Theory</b>	<b>15</b>
2.1	Introduction to Set Theory	15
2.2	Logical Connectives and Quantifiers	17
2.3	Problems	19
<b>3</b>	<b>Probability</b>	<b>21</b>
3.1	Probability and Measurement	21
3.2	Basic Probability Theory	22
3.3	Independence and Conditional Probability	23
3.4	Bayes' Theorem	25
3.5	Random Variables	29
3.6	Probability Density and Mass Functions	31
3.7	Expectation	35
3.8	Central Moments	38
3.9	Transforming Random Variables	39
3.10	Multivariate Probability and Correlation	46
3.11	Problems	56
<b>4</b>	<b>Statistics</b>	<b>57</b>
4.1	Populations, Samples, and Machine Learning	58
4.2	Estimation of Sample Mean and Variance	59
4.3	Confidence	68
4.4	Student Confidence	77

4.5	Regression	80
4.6	Problems	85
<b>5</b>	<b>Vector Calculus</b>	<b>89</b>
5.1	Divergence, Surface Integrals, and Flux	90
5.2	Curl, Line Integrals, and Circulation	96
5.3	Gradient	103
5.4	Stokes and Divergence Theorems	120
5.5	Problems	122
<b>6</b>	<b>Fourier and Orthogonality</b>	<b>125</b>
6.1	Fourier Series	125
6.2	Partial Sums of Fourier Series	129
6.3	Fourier Transform	133
6.4	Generalized Fourier Series and Orthogonality	141
6.5	Problems	148
<b>7</b>	<b>Ordinary Differential Equations</b>	<b>155</b>
7.1	SISO Linear Systems	155
7.2	A Unique Solution Exists	159
7.3	Homogeneous Solution	162
7.4	Particular Solution	165
7.5	General and Specific Solutions	169
7.6	Problems	172
<b>8</b>	<b>Partial Differential Equations</b>	<b>173</b>
8.1	Classifying PDEs	174
8.2	Boundary Value Problems	176
8.3	PDE Solution by Separation of Variables	183
8.4	The 1D Wave Equation	188
8.5	The Laplacian in Polar Coordinates and the Fourier-Bessel Series	194
8.6	Problems	201
<b>9</b>	<b>Optimization</b>	<b>207</b>
9.1	Gradient Descent	208
9.2	Constrained Linear Optimization	217
9.3	The Simplex Algorithm	219
9.4	The Calculus of Variations	223
9.5	Problems	231
<b>10</b>	<b>Nonlinear Analysis</b>	<b>233</b>

10.1	Nonlinear State-Space Models	234
10.2	Nonlinear System Characteristics	234
10.3	Simulating Nonlinear Systems	237
10.4	Problems	239
<b>A</b>	<b>Distribution Tables</b>	241
A.1	Gaussian Distribution Table	241
A.2	Student's T-Distribution Table	244
<b>B</b>	<b>Fourier and Laplace Tables</b>	245
B.1	Laplace Transforms	245
B.2	Fourier Transforms	246
<b>C</b>	<b>Mathematics Reference</b>	249
C.1	Quadratic Forms	249
C.2	Trigonometry	249
C.3	Matrix Inverses	253
C.4	Euler's Formulas	253
C.5	Laplace Transforms	254
	Bibliography	255





# 1 Mathematics



This chapter describes the foundations of mathematics and why it is so useful to engineers.

## 1.1 Truth



Before we can discuss mathematical truth, we should begin with a discussion of truth itself.<sup>1</sup> It is important to note that this is obviously extremely incomplete. My aim is to give a sense of the subject via brutal (mis)abbreviation.

Of course, the study of truth cannot but be entangled with the study of the world as such (metaphysics) and of knowledge (epistemology). Some of the following theories presuppose or imply a certain metaphysical and/or epistemological theory, but which these are is controversial.

### 1.1.1 Neo-Classical Theories of Truth

The neo-classical theories of truth take for granted that there *is* truth and attempt to explain what its precise nature is (Glanzberg 2018). What are provided here are modern understandings of theories developed primarily in the early 20<sup>th</sup> century.

**1.1.1.1 The Correspondence Theory** A version of what is called the **correspondence theory** of truth is the following.

A proposition is true iff there is an existing entity in the world that corresponds with it.

Such existing entities are called **facts**. Facts are relational in that their parts (e.g., subject, predicate, etc.) are related in a certain way.

Under this theory, then, if a proposition does not correspond to a fact, it is **false**.

This theory of truth is rather intuitive and consistently popular (David 2016).

1. For much of this lecture I rely on the thorough overview of (Glanzberg 2018).

**1.1.1.2 The Coherence Theory** The **coherence theory** of truth is adamant that the truth of any given proposition is only as good as its holistic system of propositions.<sup>2</sup> This includes (but typically goes beyond) a requirement for consistency of a given proposition with the whole and the self-consistency of the whole, itself—sometimes called **coherence**.

For parallelism, let's attempt a succinct formulation of this theory, cast in terms of propositions.

A proposition is true iff it has coherence with a system of propositions.

Note that this has no reference to facts, whatsoever. However, it need not necessarily preclude them.

**1.1.1.3 The Pragmatic Theory** Of the neo-classical theories of truth, this is probably the least agreed upon as having a single clear statement (Glanzberg 2018). However, as with **pragmatism** in general,<sup>3</sup> the pragmatic truth is oriented practically.

Perhaps the most important aspect of this theory is that it is thoroughly a correspondence theory, agreeing that true propositions are those that correspond to the world. However, there is a different focus here that differentiates it from correspondence theory, proper: it values as more true that which has some sort of practical use in human life.

We'll try to summarize pragmatism in two slogans with slightly different emphases; here's the first, again cast in propositional parallel.

A proposition is true iff it works.<sup>4</sup>

Now, there are two ways this can be understood: (a) the proposition "works" in that it empirically corresponds to the world or (b) the proposition "works" in that it has an effect that some agent intends. The former is pretty standard correspondence theory. The latter is new and fairly obviously has ethical implications, especially today.

Let us turn to a second formulation.

A proposition is true if it corresponds with a process of inquiry.<sup>5</sup>

This has two interesting facets: (a) an agent's active **inquiry** creates truth and (b) it is a sort of correspondence theory that requires a correspondence of a proposition

2. This is typically put in terms of "beliefs" or "judgments," but for brevity and parallelism I have cast it in terms of propositions. It is to this theory I have probably committed the most violence.

3. Pragmatism was an American philosophical movement of the early 20<sup>th</sup> century that valued the success of "practical" application of theories. For an introduction, see (Legg and Hookway 2019).

4. This is especially congruent with the work of William James (Legg and Hookway 2019).

5. This is especially congruent with the work of Charles Sanders Peirce (Legg and Hookway 2019).

with a process of inquiry, *not*, as in the correspondence theory, with a fact about the world. The latter has shades of both correspondence theory and coherence theory.

### 1.1.2 The Picture Theory

Before we delve into this theory, we must take a moment to clarify some terminology.

**1.1.2.1 States of Affairs and Facts** When discussing the correspondence theory, we have used the term **fact** to mean an actual state of things in the world. A problem arises in the correspondence theory, here. It says that a proposition is true iff there is a fact that corresponds with it. What of a negative proposition like “there are no cows in Antarctica”? We would seem to need a corresponding “negative fact” in the world to make this true. If a fact is taken to be composed of a complex of actual objects and relations, it is hard to imagine such facts.<sup>6</sup>

Furthermore, if a proposition is true, it seems that it is the corresponding fact that makes it so; what, then, makes a proposition false, since there is no fact to support the falsity? (Textor 2016)

And what of nonsense? There are some propositions like “there is a round cube” that are neither true nor false. However, the preceding correspondence theory cannot differentiate between false and nonsensical propositions.

A **state of affairs** is something possible that may or may not be actual (Textor 2016). If a state of affairs is actual, it is said to **obtain**. The picture theory will make central this concept instead of that of the fact.

**1.1.2.2 The Picture Theory of Meaning (And Truth)** The picture theory of meaning uses the analogy of the **model** or **picture** to explain the meaningfulness of propositions.<sup>7</sup>

A proposition names possible objects and arranges these names to correspond to a *state of affairs*.

See figure 1.1. This also allows for an easy account of truth, falsity, and nonsense.

6. But (Barker and Jago 2012) have attempted just that.

7. See (Wittgenstein 1922), (Biletzki and Matar 2018), (**glock2016**), and (Dolby 2016).

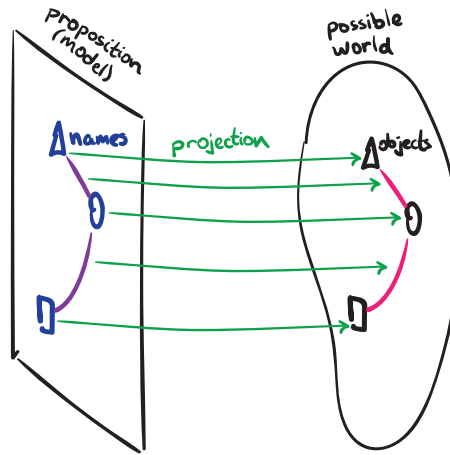


Figure 1.1. A representation of the picture theory.

**Nonsense** A sentence that appears to be a proposition is actually not if the arrangement of named objects is impossible. Such a sentence is simply **nonsense**.

**Truth** A proposition is true if the state of affairs it depicts obtains.

**Falsity** A proposition is false if the state of affairs it depicts does not obtain.

Now, some (Glock 2006) argue this is a correspondence theory and others (Dolby 2016) that it is not. In any case, it certainly solves some issues that have plagued the correspondence theory.

**1.1.2.3 “What Cannot Be Said Must Be Shown”** Something the picture theory does is declare a limit on what can meaningfully be said. A proposition (as defined above) must be potentially true or false. Therefore, something that cannot be false (something necessarily true) cannot be a proposition (Dolby 2016). And there are certain things that are necessarily true for language itself to be meaningful—paradigmatically, the logical structure of the world. What a proposition does, then, is *show*, via its own logical structure, the *necessary* (for there to be meaningful propositions at all) logical structure of the world.<sup>8</sup>

An interesting feature of this perspective is that it opens up **language itself** to analysis and limitation.<sup>9</sup> And, furthermore, it suggests that the set of what is, is smaller than the set of what can be meaningfully spoken about.

8. See, also, (Žižek 2012), pp. 25-26, from whom I stole the section title.

9. This was one of the contributions to the “linguistic turn” (Wikipedia 2019f) of philosophy in the early 20<sup>th</sup> century.

### 1.1.3 The Relativity of Truth

Each subject (i.e., agent) in the world, with their propositions, has a **perspective**: a given moment, a given place, an historical-cultural-linguistic situation. At the very least, the truth of propositions must account for this. Just how a theory of truth should do so is a matter of significant debate (Baghrarian and Carter 2019).

Some go so far as to be **skeptical** about truth (Klein 2015), regarding it to be entirely impossible. Others say that while a proposition may or may not be true, we could never come to know this.

Often underlying this conversation is the question of there being a common world in which we all participate, and, if so, whether or not we can properly represent this world in language such that multiple subjects could come to justifiably agree or disagree on the truth of a proposition. If every proposition is so relative that it is relevant to only the proposer, truth would seem of little value. On the other hand, if truth is understood to be “objective”—independent of subjective perspective—a number of objections can be made (Baghrarian and Carter 2019), such as that there is no non-subjective perspective from which to judge truth.

### 1.1.4 Other Ideas about Truth

There are too many credible ideas about truth to attempt a reasonable summary; however, I will attempt to highlight a few important ones.

**1.1.4.1 Formal Methods** A set of tools was developed for exploring theories of truth, especially correspondence theories.<sup>10</sup> Focus turned from **beliefs** to **sentences**, which are akin to propositions. (Recall that the above theories have already been recast in the more modern language of propositions.) Another aspect of these sentences under consideration is that they begin to be taken as **interpreted sentences**: they are already have meaning.

Beyond this, several technical apparatus are introduced that formalize criteria for truth. For instance, a sentence is given a sign  $\phi$ . A need arises to distinguish between the quotation of sentence  $\phi$  and the unquoted sentence  $\phi$ , which is then given the **quasi-quotation** notation  $\ulcorner \phi \urcorner$ . For instance, let  $\phi$  stand for *snow is white*; then  $\phi \rightarrow \textit{snow is white}$  and  $\ulcorner \phi \urcorner \rightarrow \textit{'snow is white'}$ . Tarski introduces **Convention T**, which states that for a fixed language  $L$  with fully interpreted sentences, (Glanzberg 2018)

An adequate theory of truth for  $L$  must imply for each sentence  $\phi$  of  $L$   
 $\ulcorner \phi \urcorner$  is true if and only if  $\phi$ .

Using the same example, then,

10. Especially notable here is the work of Alfred Tarski in the mid-20<sup>th</sup> century.

‘snow is white’ if and only if snow is white.

Convention T states a general rule for the adequacy of a theory of truth and is used in several contemporary theories.

We can see that these formal methods get quite technical and fun! For more, see (Hodges 2018b; Gómez-Torrente 2019; Hylton and Kemp 2019).

**1.1.4.2 Deflationary Theories** Deflationary theories of truth try to minimize or eliminate altogether the concept of or use of the term ‘truth’. For instance, the **redundancy theory** claim that (Glanzberg 2018):

To assert that ‘ $\phi$ ’ is true is just to assert that  $\phi$ .

Therefore, we can eliminate the use of ‘is true’.

For more or less, see (Stoljar and Damjanovic 2014).

**1.1.4.3 Language** It is important to recognize that language mediates truth; that is, truth is embedded in language. The way language in general affects theories of truth has been studied extensively. For instance, whether the **truth-bearer** is a belief or a proposition or a sentence—or something else—has been much discussed. The importance of the **meaning** of truth-bearers like sentences has played another large role. Theories of meaning, like the picture theory presented above, are often closely intertwined with theories of truth.

One of the most popular theories of meaning is called the **theory of use**:

For a large class of cases of the employment of the word “meaning” – though not for all – this word can be explained in this way: the meaning of a word is its use in the language. (Wittgenstein, Hacker, and Schulte 2010)

This theory is accompanied by the concept of **language-games**, which are loosely defined as rule-based contexts within which sentences have uses. The idea is that the meaning of a given sentence is its use in a network of meaning that is constantly evolving. This view tends to be understood as deflationary or relativistic about truth.

**1.1.4.4 Metaphysical and Epistemological Considerations** We began with the recognition that truth is intertwined with metaphysics and epistemology. Let’s consider a few such topics.

The first is **metaphysical realism**, which claims that there is a world existing objectively: independently of how we think about or describe it. This “realism” tends to be closely tied to, yet distinct from, **scientific realism**, which goes further, claiming the world is “actually” as science describes, independently of the scientific descriptions (e.g., there are actual objects corresponding to the phenomena we call atoms, molecules, light particles, etc.).

There have been many challenges to the realist claim (for some recent versions, see (Khlentzos 2016)) put forth by what is broadly called **anti-realism**. These vary, but often challenge the ability of realists to properly link language to supposed objects in the world.

**Metaphysical idealism** has been characterized as claiming that “mind” or “subjectivity” generate or completely compose the world, which has no being outside mind. **Epistemological idealism**, on the other hand, while perhaps conceding that there is a world independent of mind, claims all knowledge of the world is created through mind and for mind and therefore can never escape a sort of mind-world gap.<sup>11</sup> This epistemological idealism has been highly influential since the work of Immanuel Kant (Kant, Guyer, and Wood 1999) in the late 18<sup>th</sup> century, which ushered in the idea of the **noumenal world** in-itself and the **phenomenal world**, which is how the noumenal world presents to us. Many have held that phenomena can be known through inquiry, whereas noumena are inaccessible. Furthermore, what can be known is restricted by the categories pre-existent in the knower.

Another approach, taken by Georg Wilhelm Friedrich Hegel (Redding 2018) and other German idealists following Kant, is to reframe reality as thoroughly integrating subjectivity (Hegel and Miller 1998; Žižek 2012); that is, “everything turns on grasping and expressing the True, not only as *Substance*, but equally as *Subject*.” A subject’s proposition is true inasmuch as it corresponds with its **Notion** (approximately: the idea or meaning for the subject). Some hold that this idealism is compatible with a sort of metaphysical realism, at least as far as understanding is not independent of but rather beholden to reality (Žižek 2012; p. 906 ff.).

Clearly, all these ideas have many implications for theories of truth and vice versa.

### 1.1.5 Where This Leaves Us

The truth is hard. What may at first appear to be a simple concept becomes complex upon analysis. It is important to recognize that we have only sampled some highlights of the theories of truth. I recommend further study of this fascinating topic.

Despite the difficulties of finding definitive grounds for understanding truth, we are faced with the task of provisionally forging ahead. Much of what follows in the study of mathematics makes certain implicit and explicit assumptions about truth. However, we have found that the foundations of these assumptions may themselves be problematic. It is my contention that, despite the lack of clear foundations, *it is still worth studying engineering analysis, its mathematical foundations, and the foundations*

11. These definitions are explicated by (Guyer and Horstmann 2018).

*of truth itself.* My justification for this claim is that I find the utility and the beauty of this study highly rewarding.

## 1.2 The Foundations of Mathematics



Mathematics has long been considered exemplary for establishing truth. Primarily, it uses a method that begins with **axioms**—unproven propositions that include undefined terms—and uses logical **deduction** to **prove** other propositions (**theorems**): to show that they are necessarily true if the axioms are.

It may seem obvious that truth established in this way would always be relative to the truth of the axioms, but throughout history this footnote was often obscured by the “obvious” or “intuitive” universal truth of the axioms.<sup>12</sup> For instance, **Euclid** (Wikipedia 2019c) founded **geometry**—the study of mathematical objects traditionally considered to represent physical space, like points, lines, etc.—on axioms thought so solid that it was not until the early 19<sup>th</sup> century that **Carl Friedrich Gauss** (Wikipedia 2019b) and others recognized this was only one among many possible geometries (Kline 1982) resting on different axioms. Furthermore, **Aristotle** (Shields 2016) had acknowledged that reasoning must begin with undefined terms; however, even Euclid (presumably aware of Aristotle’s work) seemed to forget this and provided definitions, obscuring the foundations of his work and starting mathematics on a path that for over 2,000 years would forget its own relativity (Kline 1982; p. 101-2).

The foundations of Euclid were even shakier than its murky starting point: several unstated axioms were used in proofs and some proofs were otherwise erroneous. However, for two millennia, mathematics was seen as the field wherein truth could be established beyond doubt.

### 1.2.1 Algebra *Ex nihilo*

Although not much work new geometry appeared during this period, the field of **algebra** (Wikipedia 2019a)—the study of manipulations of symbols standing for numbers in general—began with no axiomatic foundation whatsoever. The Greeks had a notion of **rational numbers**, ratios of **natural numbers** (positive **integers**), and it was known that many solutions to algebraic equations were **irrational** (could not be expressed as a ratio of integers). But these irrational numbers, like virtually everything else in algebra, were gradually accepted because they were so useful in solving practical problems (they could be approximated by rational numbers and this seemed good enough). The rules of basic arithmetic were accepted as applying

12. Throughout this section, for the history of mathematics I rely heavily on (Kline 1982).



to these and other forms of new numbers that arose in algebraic solutions: **negative**, **imaginary**, and **complex numbers**.

### 1.2.2 The Application of Mathematics to Science

During this time, mathematics was being applied to **optics** and **astronomy**. Sir Isaac Newton then built **calculus** upon algebra, applying it to what is now known as **Newtonian mechanics**, which was really more the product of Leonhard Euler (Smith 2008; Wikipedia 2019e). Calculus introduced its own dubious operations, but the success of mechanics in describing and predicting physical phenomena was astounding. Mathematics was hailed as the language of God (later, Nature).

### 1.2.3 The Rigorization of Mathematics

It was not until Gauss created **non-Euclidean geometry**, in which Euclid's were shown to be one of many possible axioms compatible with the world, and William Rowan Hamilton (Wikipedia 2019k) created **quaternions** (Wikipedia 2019i), a number system in which multiplication is noncommunicative, that it became apparent something was fundamentally wrong with the way truth in mathematics had been understood. This started a period of **rigorization** in mathematics that set about axiomatizing and proving 19<sup>th</sup> century mathematics. This included the development of **symbolic logic**, which aided in the process of deductive reasoning.

An aspect of this rigorization is that mathematicians came to terms with the axioms that include undefined terms. For instance, a “point” might be such an undefined term in an axiom. A **mathematical model** is what we create when we attach these undefined terms to objects, which can be anything consistent with the axioms.<sup>13</sup> The system that results from proving theorems would then apply to anything “properly” described by the axioms. So two masses might be assigned “points” in a Euclidean geometric space, from which we could be confident that, for instance, the “distance” between these masses is the Euclidean norm of the line drawn between the points. It could be said, then, that a “point” in Euclidean geometry is **implicitly defined** by its axioms and theorems, and nothing else. That is, mathematical objects are *not* inherently tied to the physical objects to which we tend to apply them. Euclidean geometry is not the study of physical space, as it was long considered—it is the study of the objects implicitly defined by its axioms and theorems.

13. The branch of mathematics called *model theory* concerns itself with general types of models that can be made from a given formal system, like an axiomatic mathematical system. For more on model theory, see (Hodges 2018a). It is noteworthy that the engineering/science use of the term “mathematical model” is only loosely a “model” in the sense of model theory.

### 1.2.4 The Foundations of Mathematics Are Built

The building of the modern foundations mathematics began with clear axioms, solid reasoning (with symbolic logic), and lofty yet seemingly attainable goals: prove theorems to support the already ubiquitous mathematical techniques in geometry, algebra, and calculus from axioms; furthermore, prove that these axioms (and things they imply) do not contradict each other, i.e., are **consistent**, and that the axioms are not results of each other (one that can be derived from others is a **theorem**, not an axiom).

**Set theory** is a type of formal axiomatic system that all modern mathematics is expressed with, so set theory is often called the **foundation** of mathematics (Bagaria 2019). We will study the basics in (**ch:set\_theory**). The primary objects in set theory are **sets**: informally, collections of mathematical objects. There is not just one a single set of axioms that is used as the foundation of all mathematics for reasons will review in a moment. However, the most popular set theory is **Zermelo-Fraenkel set theory with the axiom of choice** (ZFC). The axioms of ZF sans C are as follows. (Bagaria 2019)

**Extensionality** If two sets  $A$  and  $B$  have the same elements, then they are equal.

**Empty set** There exists a set, denoted by  $\emptyset$  and called the empty set, which has no elements.

**Pair** Given any sets  $A$  and  $B$ , there exists a set, denoted by  $\{A, B\}$ , which contains  $A$  and  $B$  as its only elements. In particular, there exists the set  $\{A\}$  which has  $A$  as its only element.

**Power set** For every set  $A$  there exists a set, denoted by  $\mathcal{P}(A)$  and called the power set of  $A$ , whose elements are all the subsets of  $A$ .

**Union** For every set  $A$ , there exists a set, denoted by  $\bigcup A$  and called the union of  $A$ , whose elements are all the elements of the elements of  $A$ .

**Infinity** There exists an infinite set. In particular, there exists a set  $Z$  that contains  $\emptyset$  and such that if  $A \in Z$ , then  $\bigcup\{A, \{A\}\} \in Z$ .

**Separation** For every set  $A$  and every given property, there is a set containing exactly the elements of  $A$  that have that property. A property is given by a formula  $\varphi$  of the first-order language of set theory. Thus, separation is not a single axiom but an axiom schema, that is, an infinite list of axioms, one for each formula  $\varphi$ .

**Replacement** For every given definable function with domain a set  $A$ , there is a set whose elements are all the values of the function.

ZFC also has the axiom of choice. (Bagaria 2019)

**Choice** For every set  $A$  of pairwise-disjoint non-empty sets, there exists a set that contains exactly one element from each set in  $A$ .

### 1.2.5 The Foundations Have Cracks

The foundationalists' goal was to prove that some set of axioms from which all of mathematics can be derived is both consistent (contains no contradictions) and complete (every true statement is provable). The work of Kurt Gödel (Kennedy 2018) in the mid 20<sup>th</sup> century shattered this dream by proving in his **first incompleteness theorem** that any consistent formal system within which one can do some amount of basic arithmetic is **incomplete**! His argument is worth reviewing (see (Raatikainen 2018)), but at its heart is an **undecidable** statement like "This sentence is unprovable." Let  $U$  stand for this statement. If it is true it is unprovable. If it is provable it is false. Therefore, it is true iff it is provable. Then he shows that if a statement  $A$  that essentially says "arithmetic is consistent" is provable, then so is the undecidable statement  $U$ . But if  $U$  is to be consistent, it cannot be provable, and, therefore neither can  $A$  be provable!

This is problematic. It tells us virtually any conceivable axiomatic foundation of mathematics is incomplete. If one is complete, it is inconsistent (and therefore worthless). One problem this introduces is that a true theorem may be impossible to prove; but, it turns out, we can never know that in advance of its proof if it is provable.

But it gets worse: Gödel's **second incompleteness theorem** shows that such systems cannot even be shown to be consistent! This means, at any moment, someone could find an inconsistency in mathematics, and not only would we lose some of the theorems: we would lose them all. This is because, by what is called the **material implication** (Kline 1982; pp. 187-8 264), if one contradiction can be found, *every proposition can be proven* from it. And if this is the case, all (even proven) theorems in the system would be suspect.

Even though no contradiction has yet appeared in ZFC, its axiom of choice, which is required for the proof of most of what has thus far been proven, generates the **Banach-Tarski paradox** that says a sphere of diameter  $x$  can be partitioned into a finite number of pieces and recombined to form *two* spheres of diameter  $x$ . Troubling, to say the least! Attempts were made for a while to eliminate the use of the axiom of choice, but our buddy Gödel later proved that if ZF is consistent, so is ZFC ( p. 267).

### 1.2.6 Mathematics Is Considered Empirical

Since its inception, mathematics has been applied extensively to the modeling of the world. Despite its cracked foundations, it has striking utility. Many recent leading minds of mathematics, philosophy, and science suggest we treat mathematics as **empirical**, like any science, subject to its success in describing and predicting events in the world. As (Kline 1982) summarizes,

The upshot [...] is that sound mathematics must be determined not by any one foundation which may some day prove to be right. The “correctness” of mathematics must be judged by its application to the physical world. Mathematics is an empirical science much as Newtonian mechanics. It is correct only to the extent that it works and when it does not, it must be modified. It is not a priori knowledge even though it was so regarded for two thousand years. It is not absolute or unchangeable.

---

### 1.3 Problems





## 2 Mathematical Reasoning, Logic, and Set Theory



In order to communicate mathematical ideas effectively, **formal languages** have been developed within which **logic**, i.e. deductive (mathematical) **reasoning**, can proceed. **Propositions** are statements that can be either true  $\top$  or false  $\perp$ . Axiomatic systems begin with statements (axioms) assumed true. **Theorems** are **proven** by deduction. In many forms of logic, like **propositional calculus** (Wikipedia 2019h), compound propositions are constructed via **logical connectives** like “and” and “or” of atomic propositions (see section 2.2). In others, like **first-order logic** (Wikipedia 2019d), there are also logical **quantifiers** like “for every” and “there exists.”

The mathematical objects and operations about which most propositions are made are expressed in terms of **set theory**, which was introduced in section 1.2 and will be expanded upon in section 2.1. We can say that mathematical reasoning is comprised of mathematical objects and operations expressed in set theory and logic allows us to reason therewith.

### 2.1 Introduction to Set Theory



**Set theory** is the language of the modern foundation of mathematics, as discussed in chapter 1. It is unsurprising, then, that it arises throughout the study of mathematics. We will use set theory extensively in chapter 3 on probability theory.

The axioms of ZFC set theory were introduced in chapter 1. Instead of proceeding in the pure mathematics way of introducing and proving theorems, we will opt for a more applied approach in which we begin with some simple definitions and include basic operations. A more thorough and still readable treatment is given by (Ciesielski 1997) and a very gentle version by (Enderton 1977).

A **set** is a collection of objects. Set theory gives us a way to describe these collections. Often, the objects in a set are numbers or sets of numbers. However, a set could represent collections of zebras and trees and hairballs. For instance, here are

some sets:

$$\{1, 5, \pi\} \quad \{\text{zebra named "Calvin", a burnt cheeto}\} \quad \{\{1, 2\}, \{5, \text{hippo}, 7\}, 62\}.$$

A **field** is a set with special structure. This structure is provided by the **addition** (+) and **multiplication** (×) operators and their inverses **subtraction** (−) and **division** (÷). The quintessential example of a field is the set of **real numbers**  $\mathbb{R}$ , which admits these operators, making it a field. The reals  $\mathbb{R}$ , the complex numbers  $\mathbb{C}$ , the integers  $\mathbb{Z}$ , and the natural numbers<sup>1</sup>  $\mathbb{N}$  are the fields we typically consider.

**Set membership** is the belonging of an object to a set. It is denoted with the symbol  $\in$ , which can be read “is an element of,” for element  $x$  and set  $X$ :

$$x \in X.$$

For instance, we might say  $7 \in \{1, 7, 2\}$  or  $4 \notin \{1, 7, 2\}$ . Or, we might declare that  $a$  is a real number by stating:  $x \in \mathbb{R}$ .

**Set operations** can be used to construct new sets from established sets. We consider a few common set operations, now.

The **union**  $\cup$  of sets is the set containing all the elements of the original sets (no repetition allowed). The union of sets  $A$  and  $B$  is denoted  $A \cup B$ . For instance, let  $A = \{1, 2, 3\}$  and  $B = \{-1, 3\}$ ; then

$$A \cup B = \{1, 2, 3, -1\}.$$

The **intersection**  $\cap$  of sets is a set containing the elements common to all the original sets. The intersection of sets  $A$  and  $B$  is denoted  $A \cap B$ . For instance, let  $A = \{1, 2, 3\}$  and  $B = \{2, 3, 4\}$ ; then

$$A \cap B = \{2, 3\}.$$

If two sets have no elements in common, the intersection is the **empty set**  $\emptyset = \{\}$ , the unique set with no elements.

The **set difference** of two sets  $A$  and  $B$  is the set of elements in  $A$  that aren’t also in  $B$ . It is denoted  $A \setminus B$ . For instance, let  $A = \{1, 2, 3\}$  and  $B = \{2, 3, 4\}$ . Then

$$A \setminus B = \{1\} \quad B \setminus A = \{4\}.$$

A **subset**  $\subseteq$  of a set is a set, the elements of which are contained in the original set. If the two sets are equal, one is still considered a subset of the other. We call a subset that is not equal to the other set a **proper subset**  $\subset$ . For instance, let  $A = \{1, 2, 3\}$  and  $B = \{1, 2\}$ . Then

$$B \subseteq A \quad B \subset A \quad A \subseteq A.$$

1. When the natural numbers include zero, we write  $\mathbb{N}_0$ .



The **complement** of a subset is a set of elements of the original set that aren't in the subset. For instance, if  $B \subseteq A$ , then the complement of  $B$ , denoted  $\overline{B}$  is

$$\overline{B} = A \setminus B.$$

The **cartesian product** of two sets  $A$  and  $B$  is denoted  $A \times B$  and is the set of all ordered pairs  $(a, b)$  where  $a \in A$  and  $b \in B$ . It's worthwhile considering the following notation for this definition:

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

which means “the cartesian product of  $A$  and  $B$  is the ordered pair  $(a, b)$  such that  $a \in A$  and  $b \in B$ ” in **set-builder notation** (Wikipedia 2019j).

Let  $A$  and  $B$  be sets. A **map** or **function**  $f$  from  $A$  to  $B$  is an assignment of some element  $a \in A$  to each element  $b \in B$ . The function is denoted  $f : A \rightarrow B$  and we say that  $f$  maps each element  $a \in A$  to an element  $f(a) \in B$  called the **value** of  $a$  under  $f$ , or  $a \mapsto f(a)$ . We say that  $f$  has **domain**  $A$  and **codomain**  $B$ . The **image** of  $f$  is the subset of its codomain  $B$  that contains the values of all elements mapped by  $f$  from its domain  $A$ .

## 2.2 Logical Connectives and Quantifiers



In order to make compound propositions, we need to define logical connectives. In order to specify quantities of variables, we need to define logical quantifiers. The following is a form of **first-order logic** (Wikipedia 2019d).

### 2.2.1 Logical Connectives

A proposition can be either true  $\top$  and false  $\perp$ . When it does not contain a logical connective, it is called an **atomistic proposition**. To combine propositions into a **compound proposition**, we require **logical connectives**. They are **not** ( $\neg$ ), **and** ( $\wedge$ ), and **or** ( $\vee$ ). Table 2.1 is a **truth table** for a number of connectives.

Table 2.1: a truth table for logical connectives. The first two columns are the truth values of propositions  $p$  and  $q$ ; the rest are *outputs*.

		not	and	or	nand	nor	xor	xnor
$p$	$q$	$\neg p$	$p \wedge q$	$p \vee q$	$p \uparrow q$	$p \downarrow q$	$p \underline{\vee} q$	$p \Leftrightarrow q$
$\perp$	$\perp$	$\top$	$\perp$	$\perp$	$\top$	$\top$	$\perp$	$\top$
$\perp$	$\top$	$\top$	$\perp$	$\top$	$\top$	$\perp$	$\top$	$\perp$
$\top$	$\perp$	$\perp$	$\perp$	$\top$	$\top$	$\perp$	$\top$	$\perp$
$\top$	$\top$	$\perp$	$\top$	$\top$	$\perp$	$\perp$	$\perp$	$\top$

### 2.2.2 Quantifiers

Logical quantifiers allow us to indicate the quantity of a variable. The **universal quantifier symbol**  $\forall$  means “for all”. For instance, let  $A$  be a set; then  $\forall a \in A$  means “for all elements in  $A$ ” and gives this quantity variable  $a$ . The **existential quantifier**  $\exists$  means “there exists at least one” or “for some”. For instance, let  $A$  be a set; then  $\exists a \in A \dots$  means “there exists at least one element  $a$  in  $A$  ....”

## 2.3 Problems



**Problem 2.1** For the following, write the set described in set-builder notation.

- $A = \{2, 3, 5, 9, 17, 33, \dots\}$ .
- $B$  is the set of integers divisible by 11.
- $C = \{1/3, 1/4, 1/5, \dots\}$ .
- $D$  is the set of reals between  $-3$  and  $42$ .

**Problem 2.2** Let  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ . Prove the *Cauchy-Schwarz Inequality*

$$|\mathbf{x} \cdot \mathbf{y}| \leq \|\mathbf{x}\| \|\mathbf{y}\|. \quad (2.1)$$

Hint: you may find the geometric definition of the dot product helpful.

**Problem 2.3** Let  $\mathbf{x} \in \mathbb{R}^n$ . Prove that

$$\mathbf{x} \cdot \mathbf{x} = \|\mathbf{x}\|^2. \quad (2.2)$$

Hint: you may find the geometric definition of the dot product helpful.

**Problem 2.4** Let  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ . Prove the *Triangle Inequality*

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|. \quad (2.3)$$

Hint: you may find the Cauchy-Schwarz Inequality helpful.



# 3 Probability and Random Processes



This chapter introduces probability and random variables. Important in itself, it will also provide the basis for statistics, described in chapter 4.

## 3.1 Probability and Measurement



**Probability theory** is a well-defined branch of mathematics. Andrey Kolmogorov described a set of axioms in 1933 that are still in use today as the foundation of probability theory.<sup>1</sup>

We will implicitly use these axioms in our analysis. The **interpretation** of probability is a contentious matter. Some believe probability quantifies the frequency of the occurrence of some **event** that is repeated in a large number of trials. Others believe it quantifies the state of our knowledge or belief that some event will occur.

In experiments, our measurements are tightly coupled to probability. This is apparent in the questions we ask. Here are some examples.

1. How common is a given event?
2. What is the probability we will reject a good theory based on experimental results?
3. How repeatable are the results?
4. How confident are we in the results?
5. What is the character of the fluctuations and drift in the data?
6. How much data do we need?

1. For a good introduction to probability theory, see (Ash 2008) or (Jaynes et al. 2003).

### 3.2 Basic Probability Theory



The mathematical model for a class of measurements is called the **probability space** and is composed of a mathematical triple of a sample space  $\Omega$ ,  $\sigma$ -algebra  $\mathcal{F}$ , and probability measure  $P$ , typically denoted  $(\Omega, \mathcal{F}, P)$ , each of which we will consider in turn (Wikipedia 2019g).

The **sample space**  $\Omega$  of an experiment is the set representing all possible **outcomes** of the experiment. If a coin is flipped, the sample space is  $\Omega = \{H, T\}$ , where  $H$  is *heads* and  $T$  is *tails*. If a coin is flipped twice, the sample space could be

$$\Omega = \{HH, HT, TH, TT\}.$$

However, *the same experiment can have different sample spaces*. For instance, for two coin flips, we could also choose

$$\Omega = \{\text{the flips are the same, the flips are different}\}.$$

We base our choice of  $\Omega$  on the problem at hand.

An **event** is a subset of the sample space. That is, an event corresponds to a yes-or-no question about the experiment. For instance, event  $A$  (remember:  $A \subseteq \Omega$ ) in the coin flipping experiment (two flips) might be  $A = \{HT, TH\}$ .  $A$  is an event that corresponds to the question, “Is the second flip different than the first?”  $A$  is the event for which the answer is “yes.”

#### 3.2.1 Algebra of Events

Because events are sets, we can perform the usual set operations with them.

##### Example 3.1

Consider a toss of a single die. We choose the sample space to be  $\Omega = \{1, 2, 3, 4, 5, 6\}$ . Let the following define events.

$$A \equiv \{\text{the result is even}\} = \{2, 4, 6\}$$

$$B \equiv \{\text{the result is greater than 2}\} = \{3, 4, 5, 6\}.$$

Find the following event combinations:

$$A \cup B \quad A \cap B \quad A \setminus B \quad B \setminus A \quad \overline{A} \setminus B.$$

$$A \cup B = \{2, 3, 4, 5, 6\} \quad (\text{even or greater than 2})$$

$$A \cap B = \{4, 6\} \quad (\text{even and greater than 2})$$

$$A \setminus B = \{2\} \quad (\text{even but not greater than 2})$$

$$B \setminus A = \{3, 5\} \quad (\text{greater than two and odd})$$

$$\overline{A} \setminus B = \{1, 3, 5\} \setminus \{3, 4, 5, 6\} \quad (\text{not even and not greater than 2}).$$

The  $\sigma$ -**algebra**  $\mathcal{F}$  is the collection of events of interest. Often,  $\mathcal{F}$  is the set of all possible events given a sample space  $\Omega$ , which is just the power set of  $\Omega$  (Wikipedia 2019g). When referring to an event, we often state that it is an element of  $\mathcal{F}$ . For instance, we might say an event  $A \in \mathcal{F}$ .

We're finally ready to assign probabilities to events. We define the **probability measure**  $P : \mathcal{F} \rightarrow [0, 1]$  to be a function satisfying the following conditions.

1. For every event  $A \in \mathcal{F}$ , the probability measure of  $A$  is greater than or equal to zero—i.e.  $P(A) \geq 0$ .
2. If an event is the entire sample space, its probability measure is unity—i.e. if  $A = \Omega$ ,  $P(A) = 1$ .
3. If events  $A_1, A_2, \dots$  are disjoint sets (no elements in common), then  $P(A_1 \cup A_2 \cup \dots) = P(A_1) + P(A_2) + \dots$ .

We conclude the basics by observing four facts that can be proven from the definitions above.

1.  $P(\emptyset) = 0$ .
2.  $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ .
3. If  $B \subset A$ , then  $P(B) < P(A)$ . In fact,  $P(A \setminus B) = P(A) - P(B)$ .
4.  $P(A_1 \cup A_2 \cup \dots) \leq P(A_1) + P(A_2) + \dots$ .

### 3.3 Independence and Conditional Probability



Two events  $A$  and  $B$  are **independent** if and only if

$$P(A \cap B) = P(A)P(B).$$

If an experimenter must make a judgment without data about the independence of events, they base it on their knowledge of the events, as discussed in the following example.

#### Example 3.2

Answer the following questions and imperatives.

1. Consider a single fair die rolled twice. What is the probability that both rolls are 6?
2. What changes if the die is biased by a weight such that  $P(\{6\}) = 1/7$ ?
3. What changes if the die is biased by a magnet, rolled on a magnetic dice-rolling tray such that  $P(\{6\}) = 1/7$ ?
4. What changes if there are two dice, biased by weights such that for each  $P(\{6\}) = 1/7$ , rolled once, both resulting in 6?
5. What changes if there are two dice, biased by magnets, rolled together?

1. Let event  $A = \{6\}$ . Assuming a fair die,  $P(A) = 1/6$ . Having no reason to judge otherwise, we assume the results are independent events. Therefore,

$$P(A \cap A) = P(A)P(A) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}.$$

2. Bias is not dependence. So

$$P(A \cap A) = P(A)P(A) = \frac{1}{7} \cdot \frac{1}{7} = \frac{1}{49}.$$

3. Again, just bias, still independent.
4. Still independent.
5. The magnet dice can influence each other! This means they are not independent! If one wanted to estimate the probability, either a theoretical prediction based on the interaction would need to be developed or several trials could be conducted to obtain an estimation.

### 3.3.1 Conditional Probability

If events  $A$  and  $B$  are somehow **dependent**, we need a way to compute the probability of  $B$  occurring given that  $A$  occurs. This is called the **conditional probability** of  $B$  given  $A$ , and is denoted  $P(B | A)$ . For  $P(A) > 0$ , it is defined as

$$P(B | A) = \frac{P(A \cap B)}{P(A)}. \quad (3.1)$$

We can interpret this as a restriction of the sample space  $\Omega$  to  $A$ ; i.e. the new sample space  $\Omega' = A \subseteq \Omega$ . Note that if  $A$  and  $B$  are independent, we obtain the obvious result:

$$\begin{aligned} P(B | A) &= \frac{P(A)P(B)}{P(A)} \\ &= P(B). \end{aligned}$$

#### Example 3.3

Consider two unbiased dice rolled once. Let events  $A = \{\text{sum of faces} = 8\}$  and  $B = \{\text{faces are equal}\}$ . What is the probability the faces are equal given that their sum is 8?



Directly applying equation (3.1),

$$\begin{aligned}
 P(B | A) &= \frac{P(A \cap B)}{P(A)} \\
 &= \frac{P(\{(4, 4)\})}{P(\{(4, 4)\}) + P(\{(2, 6)\}) + P(\{(6, 2)\}) + P(\{(3, 5)\}) + P(\{(5, 3)\})} \\
 &= \frac{\frac{1}{6} \cdot \frac{1}{6}}{5 \cdot \frac{1}{6} \cdot \frac{1}{6}} \\
 &= \frac{1}{5}.
 \end{aligned}$$

We don't count the event  $\{(4, 4)\}$  twice, but we do count both  $\{(3, 5)\}$  and  $\{(5, 3)\}$ , since they are distinct events. We say "order matters" for these types of events.

### 3.4 Bayes' Theorem



Given two events  $A$  and  $B$ , **Bayes' theorem** (aka Bayes' rule) states that

$$P(A | B) = P(B | A) \frac{P(A)}{P(B)}.$$

Sometimes this is written

$$P(A | B) = \frac{P(B | A)P(A)}{P(B | A)P(A) + P(B | \neg A)P(\neg A)} \quad (3.2)$$

$$= \frac{1}{1 + \frac{P(B | \neg A)}{P(B | A)} \cdot \frac{P(\neg A)}{P(A)}}. \quad (3.3)$$

This is a useful theorem for determining a test's effectiveness. If a test is performed to determine whether an event has occurred, we might ask questions like "if the test indicates that the event has occurred, what is the probability it has actually occurred?" Bayes' theorem can help compute an answer.

#### 3.4.1 Testing Outcomes

The test can be either positive or negative, meaning it can either indicate or not indicate that  $A$  has occurred. Furthermore, this result can be either *true* 😊 or *false* ☹️.

There are four options, then. Consider an event  $A$  and an event that is a test result  $B$  indicating that event  $A$  has occurred. table 3.1 shows these four possible test outcomes. The event  $A$  occurring can lead to a true positive or a false negative, whereas  $\neg A$  can lead to a true negative or a false positive.

Table 3.1: Test outcome  $B$  for event  $A$ .

		$A$		$\neg A$	
positive ( $B$ )	■	true	😊	false	😞
negative ( $\neg B$ )	■	false	😞	true	😊

Terminology is important, here.

- $P(\{\text{true positive}\}) = P(B | A)$ , aka **sensitivity** or **detection rate**,
- $P(\{\text{true negative}\}) = P(\neg B | \neg A)$ , aka **specificity**,
- $P(\{\text{false positive}\}) = P(B | \neg A)$ ,
- $P(\{\text{false negative}\}) = P(\neg B | A)$ .

Clearly, the desirable result for any test is that it is *true*. However, no test is true 100 percent of the time. So sometimes it is desirable to err on the side of the false positive, as in the case of a medical diagnostic. Other times, it is more desirable to err on the side of a false negative, as in the case of testing for defects in manufactured balloons (when a false negative isn't a big deal).

### 3.4.2 Posterior Probabilities

Returning to Bayes' theorem, we can evaluate the **posterior probability**  $P(A | B)$  of the event  $A$  having occurred given that the test  $B$  is positive, given information that includes the **prior probability**  $P(A)$  of  $A$ . The form in equation (3.2) or equation (3.3) is typically useful because it uses commonly known test probabilities: of the true positive  $P(B | A)$  and of the false positive  $P(B | \neg A)$ . We calculate  $P(A | B)$  when we want to interpret test results.

Some interesting results can be found from this. For instance, if we let  $P(B | A) = P(\neg B | \neg A)$  (sensitivity equal specificity) and realize that  $P(B | \neg A) + P(\neg B | \neg A) = 1$  (when  $\neg A$ , either  $B$  or  $\neg B$ ), we can derive the expression

$$P(B | \neg A) = 1 - P(B | A).$$

Using this and  $P(\neg A) = 1 - P(A)$  in equation (3.3) gives (recall we've assumed sensitivity equals specificity!)

$$\begin{aligned}
 P(A | B) &= \frac{1}{1 + \frac{1 - P(B | A)}{P(B | A)} \cdot \frac{1 - P(A)}{P(A)}} \\
 &= \frac{1}{1 + \left( \frac{1}{P(B | A)} - 1 \right) \left( \frac{1}{P(A)} - 1 \right)}
 \end{aligned}$$

This expression is plotted in figure 3.1. See that a positive result for a rare event (small  $P(A)$ ) is hard to trust unless the sensitivity  $P(B | A)$  and specificity  $P(\neg B | \neg A)$  are very high, indeed!

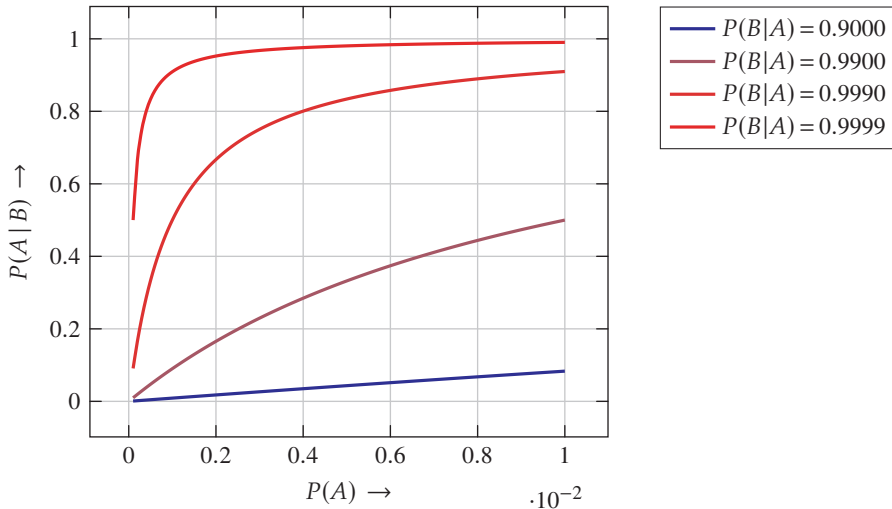


Figure 3.1. For different high-sensitivities, the probability that an event  $A$  occurred given that a test for it  $B$  is positive versus the probability that the event  $A$  occurs, under the assumption the specificity equals the sensitivity.

### Example 3.4

Suppose 0.1 percent of springs manufactured at a given plant are defective. Suppose you need to design a test that, when it indicates a defective part, the part is actually defective 99 percent of the time. What sensitivity should your test have assuming it can be made equal to its specificity?

We proceed in Python.

```
from sympy import * # for symbolics
import numpy as np # for numerics
import matplotlib.pyplot as plt # for plots

Define symbolic variables.
var('p_A,p_nA,p_B,p_nB,p_B_A,p_B_nA,p_A_B',real=True)
(p_A, p_nA, p_B, p_nB, p_B_A, p_B_nA, p_A_B)
```

Beginning with Bayes' theorem and assuming the sensitivity and specificity are equal by section 3.4.2, we can derive the following expression for the posterior probability  $P(A | B)$ .

```
p_A_B_e1 = Eq(p_A_B, p_B_A * p_A / p_B).subs(
    {
        p_B: p_B_A * p_A + p_B_nA * p_nA, # conditional prob
        p_B_nA: 1 - p_B_A, # Eq (3.5)
        p_nA: 1 - p_A
    }
)
print(p_A_B_e1)
```

$$p_{AB} = \frac{p_A p_{BA}}{p_A p_{BA} + (1 - p_A)(1 - p_{BA})}$$

Solve this for  $P(B | A)$ , the quantity we seek.

```
p_B_A_sol = solve(p_A_B_e1, p_B_A, dict=True)
p_B_A_eq1 = Eq(p_B_A, p_B_A_sol[0][p_B_A])
print(p_B_A_eq1)
```

$$p_{BA} = \frac{p_{AB}(1 - p_A)}{-2p_A p_{AB} + p_A + p_{AB}}$$

Now let's substitute the given probabilities.

```
p_B_A_spec = p_B_A_eq1.subs(
    {
        p_A: 0.001,
        p_A_B: 0.99,
    }
)
print(p_B_A_spec)
```

$$p_{BA} = 0.999989888981011$$

That's a tall order!

### 3.5 Random Variables



Probabilities are useful even when they do not deal strictly with events. It often occurs that we measure something that has randomness associated with it. We use random variables to represent these measurements.

A **random variable**  $X : \Omega \rightarrow \mathbb{R}$  is a function that maps an outcome  $\omega$  from the sample space  $\Omega$  to a real number  $x \in \mathbb{R}$ , as shown in figure 3.2. A random variable will be denoted with a capital letter (e.g.  $X$  and  $K$ ) and a specific value that it maps to (the value) will be denoted with a lowercase letter (e.g.  $x$  and  $k$ ).

A **discrete random variable**  $K$  is one that takes on discrete values. A **continuous random variable**  $X$  is one that takes on continuous values.

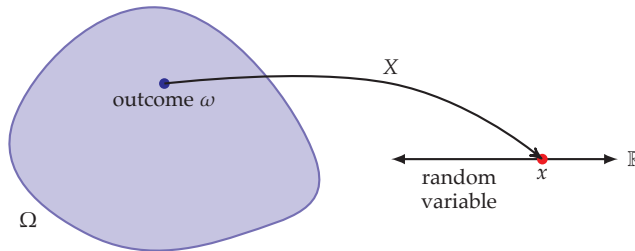


Figure 3.2. A random variable  $X$  maps an outcome  $\omega \in \Omega$  to an  $x \in \mathbb{R}$ .

#### Example 3.5

Roll two unbiased dice. Let  $K$  be a random variable representing the sum of the two. Let  $P(k)$  be the probability of the result  $k \in K$ . Plot and interpret  $P(k)$ .

Figure 3.3 shows the probability of each sum occurring.

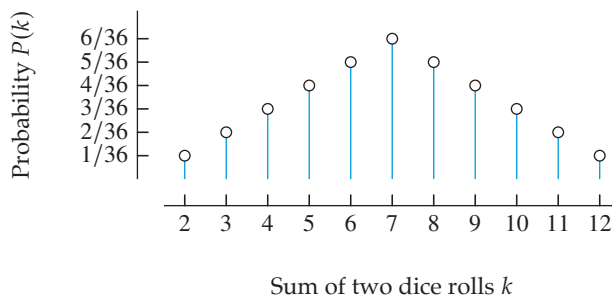


Figure 3.3. PMF for the summ of two dice rolled.

We call this a **probability mass function**. It tells us the probability with which each outcome will occur.

### Example 3.6

A resistor at nonzero temperature without any applied voltage exhibits an interesting phenomenon: its voltage randomly fluctuates. This is called *Johnson-Nyquist noise* and is a result of *thermal excitation* of charge carriers (electrons, typically). For a given resistor and measurement system, let the *probability density function*  $f_V$  of the voltage  $V$  across an unrealistically hot resistor be

$$f_V(V) = \frac{1}{\sqrt{\pi}} e^{-V^2}.$$

Plot and interpret the meaning of this function.

The PDF is shown in figure 3.4.

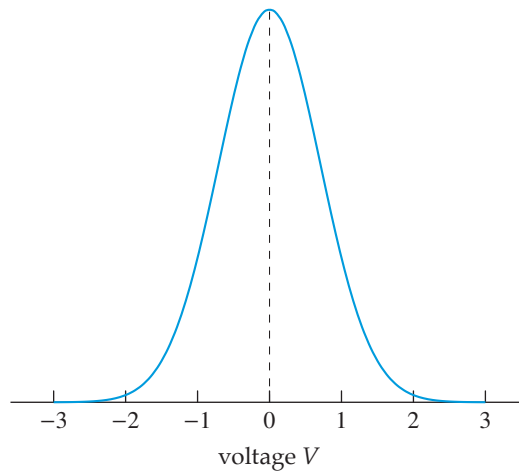


Figure 3.4. The probability density function.

A probability density function must be integrated to find probability. The probability a randomly measured voltage will be between two voltages is the integral of  $f_V$  across that voltage interval. Note that a resistor would need to be extremely hot to have such a large thermal noise. In the next lecture, we consider more probability density functions.

### 3.6 Probability Density and Mass Functions



Consider an experiment that measures a random variable. We can plot the relative frequency of the measurand landing in different “bins” (ranges of values). This is called a **frequency distribution** or a **probability mass function** (PMF).

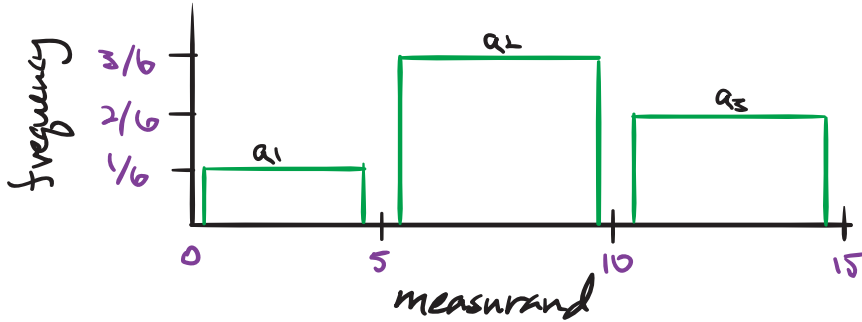


Figure 3.5. Plot of a probability mass function.

Consider, for instance, a probability mass function as plotted in figure 3.5, where a frequency  $a_i$  can be interpreted as an estimate of the probability of the measurand being in the  $i$ th interval. The sum of the frequencies must be unity:

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

with  $k$  being the number of bins.

The **frequency density distribution** is similar to the frequency distribution, but with  $a_i \mapsto a_i/\Delta x$ , where  $\Delta x$  is the bin width.

If we let the bin width approach zero, we derive the **probability density function** (PDF)

$$f(x) = \lim_{\substack{k \rightarrow \infty \\ \Delta x \rightarrow 0}} \sum_{j=1}^k a_j / \Delta x.$$

We typically think of a probability density function  $f$ , like the one in figure 3.6 as a function that can be integrated over to find the probability of the random variable (measurand) being in an interval  $[a, b]$ :

$$P(x \in [a, b]) = \int_a^b f(x) dx.$$

Of course,

$$\begin{aligned} P(x \in (-\infty, \infty)) &= \int_{-\infty}^{\infty} f(x) dx \\ &= 1. \end{aligned}$$

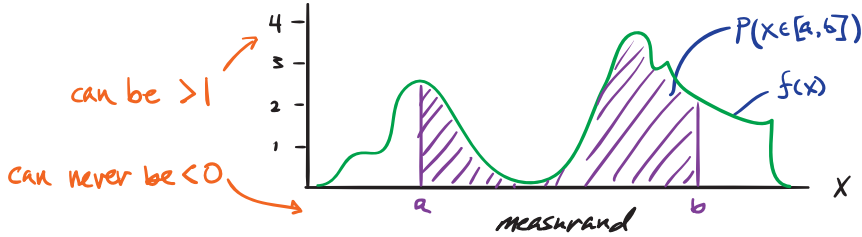


Figure 3.6. Plot of a probability density function.

We now consider a common PMF and a common PDF.

### 3.6.1 Binomial PMF

Consider a random binary sequence of length  $n$  such that each element is a random 0 or 1, generated independently, like

$$(1, 0, 1, 1, 0, \dots, 1, 1).$$

Let events  $\{1\}$  and  $\{0\}$  be mutually exclusive and exhaustive and  $P(\{1\}) = p$ . The probability of the sequence above occurring is

$$P((1, 0, 1, 1, 0, \dots, 1, 1)) = p(1-p)pp(1-p) \cdots pp.$$

There are  $n$  choose  $k$ ,

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

possible combinations of  $k$  ones for  $n$  bits. Therefore, the probability of any combination of  $k$  ones in a series is

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

We call section 3.6.1 the **binomial distribution PDF**.

#### Example 3.7

Consider a field sensor that fails for a given measurement with probability  $p$ . Given  $n$  measurements, plot the binomial PMF as a function of  $k$  failed measurements for a few different probabilities of failure  $p \in [0.04, 0.25, 0.5, 0.75, 0.96]$ .



We proceed in Python.

Import the necessary libraries.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb
```

Define the parameters.

```
n = 100
k_a = np.arange(1, n + 1)
p_a = np.array([0.04, 0.25, 0.5, 0.75, 0.96])
```

Define the binomial distribution function.

```
def binomial(n, k, p):
    return comb(n, k) * (p ** k) * ((1 - p) ** (n - k))
```

Construct the binomial distribution array.

```
f_a = np.zeros((len(k_a), len(p_a)))
for i in range(len(k_a)):
    for j in range(len(p_a)):
        f_a[i, j] = binomial(n, k_a[i], p_a[j])
```

Plot the binomial distribution.

```
colors = plt.cm.jet(np.linspace(0, 1, len(p_a)))
fig, ax = plt.subplots()
for j in range(len(p_a)):
    plt.bar(k_a, f_a[:, j], color=colors[j], alpha=0.5, label=f'$p = {p_a[j]}$')
plt.legend(loc='best', frameon=False, fontsize='medium')
plt.xlabel('Number of ones in sequence k')
plt.ylabel('Probability')
plt.xlim([0, 100])
plt.show()
```

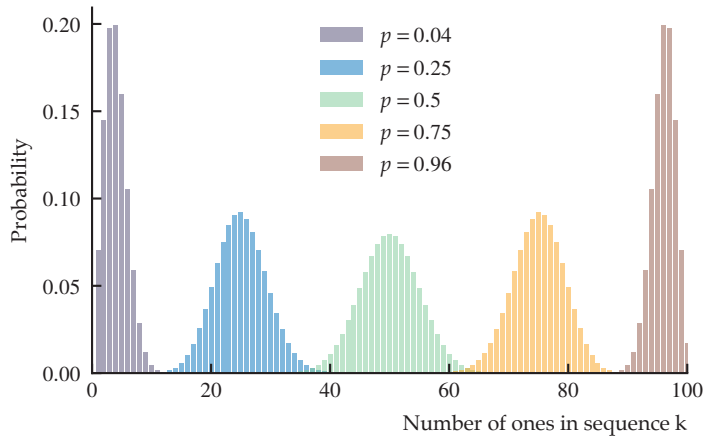


Figure 3.7. Binomial PDF for  $n = 100$  measurements and different values of  $P(\{1\}) = p$ , the probability of a measurement error.

Note that the symmetry is due to the fact that events  $\{1\}$  and  $\{0\}$  are mutually exclusive and exhaustive.

### 3.6.2 Gaussian PDF

The **Gaussian** or *normal random variable*  $x$  has PDF

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(x - \mu)^2}{2\sigma^2}.$$

Although we're not quite ready to understand these quantities in detail, it can be shown that the parameters  $\mu$  and  $\sigma$  have the following meanings:

- $\mu$  is the **mean** of  $x$ ,
- $\sigma$  is the **standard deviation** of  $x$ , and
- $\sigma^2$  is the **variance** of  $x$ .

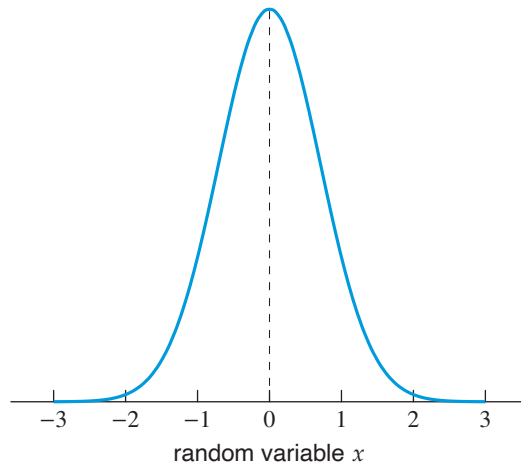


Figure 3.8. PDF for Gaussian random variable  $x$ , mean  $\mu = 0$ , and standard deviation  $\sigma = 1/\sqrt{2}$ .

Consider the “bell-shaped” Gaussian PDF in figure 3.8. It is always symmetric. The mean  $\mu$  is its central value and the standard deviation  $\sigma$  is directly related to its width. We will continue to explore the Gaussian distribution in the following lectures, especially in section 4.3.

### 3.7 Expectation

Recall that a random variable is a function  $X : \Omega \rightarrow \mathbb{R}$  that maps from the sample space to the reals. Random variables are the arguments of probability mass functions (PMFs) and probability density functions (PDFs).

The **expected value** (or **expectation**) of a random variable is akin to its “average value” and depends on its PMF or PDF. The expected value of a random variable  $X$  is denoted  $\langle X \rangle$  or  $E[X]$ . There are two definitions of the expectation, one for a discrete random variable, the other for a continuous random variable. Before we define, them, however, it is useful to predefine the most fundamental property of a random variable, its **mean**.

#### Definition 3.1

The *mean* of a random variable  $X$  is defined as

$$m_X = E[X].$$

Let’s begin with a discrete random variable.



**Definition 3.2**

Let  $K$  be a discrete random variable and  $f$  its PMF. The *expected value* of  $K$  is defined as

$$E[K] = \sum_{\forall k} k f(k).$$

**Example 3.8**

Given a discrete random variable  $K$  with PMF shown below, what is its mean  $m_K$ ?

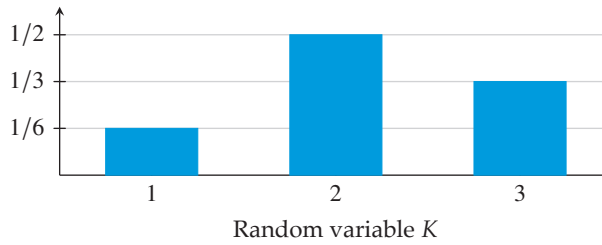


Figure 3.9. PMF of discrete random variable  $K$ .

Compute from the definitions:

$$\begin{aligned} \mu_K &= E[K] \\ &= \sum_{i=1}^3 k_i f(k_i) \\ &= 1 \cdot \frac{1}{6} + 2 \cdot \frac{3}{6} + 3 \cdot \frac{2}{6} \\ &= \frac{13}{6}. \end{aligned}$$

Let us now turn to the expectation of a continuous random variable.

**Definition 3.3**

Let  $X$  be a continuous random variable and  $f$  its PDF. The *expected value* of  $X$  is defined as

$$E[X] = \int_{-\infty}^{\infty} x f(x) dx.$$

### Example 3.9

Given a continuous random variable  $X$  with Gaussian PDF  $f$ , what is the expected value of  $X$ ?

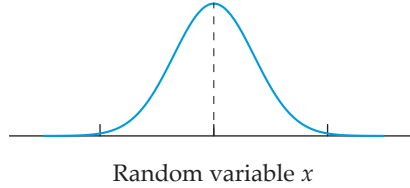


Figure 3.10. Gaussian PDF for random variable  $X$ .

Compute from the definition:

$$\begin{aligned} E[X] &= \int_{-\infty}^{\infty} x f(x) dx \\ &= \int_{-\infty}^{\infty} x \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(x-\mu)^2}{2\sigma^2} dx. \end{aligned}$$

Substitute  $z = x - \mu$ :

$$\begin{aligned} E[X] &= \int_{-\infty}^{\infty} (z + \mu) \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-z^2}{2\sigma^2} dz \\ &= \mu \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-z^2}{2\sigma^2} dz + \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\infty} z \exp \frac{-z^2}{2\sigma^2} dz. \end{aligned}$$

The first integrand is a Gaussian PDF with its  $\mu = 0$ , so, by definition, the first integral is 1. The second integrand is an *odd* function, so its improper integral over all  $z$  is 0. This leaves

$$E[X] = \mu.$$

Due to its sum or integral form, the expected value  $E[\cdot]$  has some familiar properties for random variables  $X$  and  $Y$  and reals  $a$  and  $b$ .

$$E[a] = a \tag{3.4}$$

$$E[X + a] = E[X] + a \tag{3.5}$$

$$E[aX] = a E[X] \tag{3.6}$$

$$E[E[X]] = E[X] \tag{3.7}$$

$$E[aX + bY] = a E[X] + b E[Y]. \tag{3.8}$$

### 3.8 Central Moments



Given a probability mass function (PMF) or probability density function (PDF) of a random variable, several useful parameters of the random variable can be computed. These are called **central moments**, which quantify parameters relative to its mean.

#### Definition 3.4

The  $n$ th central moment of random variable  $X$ , with PDF  $f$ , is defined as

$$E[(X - \mu_X)^n] = \int_{-\infty}^{\infty} (x - \mu_X)^n f(x) dx.$$

For discrete random variable  $K$  with PMF  $f$ ,

$$E[(K - \mu_K)^n] = \sum_{\forall k} (k - \mu_K)^n f(k).$$

#### Example 3.10

Prove that the first moment of continuous random variable  $X$  is zero.

From the definition of the first moment:

$$E[(X - \mu_X)^1] = \int_{-\infty}^{\infty} (x - \mu_X)^1 f(x) dx \quad (3.9)$$

$$= \int_{-\infty}^{\infty} x f(x) dx - \mu_X \int_{-\infty}^{\infty} f(x) dx \quad (\text{split})$$

$$= \mu_X - \mu_X \cdot 1 \quad (\text{defs. of } \mu_X \text{ and PDF})$$

$$= 0. \quad (3.10)$$

The second central moment of random variable  $X$  is called the **variance** and is denoted

$$\sigma_X^2 \quad \text{or} \quad \text{Var}[X] \quad \text{or} \quad E[(X - \mu_X)^2].$$

The variance is a measure of the *width* or *spread* of the PMF or PDF. We usually compute the variance with the formula

$$\text{Var}[X] = E[X^2] - \mu_X^2.$$

Other properties of variance include, for real constant  $c$ ,

$$\text{Var}[c] = 0$$

$$\text{Var}[X + c] = \text{Var}[X]$$

$$\text{Var}[cX] = c^2 \text{Var}[X].$$

The **standard deviation** is defined as

$$\sigma_X = \sqrt{\sigma_X^2}.$$

Although the variance is mathematically more convenient, the standard deviation has the same physical units as  $X$ , so it is often the more physically meaningful quantity. Due to its meaning as the width or spread of the probability distribution, and its sharing of physical units, it is a convenient choice for error bars on plots of a random variable.

The **skewness**  $\text{Skew}[X]$  is a normalized third central moment:

$$\text{Skew}[X] = \frac{\text{E}[(X - \mu_X)^3]}{\sigma_X^3}.$$

Skewness is a measure of **asymmetry** of a random variable's PDF or PMF. For a symmetric PMF or PDF, such as the Gaussian PDF,  $\text{Skew}[X] = 0$ .

The **kurtosis**  $\text{Kurt}[X]$  is a normalized fourth central moment:

$$\text{Kurt}[X] = \frac{\text{E}[(X - \mu_X)^4]}{\sigma_X^4}.$$

Kurtosis is a measure of the **tailedness** of a random variable's PDF or PMF. "Heavier" tails yield higher kurtosis.

A Gaussian random variable has PDF with kurtosis 3. Given that for Gaussians both skewness and kurtosis have nice values (0 and 3), we can think of skewness and kurtosis as measures of similarity to the Gaussian PDF.

### 3.9 Transforming Random Variables

TODO: describe the theory and formulae

For random variables  $X$  and  $Y$  with PDFs  $f_X$  and  $f_Y$ , and with invertible transformation  $Y = g(X)$ , we have the linear approximation

$$f_Y(y) = \frac{1}{|dy/dx|} f_X(x) \Big|_{x \mapsto g^{-1}(y)}. \quad (3.11)$$



### Example 3.11

Suppose we are to probabilistically quantify a parachutist's chances of landing within a certain horizontal distance of a landing target, accounting for random wind displacements. Develop a PDF for the random variable  $R$ , the landing distance from the target.

Without much intuition, no data, or a very good physical model of the situation, we are left to bootstrap a solution. A toehold can perhaps be found by narrowing the problem to a drop of a fixed, relatively short distance, such as that shown in figure 3.11.

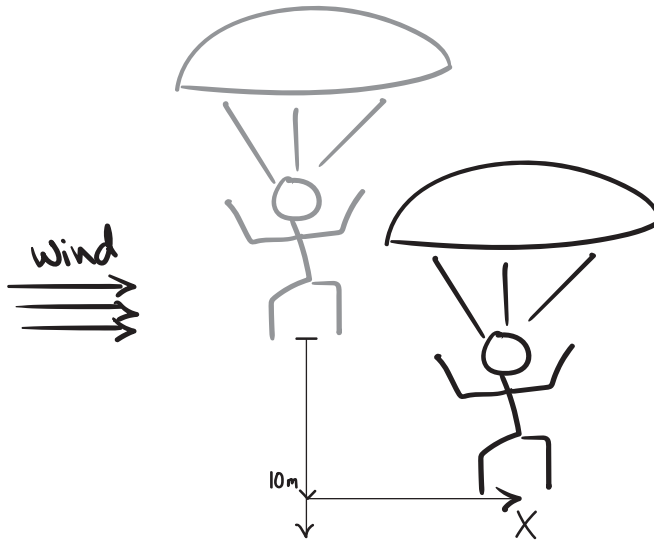


Figure 3.11. A parachutist falling 10 m and being displaced by wind an amount modeled by random variable  $X$ .

For each vertical drop of 10 m, we might expect a horizontal displacement of a few meters. Without any information about average prevailing winds, we cannot expect any particular direction to be most likely. It seems more likely that wind gusts would displace the parachutist a small amount than a large amount, and even less likely to displace a very large amount. These facts suggest a reasonable model to start with is a Gaussian distribution with PDF

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{-(x - \mu)^2}{2\sigma^2},$$



where  $\mu = 0$  m and  $\sigma = 5$  m. This model could clearly be improved with some data or a detailed analysis of the physics involved, but this seems to be a reasonable place to begin.

From here, we can extrapolate. For one 10-m drop, the displacement random variable is  $X$ . For two 10-m drops, the displacement random variable is  $2X$ , and so on. We conclude that for  $N$  drops of 10 m, the landing displacement random variable  $R$  is

$$R = NX.$$

Here we have assumed the parachutist lands after  $N$  drops of 10 m. Another way of writing this is

$$R = h(X) = NX.$$

The function  $h$  transforms random variable  $X$  (with value  $x$ ) to random variable  $R$  with value  $(r)$ .

We can apply equation (3.11) directly to find the PDF of  $R$  as follows:

$$f_Y(y) = \frac{1}{|dr/dx|} f_X(x) \Big|_{x \mapsto h^{-1}(r)} \quad (3.12)$$

$$= \frac{1}{N} \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{-(r/N - \mu)^2}{2\sigma^2}. \quad (3.13)$$

Letting  $\mu' = N\mu$  and  $\sigma' = N\sigma$ , we obtain

$$f_R(r) = \frac{1}{\sqrt{2\pi}\sigma'} \exp \frac{-(x - \mu')^2}{2\sigma'^2}.$$

That is,  $R$  also has a Gaussian PDF. We see that the linear transformation has simply transformed the mean  $\mu$  and standard deviation  $\sigma$  accordingly.

We observe that for greater  $N$  (higher jumps), the standard deviation is also greater. This is an intuitive result. We now turn to Python for graphical and simulation purposes.

Load the necessary packages:

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
```

Define fixed parameters:

```
mu = 0.0 # Mean of the Gaussian distribution for the 10 m drop
sigma = 5.0 # Standard deviation of the Gaussian distribution for the 10 m drop
```

Define the 10 m drop Gaussian distribution  $f_X(x)$  symbolically

```
x, r, N = sp.symbols('x, r, N', real=True)
f_X = 1/(sigma * sp.sqrt(2 * sp.pi)) * sp.exp(-(x - mu)**2 / (2 * sigma**2))
print(f_X)
```

```
| 0.1*sqrt(2)*exp(-0.02*x**2)/sqrt(pi)
```

Define the functional relationship between  $X$  and  $R$ , the horizontal distance from the initial drop point

```
h_eq = sp.Eq(r, N * x)
h_sol = sp.solve(h_eq, r, dict=True)[0]
h_inv = sp.solve(h_eq, x, dict=True)[0]
dr_dx = sp.diff(h_sol[r], x)
print(dr_dx)
```

```
| N
```

Define symbolically  $f_R(r)$ , the probability density function for the horizontal distance from the initial drop point:

```
f_R = 1/sp.Abs(dr_dx) * f_X.subs(h_inv)
print(f_R)
```

```
| 0.1*sqrt(2)*exp(-0.02*r**2/N**2)/(sqrt(pi)*Abs(N))
```

Lambdify the PDF for numerical evaluation

```
f_R_fun = sp.lambdify((r, N), f_R, 'numpy')
```

Plot the PDF  $f_R(r)$  for several values of  $N$ :

```
N_vals = np.array([600, 800, 1000])/10 # Drop steps of 10 m
r_vals = np.linspace(-1000, 1000, 1001)
fig, ax = plt.subplots()
for N_val in N_vals:
    p_vals = f_R_fun(r_vals, N_val)
    ax.plot(r_vals, p_vals, label=f'N = {N_val}')
ax.set_xlabel('$r_f$ (m)')
ax.set_ylabel('$p(r_f)$')
ax.legend()
plt.draw()
```

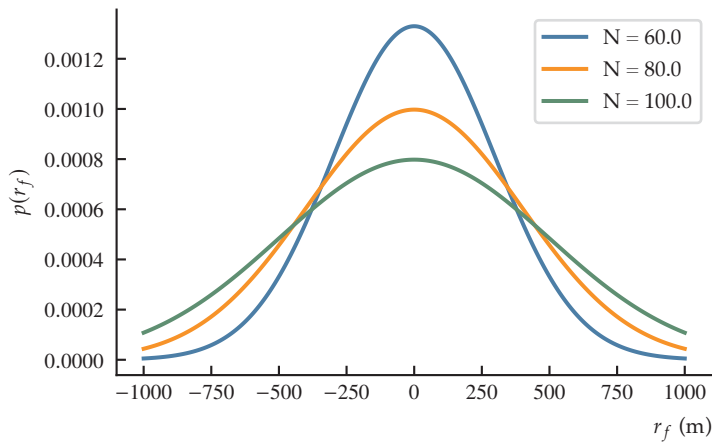


Figure 3.12. Probability density function  $f_R(r)$  for several values of  $N$

Compute the probability of landing within  $\pm 500$  m of the initial drop point:

```
r_min, r_max = -500, 500
p_landing = sp.integrate(f_R, (r, r_min, r_max))
print(p_landing)

Piecewise((0.707106781186548*sqrt(2)*Abs(N)*erf(70.7106781186548/Abs(N))/N,
↪ N >= 0), (-
↪ 0.707106781186548*sqrt(2)*Abs(N)*erf(70.7106781186548/Abs(N))/N,
↪ True))
```

Plot the probability of landing within  $\pm 500$  m of the initial drop point as a function of  $N$ :

```
N_vals = np.linspace(1, 1200, 1001)
p_landing_fun = sp.lambdify(N, p_landing, 'numpy')
p_landing_vals = np.zeros(N_vals.shape[0]) # Preallocate
for i, N_val in enumerate(N_vals):
    p_landing_vals[i] = p_landing_fun(N_val) # Evaluate
fig, ax = plt.subplots()
ax.plot(N_vals * 10, p_landing_vals)
ax.set_xlabel('Drop height (m)')
ax.set_ylabel('$p(\pm 500 \text{ m})$')
plt.draw()
```

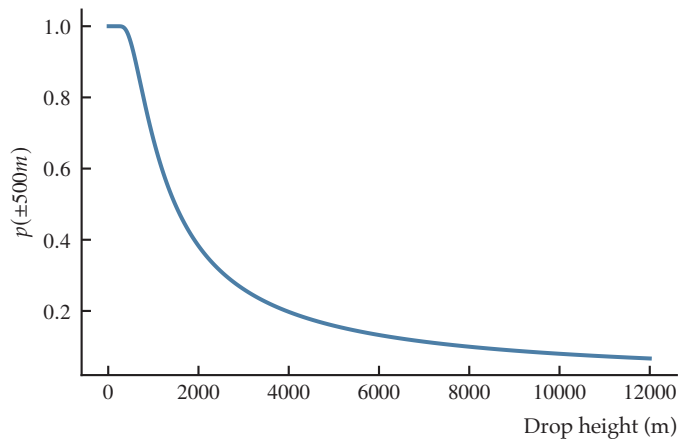


Figure 3.13. Probability of landing within  $\pm 500$  m of the initial drop point as a function of  $N$

Define a function to take one 10 m drop:

```
def take_drop(x_previous):
    x_new = x_previous + np.random.normal(mu, sigma)
    return x_new
```

Define a function to simulate a random walk:

```
def simulate_random_walk(N_sim):
    y_sim = np.flip(np.arange(0, N_sim + 1)) * 10 # Heights
    x_sim = np.zeros(N_sim + 1) # Preallocate
    x_sim[0] = 0 # Initial drop point
    for i in range(1, N_sim + 1):
        x_sim[i] = take_drop(x_sim[i - 1])
    return x_sim, y_sim
```

Simulate several random walks (drops) for various values of  $N$ :

```
N_vals = [60, 80, 100]
n_sim = 50 # Number of simulations
x_sims = [np.zeros(n_sim, N_val+1) for N_val in N_vals] # Preallocate
y_sims = [np.zeros(n_sim, N_val+1) for N_val in N_vals] # Preallocate
for i, N_val in enumerate(N_vals):
    for j in range(n_sim):
        x_sim, y_sim = simulate_random_walk(N_val)
        x_sims[i][j] = x_sim
        y_sims[i][j] = y_sim
```

Plot the random walks (drops) for several values of  $N$ :

```
fig, ax = plt.subplots()
for i, N_val in enumerate(N_vals):
    for j in range(n_sim):
        ax.plot(
            x_sims[i][j], y_sims[i][j],
            color=f'C{i}', alpha=[0.7, 0.5, 0.3][i]
        )
ax.set_xlabel('Horizontal distance (m)')
ax.set_ylabel('Height (m)')
plt.show()
```

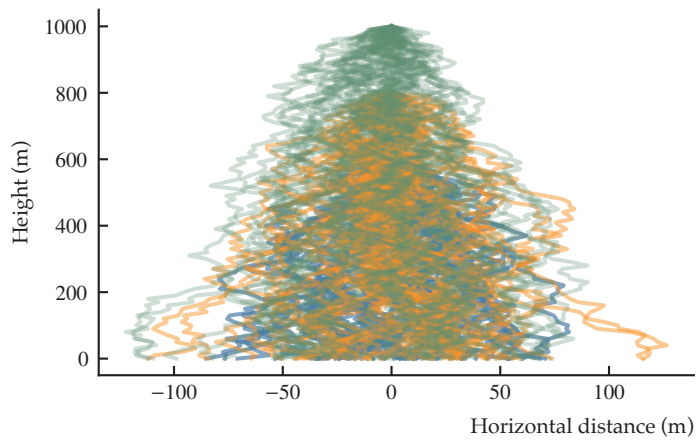


Figure 3.14. Random walks (drops) for several values of  $N$

### 3.10 Multivariate Probability and Correlation



Thus far, we have considered probability density and mass functions (PDFs and PMFs) of only one random variable. But, of course, often we measure multiple random variables  $X_1, X_2, \dots, X_n$  during a single event, meaning a measurement consists of determining values  $x_1, x_2, \dots, x_n$  of these random variables.

We can consider an  $n$ -tuple of continuous random variables to form a sample space  $\Omega = \mathbb{R}^n$  on which a multivariate function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ , called the **joint PDF** assigns a probability density to each outcome  $x \in \mathbb{R}^n$ . The joint PDF must be greater than or equal to zero for all  $x \in \mathbb{R}^n$ , the multiple integral over  $\Omega$  must be unity, and the multiple integral over a subset of the sample space  $A \subset \Omega$  is the probability of the event  $A$ .

We can consider an  $n$ -tuple of discrete random variables to form a sample space  $\mathbb{N}_0^n$  on which a multivariate function  $f: \mathbb{N}_0^n \rightarrow \mathbb{R}$ , called the **joint PMF** assigns a probability to each outcome  $x \in \mathbb{N}_0^n$ . The joint PMF must be greater than or equal to zero for all  $x \in \mathbb{N}_0^n$ , the multiple sum over  $\Omega$  must be unity, and the multiple sum over a subset of the sample space  $A \subset \Omega$  is the probability of the event  $A$ .

#### Example 3.12

Let's visualize multivariate PDFs by plotting a bivariate gaussian using the `scipy.stats` function `multivariate_normal`

We proceed in Python. First, load packages:

```
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

Define the mean and covariance matrix for a bivariate Gaussian distribution.

```
mu = [10, 20] # Mean
Sigma = [[1, 0], [0, 0.2]] # Covariance matrix
```

Generate grid points as input for the PDF.

```
x1_a = np.linspace(mu[0] - 5 * np.sqrt(Sigma[0][0]), mu[0] + 5 * np.sqrt(Sigma[0][0]))
x2_a = np.linspace(mu[1] - 5 * np.sqrt(Sigma[1][1]), mu[1] + 5 * np.sqrt(Sigma[1][1]))
```

Create a meshgrid.

```
X1, X2 = np.meshgrid(x1_a, x2_a)
```

Calculate the PDF.

```
pos = np.dstack((X1, X2))
rv = multivariate_normal(mu, Sigma)
f = rv.pdf(pos)
```

Plot the PDF.

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
p = ax.plot_surface(X1, X2, f, cmap='copper')
ax.set_xlabel(r'$x_1$', fontsize=12)
ax.set_ylabel(r'$x_2$', fontsize=12)
ax.set_zlabel(r'$f(x_1, x_2)$', fontsize=12)
plt.show()
```

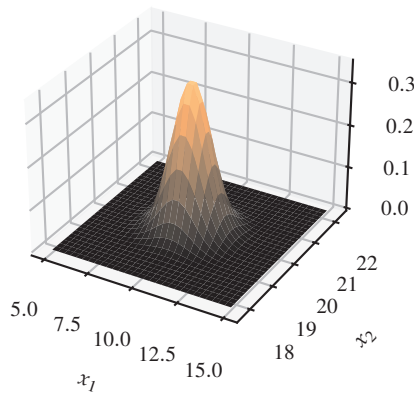


Figure 3.15. Bivariate Gaussian PDF.

The plot shows the PDF of a bivariate Gaussian distribution. Pretty neat, right?

### 3.10.1 Marginal Probability

The **marginal PDF** of a multivariate PDF is the PDF of some subspace of  $\Omega$  after one or more variables have been “integrated out,” such that a fewer number of random variables remain. Of course, these marginal PDFs must have the same properties of any PDF, such as integrating to unity.

#### Example 3.13

Let’s demonstrate this by numerically integrating over  $x_2$  in the bivariate Gaussian, above.

Continuing from where we left off, let’s integrate.

```
f1 = np.trapz(f.T, x2_a, axis=1) # Trapezoidal integration
```

Let’s plot the marginal PDF.

```
fig, ax = plt.subplots()
ax.plot(x1_a, f1, linewidth=2)
ax.set_xlabel(r'$x_1$')
ax.set_ylabel(r'$g(x_1)=\int_{-\infty}^{\infty} f(x_1,x_2) dx_2$')
plt.show()
↳ [matplotlib.lines.Line2D at 0x1680f0bd0>]
↳ Text(1, 0, '$x_1$')
↳ Text(0, 0.5, '$g(x_1)=\int_{-\infty}^{\infty} f(x_1,x_2) dx_2$')
```

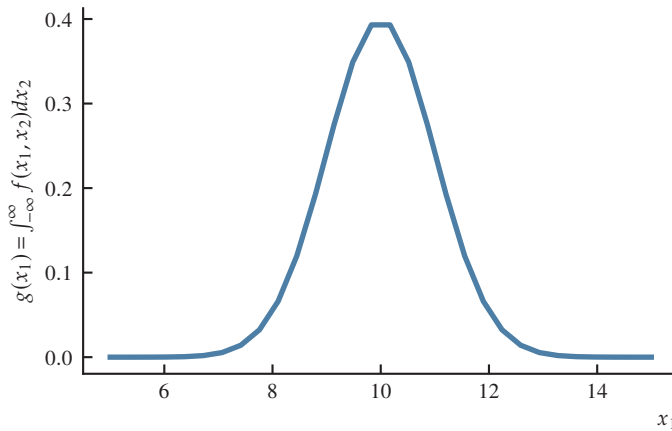


Figure 3.16. Marginal PDF of a bivariate Gaussian distribution.

We should probably verify that this integrates to one.

```
integral_value = np.trapz(f1, x1_a)
print(f'integral over x_1 = {integral_value:.7f}')
| integral over x_1 = 0.9999986
```

Not bad.



### 3.10.2 Covariance

Very often, especially in machine learning applications, the question about two random variables  $X$  and  $Y$  is: how do they co-vary? That is what is their **covariance**, defined as

$$\begin{aligned}\text{Cov}[X, Y] &\equiv E((X - \mu_X)(Y - \mu_Y)) \\ &= E(XY) - \mu_X \mu_Y.\end{aligned}$$

Note that when  $X = Y$ , the covariance is just the variance. When a covariance is large and positive, it is an indication that the random variables are *strongly correlated*. When it is large and negative, they are *strongly anti-correlated*. Zero covariance means the variables are *uncorrelated*. In fact, **correlation** is defined as

$$\text{Cor}[X, Y] = \frac{\text{Cov}[X, Y]}{\sqrt{\text{Var}[X] \text{Var}[Y]}}.$$

This is essentially the covariance “normalized” to the interval  $[-1, 1]$ .

**3.10.2.1 Sample Covariance** As with the other statistics we’ve considered, covariance can be estimated from measurement. The estimate, called the **sample covariance**  $q_{XY}$ , of random variables  $X$  and  $Y$  with sample size  $N$  is given by

$$q_{XY} = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{X})(y_i - \bar{Y}).$$

**3.10.2.2 Multivariate Covariance** With  $n$  random variables  $X_i$ , one can compute the covariance of each pair. It is common practice to define an  $n \times n$  matrix of covariances called the **covariance matrix**  $\Sigma$  such that each pair’s covariance

$$\text{Cov}[X_i, X_j]$$

appears in its row-column combination (making it symmetric), as shown below.

$$\Sigma = \begin{bmatrix} \text{Cov}[X_1, X_1] & \text{Cov}[X_1, X_2] & \cdots & \text{Cov}[X_1, X_n] \\ \text{Cov}[X_2, X_1] & \text{Cov}[X_2, X_2] & & \text{Cov}[X_2, X_n] \\ \vdots & & \ddots & \vdots \\ \text{Cov}[X_n, X_1] & \text{Cov}[X_n, X_2] & \cdots & \text{Cov}[X_n, X_n] \end{bmatrix}$$

The multivariate **sample covariance matrix**  $Q$  is the same as above, but with sample covariances  $q_{X_i X_j}$ .

Both covariance matrices have correlation analogs.

### Example 3.14

Let's use a dataset from the Scikit-Learn package with multivariate data on the attributes of wine. Compute the sample covariance and correlation matrices. Plot variables pairwise and color them with the corresponding correlation.

Load the necessary libraries.

```
import numpy as np
from sklearn.datasets import load_wine
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize
import matplotlib.cm as cm
```

Load the dataset and print the feature names.

```
data = load_wine()
print(f"Features: {data.feature_names}")

Features: ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash',
↪ 'magnesium', 'total_phenols', 'flavanoids',
↪ 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity',
↪ 'hue', 'od280/od315_of_diluted_wines', 'proline']
```

Select a list of features to analyze and select the corresponding data.

```
features = [
    'alcohol', 'malic_acid', 'ash', 'magnesium',
    'total_phenols', 'flavanoids'
]
X = data.data[
    :, [data.feature_names.index(f) for f in features]
]
```

Compute the sample covariance and correlation matrices.

```
cov = np.cov(X.T) # Covariance matrix
cor = np.corrcoef(X.T) # Correlation matrix (normalized covariance)
print(f"Covariance matrix:\n{cov}")
print(f"Correlation matrix:\n{cor}")
```

Covariance matrix:

```
[[ 6.59062328e-01  8.56113090e-02  4.71151590e-02  3.13987812e+00
   1.46887218e-01  1.92033222e-01]
 [ 8.56113090e-02  1.24801540e+00  5.02770393e-02 -8.70779534e-01
  -2.34337723e-01 -4.58630366e-01]
 [ 4.71151590e-02  5.02770393e-02  7.52646353e-02  1.12293658e+00
  2.21455913e-02  3.15347299e-02]
 [ 3.13987812e+00 -8.70779534e-01  1.12293658e+00  2.03989335e+02
  1.91646988e+00  2.79308703e+00]
 [ 1.46887218e-01 -2.34337723e-01  2.21455913e-02  1.91646988e+00
  3.91689535e-01  5.40470422e-01]
 [ 1.92033222e-01 -4.58630366e-01  3.15347299e-02  2.79308703e+00
  5.40470422e-01  9.97718673e-01]]
```

Correlation matrix:

```
[[ 1.          0.09439694  0.2115446  0.27079823  0.28910112
  ↪ 0.23681493]
 [ 0.09439694  1.          0.16404547 -0.0545751 -0.335167
  ↪ -0.41100659]
 [ 0.2115446  0.16404547  1.          0.28658669  0.12897954
  ↪ 0.11507728]
 [ 0.27079823 -0.0545751  0.28658669  1.          0.21440123
  ↪ 0.19578377]
 [ 0.28910112 -0.335167  0.12897954  0.21440123  1.
  ↪ 0.8645635 ]
 [ 0.23681493 -0.41100659  0.11507728  0.19578377  0.8645635  1.
  ↪ ]]
```

Plot the data pairings with color corresponding to the correlation matrix.

```

fig, ax = plt.subplots(cor.shape[0], cor.shape[1], figsize=(10, 10))
norm = Normalize(vmin=-1, vmax=1)
cmap = cm.coolwarm
scatter = np.empty(cor.shape, dtype=object)
for i in range(cor.shape[0]):
    for j in range(cor.shape[1]):
        scatter[i, j] = ax[i, j].scatter(
            X[:, i], X[:, j],
            c=cor[i, j] * np.ones(X.shape[0]), cmap=cmap, norm=norm,
            s=0.5 # Point size
        )
    if i == cor.shape[0] - 1:
        ax[i, j].set_xlabel(
            features[j].replace("_", " "), rotation=45, ha='right')
    if j == 0:
        ax[i, j].set_ylabel(
            features[i].replace("_", " "), rotation=0, ha='right')
    ax[i, j].set_xticks([])
    ax[i, j].set_yticks([])
plt.tight_layout()
cbar = fig.colorbar(scatter[0, 0], ax=ax, orientation='horizontal')
plt.show()

```

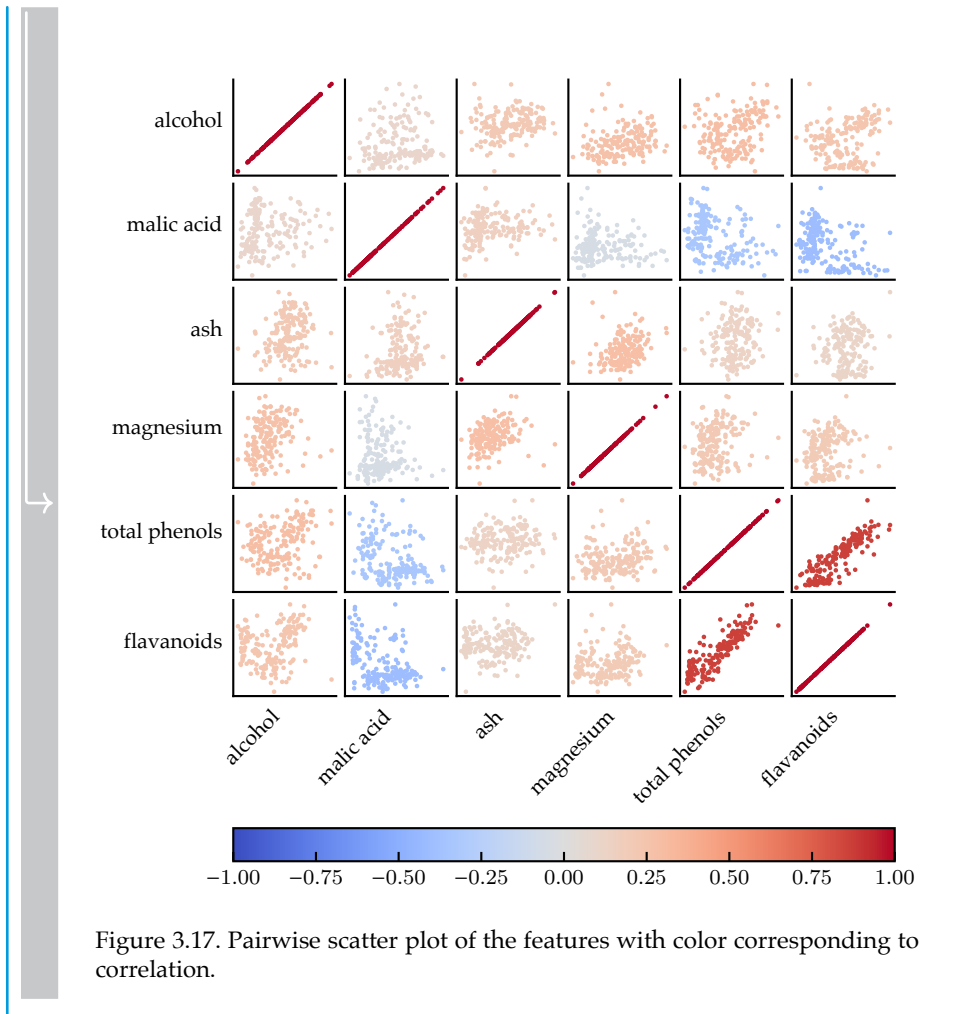


Figure 3.17. Pairwise scatter plot of the features with color corresponding to correlation.

### 3.10.3 Conditional Probability and Dependence

*Independent variables are uncorrelated.* However, *uncorrelated variables may or may not be independent.* Therefore, we cannot use correlation alone as a test for independence. For instance, for random variables  $X$  and  $Y$ , where  $X$  has some even distribution and  $Y = X^2$ , clearly the variables are *dependent*. However, they are also *uncorrelated* (due to symmetry).

### Example 3.15

Using a uniform distribution  $U(-1, 1)$ , show that  $X$  and  $Y$  are uncorrelated (but dependent) with  $Y = X^2$  with some sampling. We compute the correlation for different sample sizes.

Load the necessary libraries.

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

Generate the data for  $x$  and  $y$ .

```
N_a = np.round(np.linspace(10, 500, 100)).astype(int) # Sample sizes
qc_a = np.full(N_a.shape, np.nan) # Correlation initialization
np.random.seed(6) # Seed for reproducibility
x_a = -1 + 2 * np.random.rand(max(N_a)) # Uniform random numbers
y_a = x_a ** 2 #  $y = x^2$ 
```

Calculate the cross-correlation.

```
for i in range(len(N_a)):
    q = np.cov(x_a[:N_a[i]], y_a[:N_a[i]])
    qc = np.corrcoef(x_a[:N_a[i]], y_a[:N_a[i]])
    qc_a[i] = qc[0, 1] # "cross" correlation
```

Plot the absolute cross correlation as a function of sample size.

```
fig, ax = plt.subplots()
p, = ax.plot(N_a, np.abs(qc_a), linewidth=2)
ax.set_xlabel(r'Sample size $N$')
ax.set_ylabel(r'Absolute sample correlation')
ax.set_ylim(bottom=0)
plt.show()
```

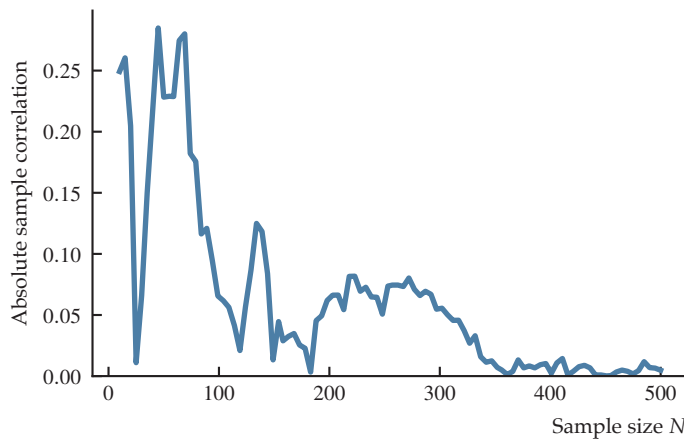



Figure 3.18. Correlation between  $x$  and  $y$  as a function of sample size.


The absolute values of the correlations are shown in the figure. Note that we should probably average several such curves to estimate how the correlation would drop off with  $N$ , but the single curve describes our understanding that the correlation, in fact, approaches zero in the large-sample limit.

---

### 3.11 Problems



**Problem 3.1**  **GRAIN** Several physical processes can be modeled with a *random walk*: a process of iteratively changing a quantity by some random amount. Infinitely many variations are possible, but common factors of variation include probability distribution, step size, dimensionality (e.g. one-dimensional, two-dimensional, etc.), and coordinate system. Graphical representations of these walks can be beautiful. Develop a computer program that generates random walks and corresponding graphics. Do it well and call it art because it is.

**Problem 3.2**  **FREE** Consider the defective spring problem from example 3.4. One way to improve the probability of a true positive test (i.e., the sensitivity) is to add a second test for which a positive event is called  $C$ . Again assuming that the sensitivity and specificity are equal for tests  $B$  and  $C$ , and that the sensitivity of test  $B$  is  $P(B|A) = 0.995$  what is the required sensitivity for test  $C$ ? Clearly state any assumptions.



# 4 Statistics



Whereas probability theory is primarily focused on the relations among mathematical objects, statistics is concerned with making sense of the outcomes of observation (Skiena 2001). However, we frequently use statistical methods to **estimate** probabilistic models. For instance, we will learn how to estimate the standard deviation of a random process we have some reason to expect has a Gaussian probability distribution.

Statistics has applications in nearly every applied science and engineering discipline. Any time measurements are made, statistical analysis is how one makes sense of the results. For instance, determining a reasonable level of confidence in a measured parameter requires statistics.

A particularly hot topic nowadays is **machine learning**, which seems to be a field with applications that continue to expand. This field is fundamentally built on statistics.

A good introduction to statistics appears at the end of (Ash 2008). A more involved introduction is given by (Jaynes et al. 2003). The treatment by (Kreyszig 2011) is rather incomplete, as will be our own.

## 4.1 Populations, Samples, and Machine Learning



An experiment's **population** is a complete collection of objects that we would like to study. These objects can be people, machines, processes, or anything else we would like to understand experimentally.

Of course, we typically can't measure *all* of the population. Instead, we take a subset of the population—called a **sample**—and infer the characteristics of the entire population from this sample.

However, this inference that the sample is somehow representative of the population assumes the sample size is sufficiently large and that the sampling is **random**. This means selection of the sample should be such that no one group within a population are systematically over- or under-represented in the sample.

**Machine learning** is a field that makes extensive use of measurements and statistical inference. In it, an algorithm is **trained** by exposure to sample data, which is called a **training set**. The variables measured are called **features**. Typically, a **predictive model** is developed that can be used to extrapolate from the data to a new situation. The methods of statistical analysis we introduce in this chapter are the foundation of most machine learning methods.

### Example 4.1

Consider a robot, Pierre, with a particular gravitas and sense of style. He seeks the nicest pair of combat boots for wearing in the autumn rains. Pierre is to purchase the boots online via image recognition, and decides to gather data by visiting a hipster hangout one evening to train his style. For a negative contrast, he also watches footage of a white nationalist rally, focusing special attention on the boots of wearers of khakis and polos. Comment on Pierre's methods.

Pierre must identify *features* in the boots, such as color, heel-height, and stitching. Choosing two places to sample certainly enhances the *sample* or *training set*. Positive correlations can be sought with the first group in the sample and negative with the second. The choosing of "desirable" and "undesirable" sample groups is an example of *supervised learning*, which is to say the desirability of one group's boots and the undesirability of the other's is assumed to be known.

## 4.2 Estimation of Sample Mean and Variance

### 4.2.1 Estimation and Sample Statistics



The mean and variance definitions of section 3.7 and section 3.8 apply only to a random variable for which we have a theoretical probability distribution. Typically, it is not until after having performed many measurements of a random variable that we can assign a good distribution model. Until then, measurements can help us *estimate* aspects of the data. We usually start by estimating basic parameters such as *mean* and *variance* before estimating a probability distribution.

There are two key aspects to randomness in the measurement of a random variable. First, of course, there is the underlying randomness with its probability distribution, mean, standard deviation, etc., which we call the *population statistics*. Second, there is the *statistical variability* that is due to the fact that we are *estimating* the random variable's statistics—called its *sample statistics*—from some sample. Statistical variability is decreased with greater sample size and number of samples, whereas the underlying randomness of the random variable does not decrease. Instead, our estimates of its probability distribution and statistics improve.

### 4.2.2 Sample Mean, Variance, and Standard Deviation

The *arithmetic mean* or **sample mean** of a measurand with sample size  $N$ , represented by random variable  $X$ , is defined as

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

If the sample size is large,  $\bar{x} \rightarrow m_X$  (the sample mean approaches the mean). The **population mean** is another name for the mean  $m_X$ , which is equal to

$$m_X = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N x_i.$$

Recall that the *definition* of the mean is  $m_X = E[x]$ .

The **sample variance** of a measurand represented by random variable  $X$  is defined as

$$S_X^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2.$$

If the sample size is large,  $S_X^2 \rightarrow \sigma_X^2$  (the sample variance approaches the variance). The **population variance** is another term for the variance  $\sigma_X^2$ , and can be expressed

as

$$\sigma_X^2 = \lim_{N \rightarrow \infty} \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2.$$

Recall that the *definition* of the variance is  $\sigma_X^2 = E[(X - m_X)^2]$ .

The *sample standard deviation* of a measurand represented by random variable  $X$  is defined as

$$S_X = \sqrt{S_X^2}.$$

If the sample size is large,  $S_X \rightarrow \sigma_X$  (the sample standard deviation approaches the standard deviation). The *population standard deviation* is another term for the standard deviation  $\sigma_X$ , and can be expressed as

$$\sigma_X = \lim_{N \rightarrow \infty} \sqrt{S_X^2}.$$

Recall that the *definition* of the standard deviation is  $\sigma_X = \sqrt{\sigma_X^2}$ .

#### 4.2.3 Sample Statistics as Random Variables

There is an ambiguity in our usage of the term “sample.” It can mean just one measurement or it can mean a collection of measurements gathered together. Hopefully, it is clear from context.

In the latter sense, often we collect multiple samples, each of which has its own sample mean  $\bar{X}_i$  and standard deviation  $S_{X_i}$ . In this situation,  $\bar{X}_i$  and  $S_{X_i}$  are themselves random variables (meta af, I know). This means they have their own sample means  $\overline{\bar{X}_i}$  and  $\overline{S_{X_i}}$  and standard deviations  $S_{\bar{X}_i}$  and  $S_{S_{X_i}}$ .

The **mean of means**  $\overline{\bar{X}_i}$  is equivalent to a mean with a larger sample size and is therefore our best estimate of the mean of the underlying random process. The **mean of standard deviations**  $\overline{S_{X_i}}$  is our best estimate of the standard deviation of the underlying random process. The **standard deviation of means**  $S_{\bar{X}_i}$  is a measure of the spread in our estimates of the mean. It is our best estimate of the standard deviation of the statistical variation and should therefore tend to zero as sample size and number of samples increases. The **standard deviation of standard deviations**  $S_{S_{X_i}}$  is a measure of the spread in our estimates of the standard deviation of the underlying process. It should also tend to zero as sample size and number of samples increases.

Let  $N$  be the size of each sample. It can be shown that the standard deviation of the means  $S_{\bar{X}_i}$  can be estimated from a single sample standard deviation:

$$S_{\bar{X}_i} \approx \frac{S_{X_i}}{\sqrt{N}}.$$

This shows that as the sample size  $N$  increases, the statistical variability of the mean decreases (and in the limit approaches zero).

#### 4.2.4 Nonstationary Signal Statistics

The sample mean, variance, and standard deviation definitions, above, assume the random process is *stationary*—that is, its population mean does not vary with time. However, a great many measurement signals have populations that *do* vary with time, i.e. they are *nonstationary*. Sometimes the nonstationarity arises from a “drift” in the dc value of a signal or some other slowly changing variable. But dynamic signals can also change in a recognizable and predictable manner, as when, say, the temperature of a room changes when a window is opened or when a water level changes with the tide.

Typically, we would like to minimize the effect of nonstationarity on the signal statistics. In certain cases, such as drift, the variation is a nuisance only, but other times it is the point of the measurement.

Two common techniques are used, depending on the overall type of nonstationarity. If it is periodic with some known or estimated period, the measurement data series can be “folded” or “reshaped” such that the  $i$ th measurement of each period corresponds to the  $i$ th measurement of all other periods. In this case, somewhat counterintuitively, we can consider the  $i$ th measurements to correspond to a sample of size  $N$ , where  $N$  is the number of periods over which measurements are made.

When the signal is aperiodic, we often simply divide it into “small” (relative to its overall trend) intervals over which statistics are computed, separately.

Note that in this discussion, we have assumed that the nonstationarity of the signal is due to a variable that is deterministic (not random).

#### Example 4.2

Consider the measurement of the temperature inside a desktop computer chassis via an inexpensive *thermistor*, a resistor that changes resistance with temperature. The processor and power supply heat the chassis in a manner that depends on processing demand. For the test protocol, the processors are cycled sinusoidally through processing power levels at a frequency of 50 mHz for  $n_T = 12$  periods and sampled at 1 Hz. Assume a temperature fluctuation between about 20 and 50 C and gaussian noise with standard deviation 4 C. Consider a *sample* to be the multiple measurements of a certain instant in the period.

1. Generate and plot simulated temperature data as a time series and as a histogram or frequency distribution. Comment on why the frequency distribution sucks.

2. Compute the sample mean and standard deviation *for each sample in the cycle*.
3. Subtract the mean from each sample in the period such that each sample distribution is centered at zero. Plot the composite frequency distribution of all samples, together. This represents our best estimate of the frequency distribution of the underlying process.
4. Plot a comparison of the theoretical mean, which is 35, and the sample mean of means with an error bar. Vary the number of samples  $n_T$  and comment on its effect on the estimate.
5. Plot a comparison of the theoretical standard deviation and the sample mean of sample standard deviations with an error bar. Vary the number of samples  $n_T$  and comment on its effect on the estimate.
6. Plot the sample means over a single period with error bars of  $\pm$  one sample standard deviation of the means. This represents our best estimate of the sinusoidal heating temperature. Vary the number of samples  $n_T$  and comment on the estimate.

We proceed in Python. First, load packages:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

**Generate the Temperature Data** The temperature data can be generated by constructing an array that is passed to a sinusoid, then “randomized” by gaussian random numbers.

Set a random seed for reproducible pseudorandom numbers.

```
np.random.seed(43)
```

Define constants with

```
f = 50e-3 # Hz
a = 15    # C
dc = 35   # C
fs = 1    # Hz
nT = 12   # number of sinusoid periods
s = 4     # C
np_ = int(fs / f + 1) # number of samples per period
n = nT * np_ + 1     # total number of samples
```

Generate the temperature data.

```
t_a = np.linspace(0, nT / f, n)
sin_a = dc + a * np.sin(2 * np.pi * f * t_a)
noise_a = s * np.random.randn(n)
signal_a = sin_a + noise_a
```

Plot temperature over time

```
fig, ax = plt.subplots()
ax.plot(t_a, signal_a, 'o-', color='0.8', markerfacecolor='b', markersize=3)
plt.xlabel('time (s)')
plt.ylabel('temperature (C)')
plt.draw()
```

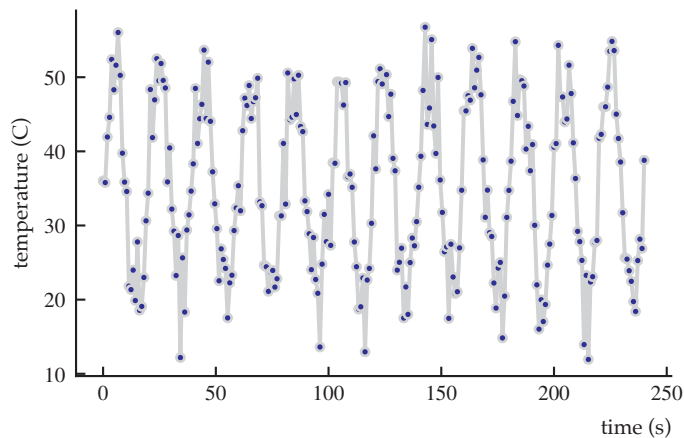


Figure 4.1. Raw temperature data over time.

This is something like what we might see for continuous measurement data. Now, the histogram.

```
fig, ax = plt.subplots()
ax.hist(signal_a, bins=30, density=True, alpha=0.5)
plt.xlabel('temperature (C)')
plt.ylabel('probability')
plt.draw()
```

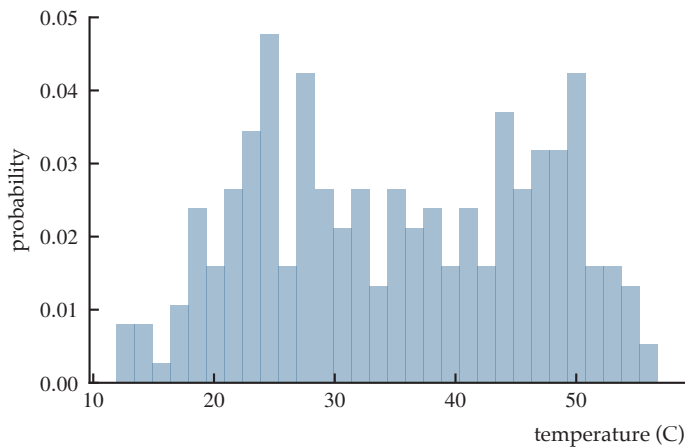


Figure 4.2. Raw temperature data histogram.

This sucks because we plot a frequency distribution to tell us about the random variation, but this data includes the sinusoid.

**Sample Mean, Variance, and Standard Deviation** To compute the sample mean  $\mu$  and standard deviation  $s$  for each sample in the period, we must “pick out” the  $nT$  data points that correspond to each other. Currently, they’re in one long  $1 \times n$  array `signal_a`. It is helpful to *reshape* the data so it is in an  $nT \times np$  array, which each row corresponding to a new period. This leaves the correct points aligned in columns. It is important to note that we can do this “folding” operation only when we know rather precisely the period of the underlying sinusoid. It is given in the problem that it is a controlled experiment variable. If we did not know it, we would have to estimate it, too, from the data.

Reshape data for sample mean, variance, and standard deviation calculations with

```
| signal_ar = signal_a[:-1].reshape((nT, np_))
```

Compute sample mean, variance, and standard deviations with

```
| mu_a = np.array([np.mean(col) for col in signal_ar.T])
| var_a = np.array([np.var(col) for col in signal_ar.T])
| s_a = np.array([np.std(col) for col in signal_ar.T])
```



**Composite Frequency Distribution** The columns represent samples. We want to subtract the mean from each column. We use  `repmat`  to reproduce  `mu_a`  in  `nT`  rows so it can be easily subtracted.

```
signal_arz = signal_ar - mu_a[np.newaxis,:]  
x_a = np.linspace(-15, 15, 100)  
pdfit_a = norm.pdf(x_a, loc=0, scale=s)  
pdf_a = norm.pdf(x_a, loc=0, scale=s)
```

Now that all samples have the same mean, we can lump them into one big bin for the frequency distribution.

Plot composite frequency distribution with a probability distribution fit and the original probability distribution used to generate the data.

```
fig,ax = plt.subplots()  
ax.hist(signal_arz.ravel(), bins=int(s * np.sqrt(nT)), density=True, alpha=0.5)  
ax.plot(x_a, pdfit_a, 'b-', linewidth=2, label='pdf est.')  
ax.plot(x_a, pdf_a, 'g--', linewidth=2, label='pdf')  
plt.xlabel('Zero-mean temperature (C)')  
plt.ylabel('Probability mass/density')  
plt.legend()  
plt.draw()
```

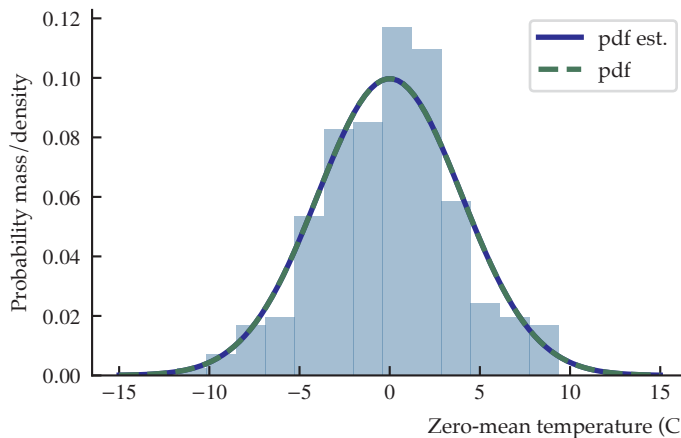


Figure 4.3. Composite frequency distribution of zero-mean temperature data.

**Means Comparison** The sample mean of means is simply the following:

```
mu_mu = np.mean(mu_a)
```

The standard deviation that works as an error bar, which should reflect how well we can estimate the point plotted, is the standard deviation of the means. It is difficult to compute this directly for a nonstationary process. We use the estimate given above and improve upon it by using the mean of standard deviations instead of a single sample's standard deviation.

```
| s_mu = np.mean(s_a) / np.sqrt(nT)
```

Plot sample mean of means with an error bar as follows:

```
| fig, ax = plt.subplots()
| ax.bar(['$\overline{\overline{X}}$'], [mu_mu], yerr=s_mu, color='b', capsize=5)
| plt.xlabel('Sample Mean of Means')
| plt.draw()
```

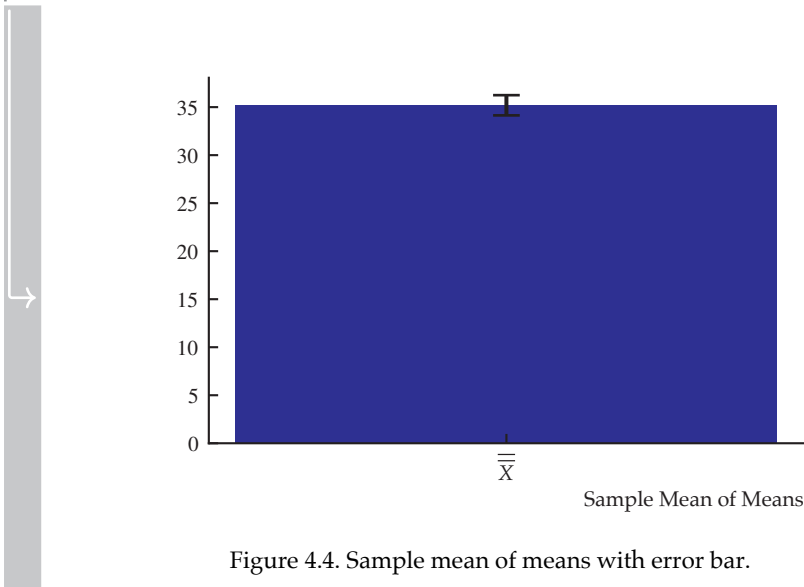


Figure 4.4. Sample mean of means with error bar.

**Standard Deviations Comparison** The sample mean of standard deviations is simply the following:

```
| mu_s = np.mean(s_a)
```

The standard deviation that works as an error bar, which should reflect how well we can estimate the point plotted, is the standard deviation of the standard deviations. We can compute this directly.

```
| s_s = np.std(s_a)
```

Plot sample mean of standard deviations with error bar as follows:

```
fig,ax = plt.subplots()
ax.bar(['$\overline{S_X}$'], [mu_s], yerr=s_s, color='b', capsize=5)
plt.xlabel('Sample Mean of Sample Standard Deviations')
plt.draw()
```

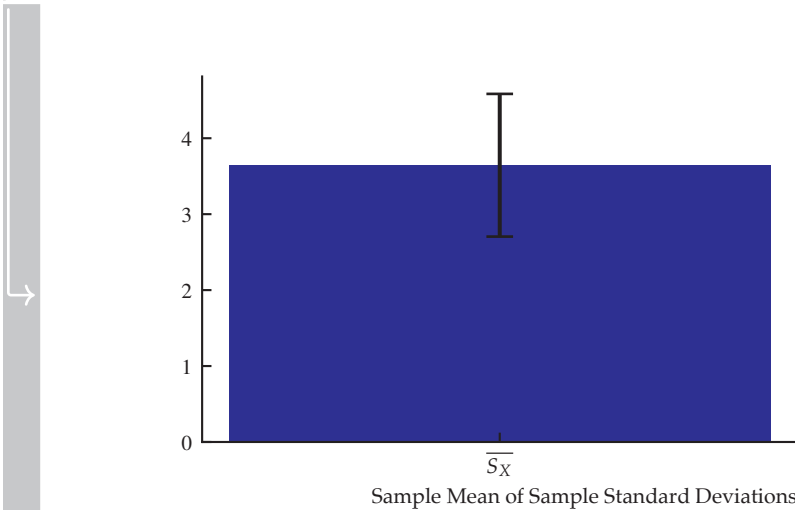


Figure 4.5. Sample mean of sample standard deviations with error bar.

**Plot a Period with Error Bars** Plotting the data with error bars is fairly straightforward. The main question is “which standard deviation?” Since we’re plotting the means, it makes sense to plot the error bars as a single sample standard deviation of the means.

Plot sample means over a single period with error bars as follows:

```
fig,ax = plt.subplots()
ax.errorbar(t_a[:np_], mu_a, yerr=s_a, fmt='o-', capsize=2, label='sample mean', color='b')
t_a2 = np.linspace(0, 1 / f, 101)
ax.plot(t_a2, dc + a * np.sin(2 * np.pi * f * t_a2), 'r-', label='population mean')
plt.xlim([t_a[0], t_a[np_ - 1]])
plt.xlabel('Folded time (s)')
plt.ylabel('Temperature (C)')
plt.legend()
plt.show() # Show all the plots
```

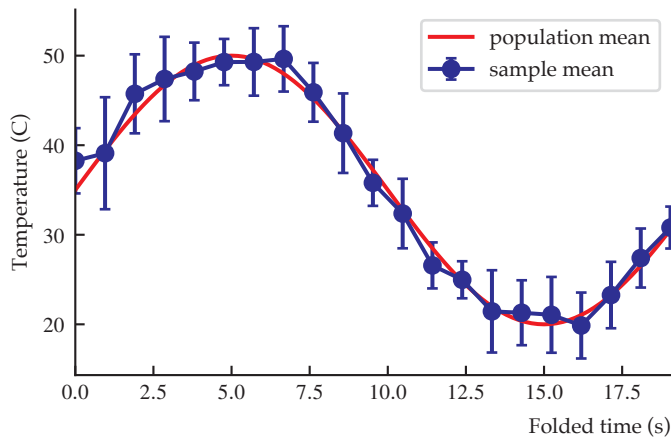


Figure 4.6. Sample means over a single period with error bars.

### 4.3 Confidence



One really ought to have it to give a lecture named it, but we'll give it a try anyway. **Confidence** is used in the common sense, although we do endow it with a mathematical definition to scare business majors, who aren't actually impressed, but indifferent. Approximately: if, under some reasonable assumptions (probabilistic model), we estimate the probability of some event to be  $P\%$ , we say we have  $P\%$  confidence in it. I mean, business majors are all, "Supply and demand? Let's call that a 'law,' " so I think we're even.

So we're back to computing probability from distributions—probability density functions (PDFs) and probability mass functions (PMFs). Usually we care most about estimating the mean of our distribution. Recall from the previous lecture that when several samples are taken, each with its own mean, the mean is itself a random variable—with a mean, of course. Meanception.

But the mean has a probability distribution of its own. The **central limit theorem** has as one of its implications that, as the sample size  $N$  gets large, *regardless of the sample distributions, this distribution of means approaches the Gaussian distribution.*

But sometimes I always worry I'm being lied to, so let's check.

### 4.3.1 Checking the Central Limit Theorem

We proceed in Python. First, load packages:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

**Generate Data** Generate some data to test the central limit theorem {#generate-some-data-to-test-the-central-limit-theorem h="3y"}

Data can be generated by constructing an array using a (seeded for consistency) random number generator. Let's use a uniformly distributed PDF between 0 and 1.

```
N = 150 # Sample size (number of measurements per sample)
M = 120 # Number of samples
n = N * M # Total number of measurements
mu_pop = 0.5 # Because it's a uniform PDF between 0 and 1
np.random.seed(11) # Seed the random number generator
signal_a = np.random.rand(N, M) # Uniform PDF
#
# Let's take a look at the data by plotting the first ten samples
# (columns) versus index, as shown in the figure below

samples_to_plot = 10
fig, ax = plt.subplots()
for j in range(samples_to_plot):
    ax.plot(signal_a[:, j], 'o-', markersize=3)
plt.xlabel('index')
plt.ylabel('measurement')
plt.draw()
```

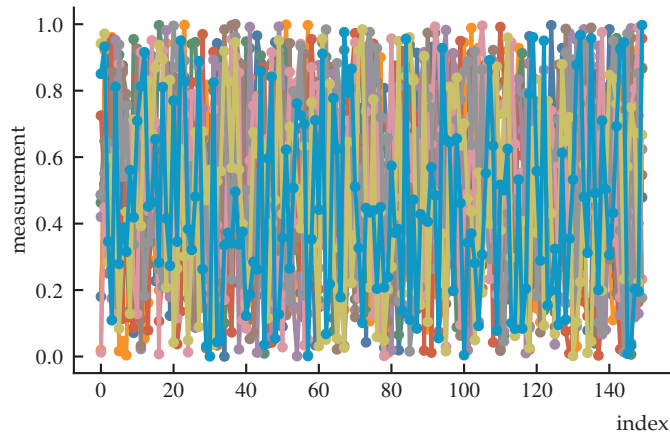


Figure 4.7. Raw data with colors corresponding to samples.

This is something like what we might see for continuous measurement data. Now make a histogram of each sample:

```
c = plt.cm.jet(np.linspace(0, 1, samples_to_plot)) # Color array
fig, ax = plt.subplots()
for j in range(samples_to_plot):
    plt.hist(signal_a[:, j],
             bins=30, # Number of bins
             color=c[j],
             alpha=0.3,
             density=True) # For PMF
plt.xlim([-0.05, 1.05])
plt.xlabel('Measurement')
plt.ylabel('Probability')
plt.draw()
```

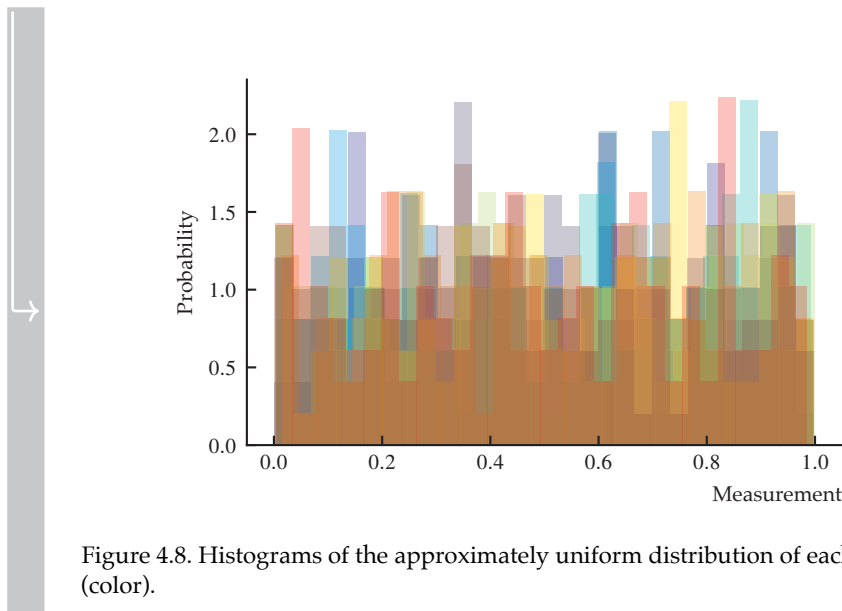


Figure 4.8. Histograms of the approximately uniform distribution of each sample (color).

This isn't a great plot, but it shows roughly that each sample is fairly uniformly distributed.

**Sample Statistics** Now let's check out the sample statistics. We want the sample mean and standard deviation of each column. Let's use the built-in functions `mean` and `std`.

```
mu_a = np.mean(signal_a, axis=0) # Mean of each column
s_a = np.std(signal_a, axis=0) # Standard deviation of each column
```

Now we can compute the mean statistics, both the mean of the mean  $\overline{\overline{X}}$  and the standard deviation of the mean  $s_{\overline{X}}$ , which we don't strictly need for this part, but we're curious. We choose to use the direct estimate instead of the  $s_X/\sqrt{N}$  formula, but they should be close.

```
mu_mu = np.mean(mu_a)
s_mu = np.std(mu_a)
```

**The Truth about Sample Means** It's the moment of truth. Let's plot the histogram of the sample means as follows:

```
fig, ax = plt.subplots()
plt.hist(mu_a,
        bins=30, # You can adjust the number of bins as needed
        density=True) # For PMF
plt.xlabel('Measurement')
plt.ylabel('Probability')
plt.draw()
```

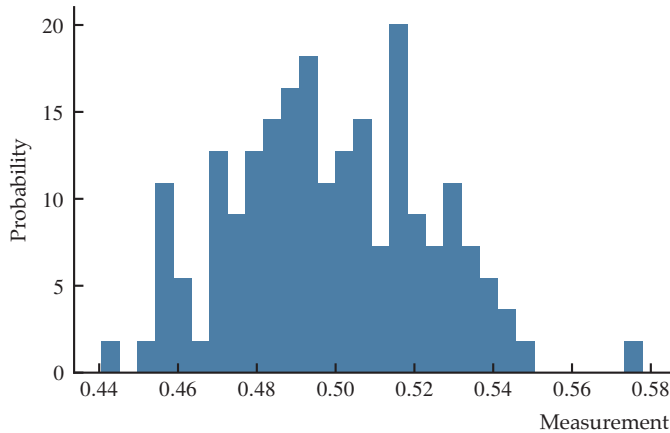


Figure 4.9. Histogram of the approximately normal distribution of the means.

This looks like a Gaussian distribution about the mean of means, so I guess the central limit theorem is legit.

### 4.3.2 Gaussian and Probability

We already know how to compute the probability  $P$  a value of a random variable  $X$  lies in a certain interval from a PMF or PDF (the sum or the integral, respectively). This means that, for sufficiently large sample size  $N$  such that we can assume from the central limit theorem that the sample means  $\bar{x}_i$  are normally distributed, *the probability a sample mean value  $\bar{x}_i$  is in a certain interval* is given by integrating the Gaussian PDF. The Gaussian PDF for random variable  $Y$  representing the sample means is

$$f(y) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-(y - \mu)^2}{2\sigma^2}.$$

where  $\mu$  is the population mean and  $\sigma$  is the population standard deviation.

The integral of  $f$  over some interval is the probability a value will be in that interval. Unfortunately, that integral is uncool. It gives rise to the definition of the *error function*, which, for the Gaussian random variable  $Y$ , is



$$\operatorname{erf}(y_b) = \frac{1}{\sqrt{\pi}} \int_{-y_b}^{y_b} e^{-t^2} dt.$$

This expresses the probability a sample mean being in the interval  $[-y_b, y_b]$  if  $Y$  has mean 0 and variance 1/2.

Python has the error function in the `scipy.special` package. Let's plot the error function:

```
from scipy.special import erf
y_a = np.linspace(0, 3, 100)
fig, ax = plt.subplots()
ax.plot(y_a, erf(y_a), linewidth=2)
plt.grid(True)
plt.xlabel(r'Interval bound $y_b$')
plt.ylabel(r'Error function $\text{erf}(y_b)$')
plt.draw()
```

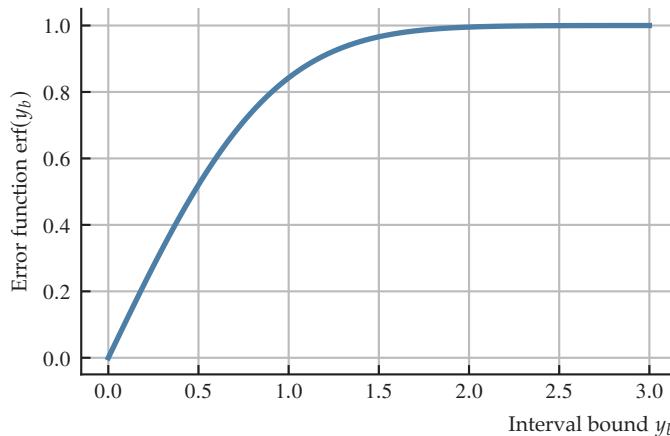


Figure 4.10. Error function for Gaussian random variable.

We could deal directly with the error function, but most people don't and we're weird enough, as it is. Instead, people use the **Gaussian cumulative distribution function** (CDF)  $\Phi: \mathbb{R} \rightarrow \mathbb{R}$ , which is defined as

$$\Phi(z) = \frac{1}{2} \left( 1 + \operatorname{erf} \left( \frac{z}{\sqrt{2}} \right) \right)$$

and which expresses the probability of a Gaussian random variable  $Z$  with mean 0 and standard deviation 1 taking on a value in the interval  $(-\infty, z]$ . The Gaussian CDF and PDF are plotted below.

```

from scipy.stats import norm
z_a = np.linspace(-3, 3, 300)
threshold = 1.5
a_pdf = lambda z: (z < threshold) * norm.pdf(z, 0, 1)
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 12)) # Two subplots
ax1.fill_between(z_a, a_pdf(z_a), color=[.8, .8, .8])
ax1.plot(z_a, norm.pdf(z_a, 0, 1), linewidth=2)
ax1.grid(True)
ax1.set_xlabel(r'$z$')
ax1.set_ylabel(r'Gaussian PDF $f(z)$')
ax1.text(1.5, norm.pdf(1.5, 0, 1) + .01, r'$z_b$')
ax1.legend([r'$\Phi(z_b)$', r'$f(z)$'])
ax2.plot(z_a, 1/2 * (1 + erf(z_a / np.sqrt(2))), linewidth=2)
ax2.grid(True)
ax2.set_xlabel(r'interval bound $z_b$')
ax2.set_ylabel(r'Gaussian CDF $\Phi(z_b)$')
plt.show()

```

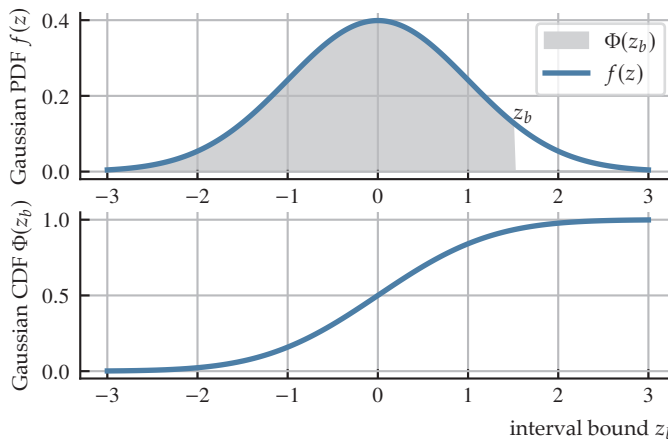


Figure 4.11. Gaussian PDF and CDF for  $z$ -scores.

Values can be taken directly from the graph, but it's more accurate to use the table of values in appendix A.1.

That's great and all, but occasionally (always) we have Gaussian random variables with nonzero means and nonunity standard deviations. It turns out we can shift any Gaussian random variable by its mean and scale it by its standard deviation to make it have zero mean and standard deviation. We can then use  $\Phi$  and interpret the results as being relative to the mean and standard deviation, using phrases like “the probability it is within two standard deviations of its mean.” The transformed

random variable  $Z$  and its values  $z$  are sometimes called the **z-score**. For a particular value  $x$  of a random variable  $X$ , we can compute its z-score (or value  $z$  of random variable  $Z$ ) with the formula

$$z = \frac{x - \mu_X}{\sigma_X}$$

and compute the probability of  $X$  taking on a value within the interval, say,  $x \in [x_{b-}, x_{b+}]$  from the table. (Sample statistics  $\bar{X}$  and  $S_X$  are appropriate when population statistics are unknown.)

For instance, compute the probability a Gaussian random variable  $X$  with  $\mu_X = 5$  and  $\sigma_X = 2.34$  takes on a value within the interval  $x \in [3, 6]$ .

1. Compute the z-score of each endpoint of the interval:

$$z_3 = \frac{3 - \mu_X}{\sigma_X} \approx -0.85$$

$$z_6 = \frac{6 - \mu_X}{\sigma_X} \approx 0.43.$$

2. Look up the CDF values for  $z_3$  and  $z_6$ , which are  $\Phi(z_3) = 0.1977$  and  $\Phi(z_6) = 0.6664$ . 3. The CDF values correspond to the probabilities  $x < 3$  and  $x < 6$ . Therefore, to find the probability  $x$  lies in that interval, we subtract the lower bound probability:

$$\begin{aligned} P(x \in [3, 6]) &= P(x < 6) - P(x < 3) \\ &= \Phi(6) - \Phi(3) \\ &\approx 0.6664 - 0.1977 \\ &\approx 0.4689. \end{aligned}$$

So there is a 46.89 percent probability, and therefore we have 46.89 percent confidence, that  $x \in [3, 6]$ .

Often we want to go the other way, estimating the symmetric interval  $[x_{b-}, x_{b+}]$  for which there is a given probability. In this case, we first look up the z-score corresponding to a certain probability. For concreteness, given the same population statistics above, let's find the symmetric interval  $[x_{b-}, x_{b+}]$  over which we have 90 percent confidence. From the table, we want two, symmetric z-scores that have CDF-value difference 0.9. Or, in maths,

$$\Phi(z_{b+}) - \Phi(z_{b-}) = 0.9 \quad \text{and} \quad z_{b+} = -z_{b-}.$$

Due to the latter relation and the additional fact that the Gaussian CDF has antisymmetry,

$$\Phi(z_{b+}) + \Phi(z_{b-}) = 1.$$

Adding the two  $\Phi$  equations, we get

$$\begin{aligned}\Phi(z_{b+}) &= 1.9/2 \\ &= 0.95\end{aligned}$$

and  $\Phi(z_{b-}) = 0.05$ . From the table, these correspond (with a linear interpolation) to  $z_b = z_{b+} = -z_{b-} \approx 1.645$ . All that remains is to solve the z-score formula for  $x$ :

$$x = \mu_X + z\sigma_X.$$

From this,

$$x_{b+} = \mu_X + z_{b+}\sigma_X \approx 8.849$$

$$x_{b-} = \mu_X + z_{b-}\sigma_X \approx 1.151.$$

and  $X$  has a 90 percent confidence interval  $[1.151, 8.849]$ .

### Example 4.3

Consider the data set generated above. What is our 95% confidence interval in our estimate of the mean?

Assuming we have a sufficiently large data set, the distribution of means is approximately Gaussian. Following the same logic as above, we need z-score that gives an upper CDF value of  $(1 + 0.95)/2 = 0.975$ . From the table, we obtain the  $z_b = z_{b+} = -z_{b-}$ , below.

```
| z_b = 1.96
```

Now we can estimate the mean using our sample and mean statistics,

$$\overline{X} = \overline{\overline{X}} \pm z_b S_{\overline{X}}. \quad (4.1)$$

```
| mu_x_95 = mu_mu + np.array([-z_b, z_b])*s_mu
```

```
| [0.4526      0.5449]
```

This is our 95 percent confidence interval in our estimate of the mean.

#### 4.4 Student Confidence



The central limit theorem tells us that, for large sample size  $N$ , the distribution of the means is Gaussian. However, for small sample size, the Gaussian isn't as good of an estimate. **Student's t-distribution** is superior for lower sample size and equivalent at higher sample size. Technically, if the population standard deviation  $\sigma_X$  is known, even for low sample size we should use the Gaussian distribution. However, this rarely arises in practice, so we can usually get away with an "always t" approach.

A way that the t-distribution accounts for low- $N$  is by having an entirely different distribution for each  $N$  (seems a bit of a cheat, to me). Actually, instead of  $N$ , it uses the **degrees of freedom**  $\nu$ , which is  $N$  minus the number of parameters required to compute the statistic. Since the standard deviation requires only the mean, for most of our cases,  $\nu = N - 1$ .

As with the Gaussian distribution, the t-distribution's integral is difficult to calculate. Typically, we will use a t-table, such as the one given in appendix A.2. There are three points of note.

1. Since we are primarily concerned with going from probability / confidence values (e.g.  $P\%$  probability / confidence) to intervals, typically there is a column for each probability.
2. The extra parameter  $\nu$  takes over one of the dimensions of the table because three-dimensional tables are illegal.
3. Many of these tables are "two-sided," meaning their t-scores and probabilities assume you want the symmetric probability about the mean over the interval  $[-t_b, t_b]$ , where  $t_b$  is your t-score bound.

Consider the following example.

##### Example 4.4

Write a Python script to generate a data set with 200 samples and sample sizes  $N \in \{10, 20, 100\}$  using any old distribution. Compare the distribution of the means for the different  $N$ . Use the sample distributions and a t-table to compute 99% confidence intervals.

We proceed in Python. First, load packages:

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

Generate the data set.

```

confidence = 0.99 # Requirement
M = 200 # Number of samples
N_a = [10, 20, 100] # Sample sizes
mu = 27 # Population mean
sigma = 9 # Population standard deviation
np.random.seed(1) # Seed random number generator
data_a = mu + sigma * np.random.randn(N_a[-1], M) # Generate normal data
print(data_a[:10, :5]) # Check 10 rows and five columns

```

```

[[41.61910827 21.49419228 22.24645423 17.3432824 34.78866866]
 [23.39209627 34.41605057 21.93925112 44.59390268 15.012435 ]
 [15.24119334 27.68742432 30.30508632 38.09609273 23.19428735]
 [17.34332149 31.4564275 18.43144109 22.33669003 13.84736756]
 [34.32908816 34.02422937 13.82351784 25.60957926 26.16810913]
 [25.62087454 5.10742339 31.57185903 24.08370904 13.40031053]
 [22.14870678 32.79689989 28.65270217 26.22215814 25.07410997]
 [28.70278877 38.01542408 24.29162355 29.26178669 35.35461193]
 [29.61904088 36.67409774 20.7197109 21.79506855 19.37292716]
 [15.22825791 40.25156676 27.67388488 10.91758137 28.48689528]]

```

Compute the means for different sample sizes.

```

mu_a = np.full((len(N_a), M), np.nan)
for i in range(len(N_a)):
    mu_a[i, :] = np.mean(data_a[:N_a[i], :M], axis=0)

```

Plot a histogram of the distribution of the means.

```

fig, ax = plt.subplots()
for i in range(len(N_a)):
    plt.hist(mu_a[i, :], bins=30, alpha=0.5, label=f'Sample size {N_a[i]}')
plt.xlabel('Mean Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

```

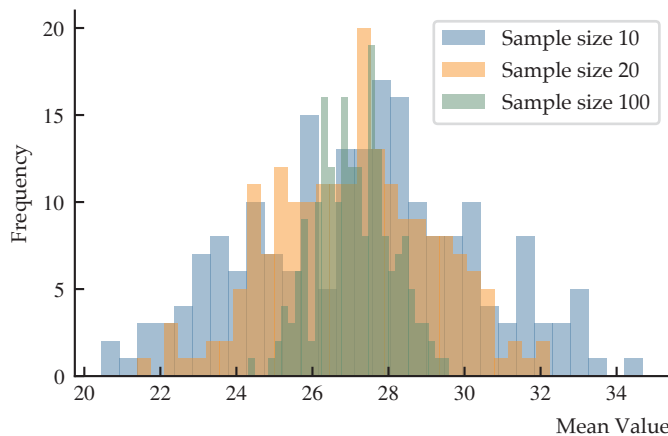


Figure 4.12. Histogram of the distribution of the means for different sample sizes.

It makes sense that the larger the sample size, the smaller the spread. A quantitative metric for the spread is, of course, the standard deviation of the means for each sample size.

```
S_mu = np.std(mu_a, axis=1, ddof=0)
print(S_mu)

[ 2.92548459  2.08250569  0.97864856]
```

Look up t-table values or use scipy to compute the t-value for different sample sizes and 99 percent confidence. Use these, the mean of means, and the standard deviation of means to compute the 99 percent confidence interval for each  $N$ .

```
from scipy.stats import t
t_a = t.ppf(confidence, np.array(N_a) - 1) # t-value for confidence
for i in range(len(N_a)):
    interval = np.mean(mu_a[i, :]) + np.array([-1, 1]) * t_a[i] * S_mu[i]
    print(f'interval for N = {N_a[i]}: {interval}')
```

```
interval for N = 10: [19.04354748 35.55169379]
interval for N = 20: [21.81853996 32.39551634]
interval for N = 100: [24.77231819 29.40055444]
```

As expected, the larger the sample size, the smaller the interval over which we have 99 percent confidence in the estimate.

## 4.5 Regression



Suppose we have a sample with two measurands: (1) the force  $F$  through a spring and (2) its displacement  $X$  (not from equilibrium).

We would like to determine an analytic function that relates the variables, perhaps for prediction of the force given another displacement.

There is some variation in the measurement. Let's say the following is the sample.

```
X_a = 1e-3 * np.array(
    [10, 21, 30, 41, 49, 50, 61, 71, 80, 92, 100]
) # m
F_a = np.array(
    [50.1, 50.4, 53.2, 55.9, 57.2, 59.9, 61.0, 63.9, 67.0, 67.9, 70.3]
) # N
```

Let's take a look at the data.

```
fig, ax = plt.subplots()
p = ax.plot(X_a * 1e3, F_a, '.b', markersize=15)
ax.set_xlabel(r'$X$ (mm)')
ax.set_ylabel(r'$F$ (N)')
ax.set_xlim([0, np.max(X_a * 1e3)])
ax.grid(True)
plt.draw()
```

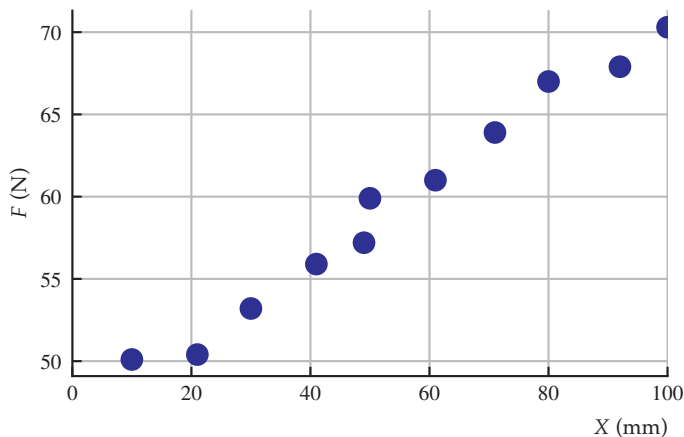


Figure 4.13. Force  $F$  as a function of displacement  $X$ .

How might we find an analytic function that agrees with the data? Broadly, our strategy will be to assume a general form of a function and use the data to set



the parameters in the function such that the difference between the data and the function is minimal.

Let  $y$  be the analytic function that we would like to fit to the data. Let  $y_i$  denote the value of  $y(x_i)$ , where  $x_i$  is the  $i$ th value of the random variable  $X$  from the sample. Then we want to minimize the differences between the force measurements  $F_i$  and  $y_i$ .

From calculus, recall that we can minimize a function by differentiating it and solving for the zero-crossings (which correspond to local maxima or minima).

First, we need such a function to minimize. Perhaps the simplest, effective function  $D$  is constructed by squaring and summing the differences we want to minimize, for sample size  $N$ :

$$D(x_i) = \sum_{i=1}^N (F_i - y_i)^2$$

(recall that  $y_i = y(x_i)$ , which makes  $D$  a function of  $x$ ).

Now the form of  $y$  must be chosen. We consider only  $m$ th-order polynomial functions  $y$ , but others can be used in a similar manner:

$$y(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m.$$

If we treat  $D$  as a function of the polynomial coefficients  $a_j$ , i.e.

$$D(a_0, a_1, \dots, a_m),$$

and minimize  $D$  for each value of  $x_i$ , we must take the partial derivatives of  $D$  with respect to each  $a_j$  and set each equal to zero:

$$\frac{\partial D}{\partial a_0} = 0, \quad \frac{\partial D}{\partial a_1} = 0, \quad \dots, \quad \frac{\partial D}{\partial a_m} = 0.$$

This gives us  $N$  equations and  $m + 1$  unknowns  $a_j$ . Writing the system in matrix form,

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ 1 & x_3 & x_3^2 & \cdots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^m \end{bmatrix}}_{A_{N \times (m+1)}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}}_{\mathbf{a}_{(m+1) \times 1}} = \underbrace{\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_N \end{bmatrix}}_{\mathbf{b}_{N \times 1}}.$$

Typically  $N > m$  and this is an *overdetermined system*. Therefore, we usually can't solve by taking  $A^{-1}$  because  $A$  doesn't have an inverse!

Instead, we either find the *Moore-Penrose pseudo-inverse*  $A^+$  and have  $\mathbf{a} = A^+\mathbf{b}$  as the solution, which is *inefficient* (even with NumPy's `linalg.pinv()` function)—or we

can approximate  $\mathbf{b}$  with an algorithm such as that used in the *least-squares* method, which has Numpy function `linalg.lstsq()`. We'll use the latter method.

### Example 4.5

Use Numpy to find the least-squares polynomial fit for the sample. There's the sometimes-difficult question, "What order should we fit?" Let's try out several and see what the squared-differences function  $D$  gives.

Begin by writing a function that takes the sample data and the order of the polynomial fit and returns the coefficients of the polynomial.

```
def poly_fit(X, F, order):
    A = np.vander(X, order + 1, increasing=True) # Vandermonde matrix
    # This is the matrix A in the system of equations
    return np.linalg.lstsq(A, F, rcond=None)[0] # Coefficients
```

Fit the data with polynomials of orders 1, 3, 5, 7, and 9.

```
orders = [1, 3, 5, 7, 9]
coefficients = [poly_fit(X_a, F_a, order) for order in orders]
```

Now we can plot the data and the fitted polynomials.

```
fig, ax = plt.subplots()
p = ax.plot(X_a * 1e3, F_a, '.b', markersize=15)
x = np.linspace(np.min(X_a), np.max(X_a), 100)
for i, order in enumerate(orders):
    y = np.polyval(coefficients[i][::-1], x)
    ax.plot(x * 1e3, y, label=f'Order {order}')
ax.set_xlabel(r'$X$ (mm)')
ax.set_ylabel(r'$F$ (N)')
ax.legend()
plt.draw()
```

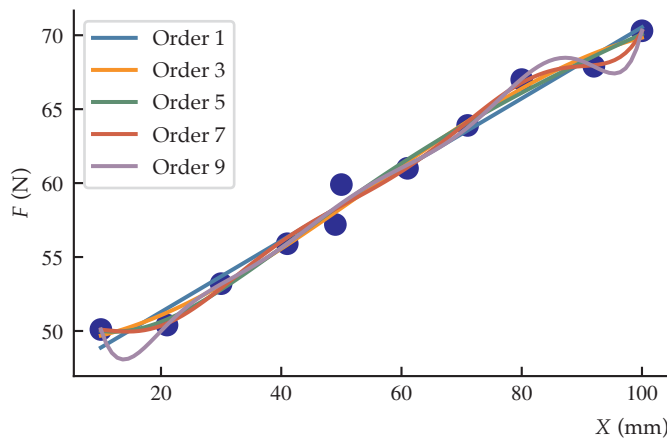


Figure 4.14. Data and fitted polynomials of different orders.

The plot shows the data points and the fitted polynomials of different orders. The higher-order polynomials seem to fit the data better, but they may be over-fitting. We can quantify the goodness of fit by calculating the sum of squared differences  $D$  for each order.

```
D = []
for i, order in enumerate(orders):
    y = np.polyval(coefficients[i][::-1], X_a)
    D.append(np.sum((F_a - y) ** 2))
```

Let's plot the sum of squared differences as a function of the order of the polynomial.

```
fig, ax = plt.subplots()
p = ax.plot(orders, D, '-b')
ax.set_xlabel('Order of polynomial')
ax.set_ylabel(r'$D(a_0, a_1, \cdots, a_m)$')
ax.set_xticks(orders)
plt.show()
```

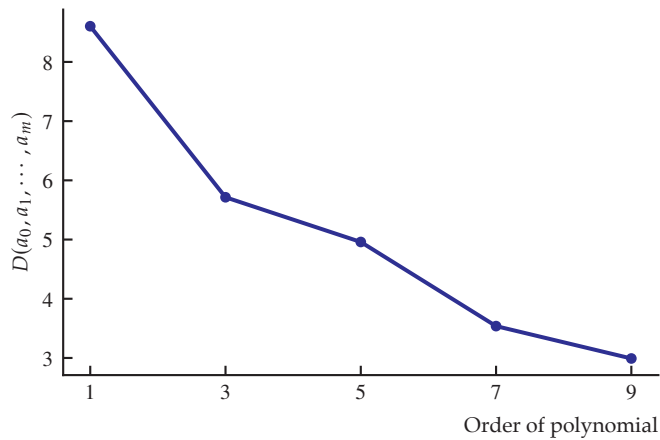


Figure 4.15. Sum of squared differences as a function of polynomial order.

The plot shows that the sum of squared differences decreases with the order of the polynomial. However, the decrease is less pronounced for higher-order polynomials. This suggests that the higher-order polynomials are overfitting the data. The optimal order of the polynomial is the one that gives the best fit without overfitting.

## 4.6 Problems



**Problem 4.1** **BREW** You need to know the duration of time a certain stage of a brewing process takes. You set up an automated test environment that repeats the test 100 times, recorded in the following JSON<sup>1</sup> data file: <https://math.ricopic.one/bt>. Perform the following analysis.

- Download and parse the JSON file (it contains a single array).
- Estimate the duration of the process from the sample.
- Choose and justify an assumed probability density function for the random variable duration.
- Use this PDF model to compute a 99 percent confidence interval for your duration estimate.
- Compute your duration confidence interval for the range of confidence values [85, 99.99] percent.<sup>2</sup>
- Plot the confidence intervals over the range of confidence in said intervals.

**Problem 4.2** **LABORATORIUM** Use linear regression techniques to find the values of  $a$ ,  $b$ ,  $c$ , and  $d$ , in a cubic function of the form,

$$f(x) = ax^3 + bx^2 + cx + d,$$

using the data below.

$x$	$f(x)$
-2.0	-4.7
-1.5	-1.9
-1.0	1.5
-0.5	1.5
0.0	1.4
0.6	0.3
1.1	-1.5
1.6	0.0
2.1	0.6
2.6	4.2

1. JSON is a simple and common programming language-independent data format. For parsing it with Matlab, see `jsondecode` here: <https://math.ricopic.one/75>. For parsing it with Python, see the module `json` here: <https://math.ricopic.one/jb>.

2. Consider using a  $z$ - or  $t$ -score inverse CDF lookup function like `t.ppf` from `scipy.stats`.

**Problem 4.3** 🤖ROBOTIZATION Use linear regression techniques to find the value of  $\tau$  in the function,

$$f(t) = 1 - e^{-\frac{t^2}{\tau}}$$

Using the data below.

$t$	$f(t)$
0.1	0.02
0.6	0.34
1.1	0.74
1.6	0.94
2.1	0.98

**Problem 4.4** 🧑TIRED There are 7 students enrolled in MME 502. If every week 5 students comes to class, for how many weeks could a unique set of 5 students come to class?

**Problem 4.5** 🤪STRANGE Use linear regression techniques to find  $\omega$  and  $\phi$  in the function,  $f(t) = \sin(\omega t + \phi)$  using the data below.

$t$	$f(t)$
0.0	0.53
0.1	0.73
0.2	0.91
0.5	0.92
0.6	0.83
0.7	0.65
0.8	0.42
0.9	0.15
1.0	-0.1
1.1	-0.35
1.2	-0.58
1.3	-0.82
1.4	-0.91
1.7	-0.86
1.8	-0.76
1.9	-0.54

Note: there are an infinite number of solutions to the inverse sine function,  $\sin^{-1}(x) = \pm y + n\pi$  where  $n \in \mathbb{Z}$ . You will have to utilize this definition to get your data in a linear form for fitting.

**Problem 4.6** 📄 The steady-state temperature  $T$  of steam at the outlet of a pipe was measured with a probe. The number of samples  $M$  was 20 and the size  $N$  of each sample was 100. The data can be downloaded at [ricopic.one/mathematical\\_foundations/source/dedicated.json](https://ricopic.one/mathematical_foundations/source/dedicated.json). Download the data and put it in your working directory for analysis.

Estimate the sample mean  $\bar{T}$ , sample variance  $S_T$ , and a 99 percent confidence interval for your estimation of the mean. The data is a list of lists of dimension  $N \times M$  (100-by-20). If you are using Python, load the data with

```
import numpy as np
import json
f = open('dedicated.json',)
Tdata = np.array(json.load(f))
print(f'data excerpt:\n{Tdata[0:3,0:4]}')

data excerpt:
[[241.97683415 213.94281418 220.42666213 225.29771168]
 [241.45909599 222.00636196 214.96718074 246.37793887]
 [230.92953215 217.66017678 227.38480454 210.21311645]]
```





# 5 Vector Calculus



A great many physical situations of interest to engineers can be described by **calculus**. It can describe how quantities continuously change over (say) time and gives tools for computing other quantities. We assume familiarity with the fundamentals of calculus: **limit**, **series**, **derivative**, and **integral**. From these and a basic grasp of vectors, we will outline some of the highlights of **vector calculus**. Vector calculus is particularly useful for describing the physics of, for instance, the following.

**mechanics of particles** wherein is studied the motion of particles and the forcing causes thereof

**rigid-body mechanics** wherein is studied the motion, rotational and translational, and its forcing causes, of bodies considered rigid (undeformable)

**solid mechanics** wherein is studied the motion and deformation, and their forcing causes, of continuous solid bodies (those that retain a specific resting shape)

**fluid mechanics** wherein is studied the motion and its forcing causes of fluids (liquids, gases, plasmas)

**heat transfer** wherein is studied the movement of thermal energy through and among bodies

**electromagnetism** wherein is studied the motion and its forcing causes of electrically charged particles

This last example was in fact very influential in the original development of both vector calculus and **complex analysis**.<sup>1</sup> It is not an exaggeration to say that the topics above comprise the majority of physical topics of interest in engineering.

A good introduction to vector calculus is given by (Kreyszig 2011; Chapters 9–10). Perhaps the most famous and enjoyable treatment is given by (Schey 2005) in the adorably titled *Div, Grad, Curl and All that*.

It is important to note that in much of what follows, we will describe (typically the three-dimensional space of our lived experience) as a **euclidean vector space**:

1. For an introduction to complex analysis, see (Kreyszig 2011; Part D).

an  $n$ -dimensional vector space isomorphic to  $\mathbb{R}^n$ . As we know from linear algebra, any vector  $v \in \mathbb{R}^n$  can be expressed in any number of **bases**. That is, the vector  $v$  is a basis-free object with multiple basis representations. The **components** and **basis vectors** of a vector change with basis changes, but the vector itself is **invariant**. A **coordinate system** is in fact *just a basis*. We are most familiar, of course, with **Cartesian coordinates**, which is the specific orthonormal basis  $\mathbf{b}$  for  $\mathbb{R}^n$ :

$$\mathbf{b}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots, \quad \mathbf{b}_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

**Manifolds** are spaces that appear locally as  $\mathbb{R}^n$ , but can be globally rather different and can describe **non-euclidean geometry** wherein euclidean geometry's **parallel postulate** is invalid. Calculus on manifolds is the focus of **differential geometry**, a subset of which we can consider our current study. A motivation for further study of differential geometry is that it is very convenient when dealing with advanced applications of mechanics, such as rigid-body mechanics of robots and vehicles. A very nice mathematical introduction is given by (Lee 2012) and (Bullo and Lewis 2005) give a compact presentation in the context of robotics.

Vector fields have several important properties of interest we'll explore in this chapter. Our goal is to gain an intuition of these properties and be able to perform basic calculation.

## 5.1 Divergence, Surface Integrals, and Flux

### 5.1.1 Flux and Surface Integrals

Consider a surface  $S$ . Let  $\mathbf{r}(u, v) = [x(u, v), y(u, v), z(u, v)]$  be a parametric position vector on a Euclidean vector space  $\mathbb{R}^3$ . Furthermore, let  $\mathbf{F} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$  be a vector-valued function of  $\mathbf{r}$  and let  $\mathbf{n}$  be a unit-normal vector on a surface  $S$ . The **surface integral**

$$\iint_S \mathbf{F} \cdot \mathbf{n} \, dS \tag{5.1}$$

which integrates the normal of  $\mathbf{F}$  over the surface. We call this quantity the **flux** of  $\mathbf{F}$  out of the surface  $S$ . This terminology comes from fluid flow, for which the flux is the mass flow rate out of  $S$ . In general, the flux is a measure of a quantity (or field) passing through a surface. For more on computing surface integrals, see Schey (2005; pp. 21-30) and Kreyszig (2011; § 10.6).



### 5.1.2 Continuity

Consider the flux out of a surface  $S$  that encloses a volume  $\Delta V$ , divided by that volume:

$$\frac{1}{\Delta V} \iint_S \mathbf{F} \cdot \mathbf{n} \, dS. \quad (5.2)$$

This gives a measure of flux per unit volume for a volume of space. Consider its physical meaning when we interpret this as fluid flow: all fluid that enters the volume is negative flux and all that leaves is positive. If physical conditions are such that we expect no fluid to enter or exit the volume via what is called a **source** or a **sink**, and if we assume the density of the fluid is uniform (this is called an **incompressible** fluid), then all the fluid that enters the volume must exit and we get

$$\frac{1}{\Delta V} \iint_S \mathbf{F} \cdot \mathbf{n} \, dS = 0. \quad (5.3)$$

This is called a **continuity equation**, although typically this name is given to equations of the form in the next section. This equation is one of the governing equations in continuum mechanics.

### 5.1.3 Divergence

Let's take the flux-per-volume as the volume  $\Delta V \rightarrow 0$  we obtain the following.

#### Equation 5.4 divergence: integral form

$$\lim_{\Delta V \rightarrow 0} \frac{1}{\Delta V} \iint_S \mathbf{F} \cdot \mathbf{n} \, dS.$$

This is called the **divergence** of  $\mathbf{F}$  and is defined at each point in  $\mathbb{R}^3$  by taking the volume to zero about it. It is given the shorthand  $\text{div } \mathbf{F}$ .

What interpretation can we give this quantity? It is a measure of the vector field's flux outward through a surface containing an infinitesimal volume. When we consider a fluid, a positive divergence is a local decrease in density and a negative divergence is a density increase. If the fluid is incompressible and has no sources or sinks, we can write the continuity equation

$$\text{div } \mathbf{F} = 0. \quad (5.5)$$

In the Cartesian basis, it can be shown that the divergence is easily computed from the field

$$\mathbf{F} = F_x \hat{\mathbf{i}} + F_y \hat{\mathbf{j}} + F_z \hat{\mathbf{k}} \quad (5.6)$$

as follows.

#### Equation 5.7 divergence: differential form

$$\operatorname{div} \mathbf{F} = \partial_x F_x + \partial_y F_y + \partial_z F_z$$

### 5.1.4 Exploring Divergence

Divergence is perhaps best explored by considering it for a vector field in  $\mathbb{R}^2$ . Such a field  $\mathbf{F} = F_x \hat{\mathbf{i}} + F_y \hat{\mathbf{j}}$  can be represented as a “quiver” plot. If we overlay the quiver plot over a “color density” plot representing  $\operatorname{div} \mathbf{F}$ , we can increase our intuition about the divergence.

First, load some Python packages.

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from matplotlib.ticker import LogLocator
from matplotlib.colors import *
from sympy.utilities.lambdify import lambdify
```

Now we define some symbolic variables and functions.

```
x = sp.Symbol('x', real=True)
y = sp.Symbol('y', real=True)
F_x = sp.Function('F_x')(x, y)
F_y = sp.Function('F_y')(x, y)
```

Rather than repeat code, let’s write a single function `quiver_plotter_2D()` to make several of these plots.

```
def quiver_plotter_2D(
    field={},
    grid_width=3, grid_decimate_x=8, grid_decimate_y=8,
    norm=Normalize(), density_operation='div',
    print_density=True):
    x, y = sp.symbols('x y', real=True)
    F_x, F_y = sp.Function('F_x')(x, y), sp.Function('F_y')(x, y)
    field_sub = field
    # Calculate density
    den = F_x.diff(x) + F_y.diff(y) if density_operation == 'div' else None
    if den is None:
```

```

        raise ValueError(f'Unknown density operation: {density_operation}')
    den_simp = den.subs(field_sub).doit().simplify()
    if den_simp.is_constant():
        print('Warning: density operator is constant (no density plot)')
    if print_density:
        print(f'The {density_operation} is:')
        print(den_simp)
    # Lambdify for numerics
    F_x_sub = F_x.subs(field_sub)
    F_y_sub = F_y.subs(field_sub)
    F_x_fun = sp.lambdify((x, y), F_x_sub, 'numpy')
    F_y_fun = sp.lambdify((x, y), F_y_sub, 'numpy')
    if F_x_sub.is_constant():
        F_x_fun1 = F_x_fun # Dummy
        F_x_fun = lambda x, y: F_x_fun1(x, y) * np.ones(x.shape)
    if F_y_sub.is_constant():
        F_y_fun1 = F_y_fun # Dummy
        F_y_fun = lambda x, y: F_y_fun1(x, y) * np.ones(x.shape)
    if not den_simp.is_constant():
        den_fun = sp.lambdify((x, y), den_simp, 'numpy')
    # Create grid
    w = grid_width
    Y, X = np.mgrid[-w:w:100j, -w:w:100j]
    # Evaluate numerically
    F_x_num = F_x_fun(X, Y)
    F_y_num = F_y_fun(X, Y)
    if not den_simp.is_constant():
        den_num = den_fun(X, Y)
    # Plot
    fig, ax = plt.subplots()
    if not den_simp.is_constant():
        cmap = plt.get_cmap('coolwarm')
        im = plt.pcolormesh(X, Y, den_num, cmap=cmap, norm=norm)
        plt.colorbar()
    dx, dy = grid_decimate_y, grid_decimate_x
    plt.quiver(X[::dx, ::dy], Y[::dx, ::dy], F_x_num[::dx, ::dy],
               F_y_num[::dx, ::dy], units='xy', scale=10)
    plt.title(fr'$F(x, y) = \left[ \text{{sp.latex(F_x.subs(field_sub))}} \right]$, ' +
              fr'$\text{{sp.latex(F_y.subs(field_sub))}} \right]$')
    return fig, ax

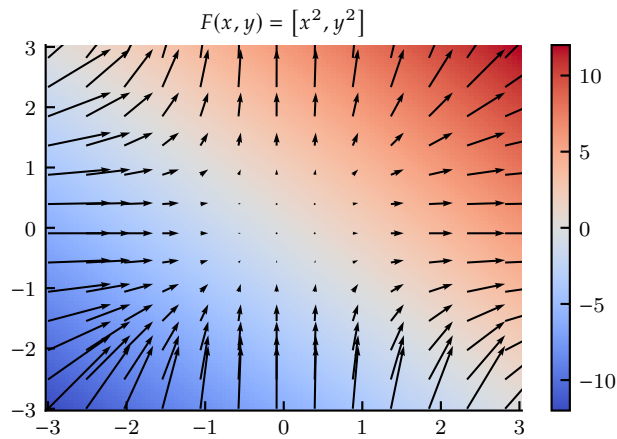
```

Let's inspect several cases.

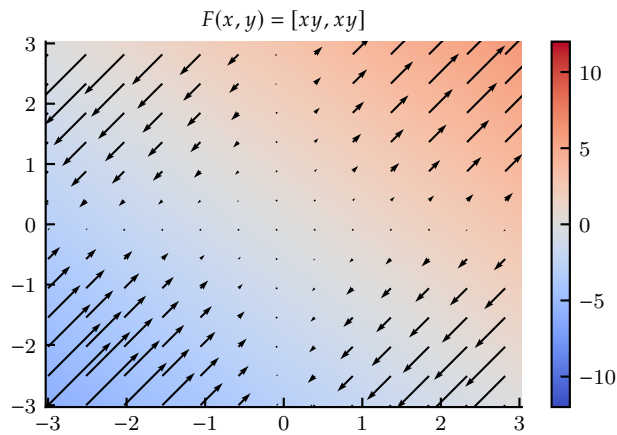
```

fig, ax = quiver_plotter_2D(field={F_x: x**2, F_y: y**2})
plt.draw()

```

Figure 5.1. Quiver plot of  $F(x, y) = [x^2, y^2]$ 

```
fig, ax = quiver_plotter_2D(field={F_x: x*y, F_y: x*y})
plt.draw()
```

Figure 5.2. Quiver plot of  $F(x, y) = [xy, xy]$ 

```
fig, ax = quiver_plotter_2D(field={F_x: x**2 + y**2, F_y: x**2 + y**2})
plt.draw()
```

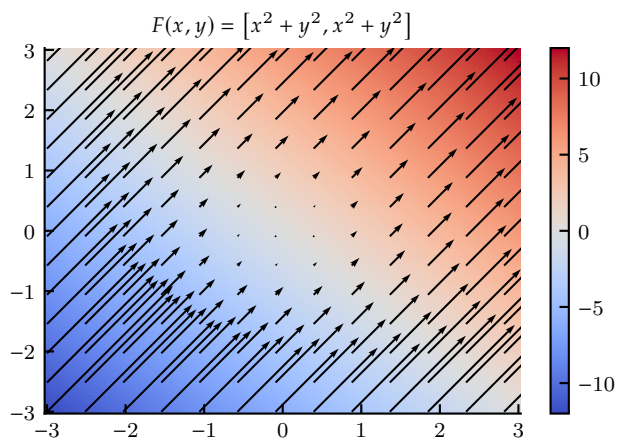


Figure 5.3. Quiver plot of  $F(x, y) = [x^2 + y^2, x^2 + y^2]$

```
fig, ax = quiver_plotter_2D(
    field={F_x: x**2/sp.sqrt(x**2+y**2), F_y: y**2/sp.sqrt(x**2+y**2)},
    norm=SymLogNorm(linthresh=.3, linscale=.3)
)
plt.show()
```

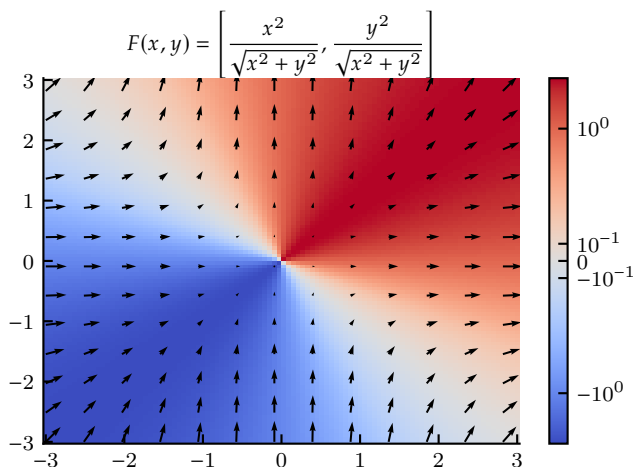


Figure 5.4. Quiver plot of  $F(x, y) = \left[ \frac{x^2}{\sqrt{x^2 + y^2}}, \frac{y^2}{\sqrt{x^2 + y^2}} \right]$

## 5.2 Curl, Line Integrals, and Circulation

### 5.2.1 Line Integrals

Consider a curve  $C$  in a Euclidean vector space  $\mathbb{R}^3$ . Let  $\mathbf{r}(t) = [x(t), y(t), z(t)]$  be a parametric representation of  $C$ . Furthermore, let  $F: \mathbb{R}^3 \rightarrow \mathbb{R}^3$  be a vector-valued function of  $\mathbf{r}$  and let  $\mathbf{r}'(t)$  be the tangent vector. The **line integral** is

$$\int_C \mathbf{F}(\mathbf{r}(t)) \cdot \mathbf{r}'(t) dt \quad (5.8)$$

which integrates  $F$  along the curve. For more on computing line integrals, see (Schey 2005; pp. 63-74) and (Kreyszig 2011; § 10.1 and 10.2).

If  $F$  is a **force** being applied to an object moving along the curve  $C$ , the line integral is the **work** done by the force. More generally, the line integral integrates  $F$  along the tangent of  $C$ .





### 5.2.2 Circulation

Consider the line integral over a closed curve  $C$ , denoted by

$$\oint_C \mathbf{F}(\mathbf{r}(t)) \cdot \mathbf{r}'(t) dt. \quad (5.9)$$

We call this quantity the **circulation** of  $\mathbf{F}$  around  $C$ .

For certain vector-valued functions  $\mathbf{F}$ , the circulation is zero for every curve. In these cases (static electric fields, for instance), this is sometimes called the **the law of circulation**.

### 5.2.3 Curl

Consider the division of the circulation around a curve in  $\mathbb{R}^3$  by the surface area it encloses  $\Delta S$ ,

$$\frac{1}{\Delta S} \oint_C \mathbf{F}(\mathbf{r}(t)) \cdot \mathbf{r}'(t) dt. \quad (5.10)$$

In a manner analogous to the operation that gives us the divergence, let's consider shrinking this curve to a point and the surface area to zero,

$$\lim_{\Delta S \rightarrow 0} \frac{1}{\Delta S} \oint_C \mathbf{F}(\mathbf{r}(t)) \cdot \mathbf{r}'(t) dt. \quad (5.11)$$

We call this quantity the “scalar” **curl** of  $\mathbf{F}$  at each point in  $\mathbb{R}^3$  *in the direction normal to  $\Delta S$*  as it shrinks to zero. Taking three (or  $n$  for  $\mathbb{R}^n$ ) “scalar” curls in independent normal directions (enough to span the vector space), we obtain the **curl** proper, which is a vector-valued function  $\text{curl}: \mathbb{R}^3 \rightarrow \mathbb{R}^3$ .

The curl is coordinate-independent. In cartesian coordinates, it can be shown to be equivalent to the following.

**Equation 5.12** curl: differential form, cartesian coordinates

$$\text{curl } \mathbf{F} = [\partial_y F_z - \partial_z F_y \quad \partial_z F_x - \partial_x F_z \quad \partial_x F_y - \partial_y F_x]^\top$$

But what does the curl of  $\mathbf{F}$  represent? It quantifies the local rotation of  $\mathbf{F}$  about each point. If  $\mathbf{F}$  represents a fluid's velocity,  $\text{curl } \mathbf{F}$  is the local rotation of the fluid about each point and it is called the **vorticity**.

## 5.2.4 Zero Curl, Circulation, and Path Independence

**5.2.4.1 Circulation** It can be shown that if the circulation of  $F$  on all curves is zero, then in each direction  $\mathbf{n}$  and at every point  $\text{curl } F = 0$  (i.e.  $\mathbf{n} \cdot \text{curl } F = 0$ ). Conversely, for  $\text{curl } F = 0$  in a simply connected region<sup>2</sup>,  $F$  has zero circulation.

Succinctly, informally, and without the requisite qualifiers above,

$$\text{zero circulation} \Rightarrow \text{zero curl} \quad (5.13)$$

$$\text{zero curl} + \text{simply connected region} \Rightarrow \text{zero circulation}. \quad (5.14)$$

**5.2.4.2 Path Independence** It can be shown that if the path integral of  $F$  on all curves between any two points is **path-independent**, then in each direction  $\mathbf{n}$  and at every point  $\text{curl } F = 0$  (i.e.  $\mathbf{n} \cdot \text{curl } F = 0$ ). Conversely, for  $\text{curl } F = 0$  in a simply connected region, all line integrals are independent of path.

Succinctly, informally, and without the requisite qualifiers above,

$$\text{path independence} \Rightarrow \text{zero curl} \quad (5.15)$$

$$\text{zero curl} + \text{simply connected region} \Rightarrow \text{path independence}. \quad (5.16)$$

**5.2.4.3 And How They Relate** It is also true that

$$\text{path independence} \Leftrightarrow \text{zero circulation}. \quad (5.17)$$

So, putting it all together, we get figure 5.5.

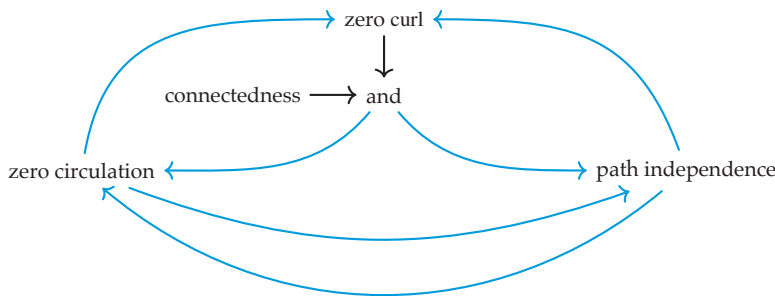


Figure 5.5. An implication graph relating zero curl, zero circulation, path independence, and connectedness. Blue edges represent implication ( $a$  implies  $b$ ) and black edges represent logical conjunctions.

2. A region is simply connected if every curve in it can shrink to a point without leaving the region. An example of a region that is not simply connected is the surface of a toroid.

### 5.2.5 Exploring Curl

Curl is perhaps best explored by considering it for a vector field in  $\mathbb{R}^2$ . Such a field in cartesian coordinates  $F = F_x \hat{i} + F_y \hat{j}$  has curl

$$\begin{aligned}\text{curl } F &= \begin{bmatrix} \partial_y 0 - \partial_z F_y & \partial_z F_x - \partial_x 0 & \partial_x F_y - \partial_y F_x \end{bmatrix}^\top \\ &= \begin{bmatrix} 0 - 0 & 0 - 0 & \partial_x F_y - \partial_y F_x \end{bmatrix}^\top \\ &= \begin{bmatrix} 0 & 0 & \partial_x F_y - \partial_y F_x \end{bmatrix}^\top.\end{aligned}\tag{5.18}$$

That is,  $\text{curl } F = (\partial_x F_y - \partial_y F_x) \hat{k}$  and the only nonzero component is normal to the  $xy$ -plane. If we overlay a quiver plot of  $F$  over a “color density” plot representing the  $\hat{k}$ -component of  $\text{curl } F$ , we can increase our intuition about the curl. First, load some Python packages.

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from matplotlib.ticker import LogLocator
from matplotlib.colors import *
```

Now we define some symbolic variables and functions.

```
x = sp.Symbol('x', real=True)
y = sp.Symbol('y', real=True)
F_x = sp.Function('F_x')(x, y)
F_y = sp.Function('F_y')(x, y)
```

We use a variation of the `quiver_plotter_2D()` from above to make several of these plots.

```
def quiver_plotter_2D(
    field={F_x: x*y, F_y: x*y},
    grid_width=3,
    grid_decimate_x=8,
    grid_decimate_y=8,
    norm=Normalize(),
    density_operation='div',
    print_density=True,
):
    # Define symbolics
    x, y = sp.symbols('x y', real=True)
    F_x = sp.Function('F_x')(x, y)
    F_y = sp.Function('F_y')(x, y)
    field_sub = field
    # Compute density
    if density_operation == 'div':
        den = F_x.diff(x) + F_y.diff(y)
```

```

elif density_operation == 'curl':
    den = F_y.diff(x) - F_x.diff(y) # in the k direction
else:
    raise ValueError('div and curl are the only density operators')
den_simp = den.subs(field_sub).doit().simplify()
if den_simp.is_constant():
    print('Warning: density operator is constant (no density plot)')
if print_density:
    print(f'The {density_operation} is: {den_simp}')
# Lambdify for numerics
F_x_sub = F_x.subs(field_sub)
F_y_sub = F_y.subs(field_sub)
F_x_fun = sp.lambdify((x, y), F_x_sub, 'numpy')
F_y_fun = sp.lambdify((x, y), F_y_sub, 'numpy')
if F_x_sub.is_constant():
    F_x_fun1 = F_x_fun # Dummy
    F_x_fun = lambda x, y: F_x_fun1(x, y)*np.ones(x.shape)
if F_y_sub.is_constant():
    F_y_fun1 = F_y_fun # Dummy
    F_y_fun = lambda x, y: F_y_fun1(x, y)*np.ones(x.shape)
if not den_simp.is_constant():
    den_fun = sp.lambdify((x, y), den_simp, 'numpy')
# Create grid
w = grid_width
Y, X = np.mgrid[-w:w:100j, -w:w:100j]
# Evaluate numerically
F_x_num = F_x_fun(X, Y)
F_y_num = F_y_fun(X, Y)
if not den_simp.is_constant():
    den_num = den_fun(X, Y)
# Plot
fig, ax = plt.subplots()
if not den_simp.is_constant():
    cmap = plt.get_cmap('coolwarm')
    im = plt.pcolormesh(X, Y, den_num, cmap=cmap, norm=norm)
    plt.colorbar()
dx = grid_decimate_y
dy = grid_decimate_x
plt.quiver(
    X[::dx, ::dy], Y[::dx, ::dy],
    F_x_num[::dx, ::dy], F_y_num[::dx, ::dy],
    units='xy', scale=10)
plt.title(fr'$F(x, y) = \left[ \text{{sp.latex(F_x.subs(field_sub))}} \right] + \backslash$
          fr'$\text{{sp.latex(F_y.subs(field_sub))}} \right]$')
return fig, ax

```

Let's inspect several cases.

```
fig, ax = quiver_plotter_2D(
    field={F_x: 0, F_y: sp.cos(2*sp.pi*x)}, density_operation='curl',
    grid_decimate_x=2, grid_decimate_y=10, grid_width=1
)
plt.draw()
```

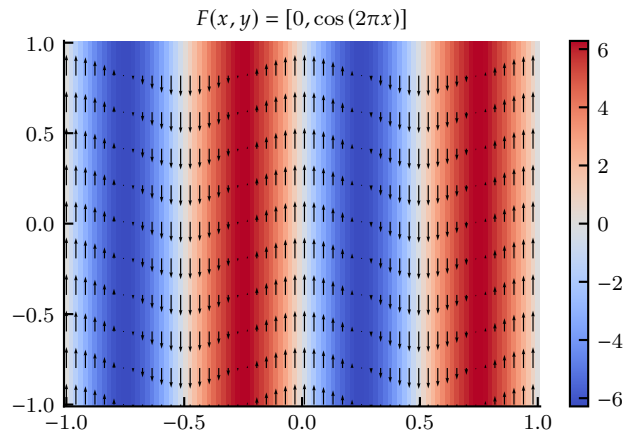
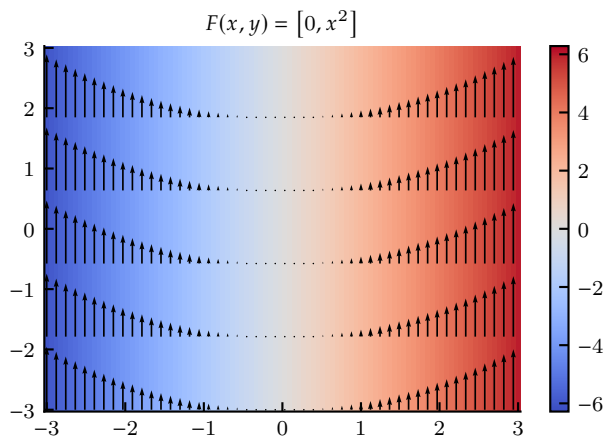
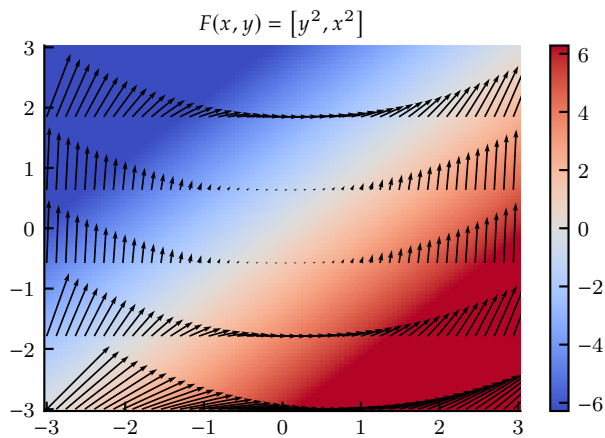


Figure 5.6. Quiver plot of  $F(x, y) = [0, \cos(2\pi x)]$

```
fig, ax = quiver_plotter_2D(
    field={F_x: 0, F_y: x**2}, density_operation='curl',
    grid_decimate_x=2, grid_decimate_y=20,
)
plt.draw()
```

Figure 5.7. Quiver plot of  $F(x, y) = [0, x^2]$ 

```
fig, ax = quiver_plotter_2D(
    field={F_x: y**2, F_y: x**2}, density_operation='curl',
    grid_decimate_x=2, grid_decimate_y=20,
)
plt.draw()
```

Figure 5.8. Quiver plot of  $F(x, y) = [y^2, x^2]$

```
fig, ax = quiver_plotter_2D(
    field={F_x: -y, F_y: x}, density_operation='curl',
    grid_decimate_x=6, grid_decimate_y=6,
)
plt.show()
```

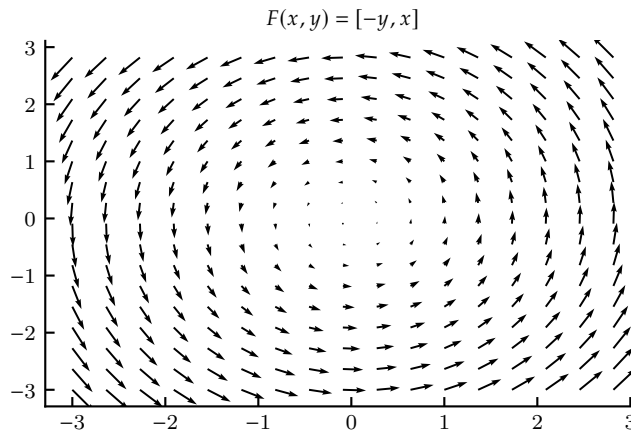


Figure 5.9. Quiver plot of  $F(x, y) = [-y, x]$

## 5.3 Gradient

### 5.3.1 Gradient

The **gradient**  $\text{grad}$  of a scalar-valued function  $f: \mathbb{R}^3 \rightarrow \mathbb{R}$  is a vector field  $F: \mathbb{R}^3 \rightarrow \mathbb{R}^3$ ; that is,  $\text{grad } f$  is a vector-valued function on  $\mathbb{R}^3$ . The gradient's local **direction** and **magnitude** are those of the local maximum rate of increase of  $f$ . This makes it useful in optimization (e.g., in the method of gradient descent).

In classical mechanics, quantum mechanics, relativity, string theory, thermodynamics, and continuum mechanics (and elsewhere) the **principle of least action** is taken as fundamental (Feynman, Leighton, and Sands 2010). This principle tells us that nature's laws quite frequently seem to be derivable by assuming a certain quantity—called *action*—is minimized. Considering, then, that the gradient supplies us with a tool for optimizing functions, it is unsurprising that the gradient enters into the equations of motion of many physical quantities.

The gradient is coordinate-independent, but its coordinate-free definitions don't add much to our intuition.



4J

**Equation 5.19   gradient: cartesian coordinates**

$$\text{grad } f = [\partial_x f \quad \partial_y f \quad \partial_z f]^\top$$

### 5.3.2 Vector Fields from Gradients Are Special

Although all gradients are vector fields, not all vector fields are gradients. That is, given a vector field  $F$ , it may or may not be equal to the gradient of any scalar-valued function  $f$ . Let's say of a vector field that is a gradient that it has **gradient**.<sup>3</sup> Those vector fields that *are* gradients have special properties. Surprisingly, those properties are connected to path independence and curl. It can be shown that iff a field is a gradient, line integrals of the field are path independent. That is, for a vector field,

$$\text{gradient} \Leftrightarrow \text{path independence.} \quad (5.20)$$

Considering what we know from section 5.2 about path independence we can expand figure 5.5 to obtain figure 5.10.

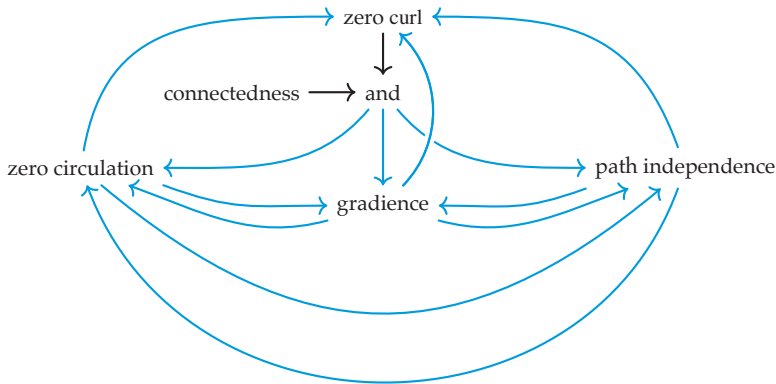


Figure 5.10. An implication graph relating gradient, zero curl, zero circulation, path independence, and connectedness. Green edges represent implication ( $a$  implies  $b$ ) and black edges represent logical conjunctions.

3. This is nonstandard terminology, but we're bold.



One implication is that *gradients have zero curl!* Many important fields that describe physical interactions (e.g., static electric fields, Newtonian gravitational fields) are gradients of scalar fields called **potentials**.

### 5.3.3 Exploring Gradient

Gradient is perhaps best explored by considering it for a scalar field on  $\mathbb{R}^2$ . Such a field in cartesian coordinates  $f(x, y)$  has gradient

$$\text{grad } f = [\partial_x f \quad \partial_y f]^\top \quad (5.21)$$

That is,  $\text{grad } f = F = \partial_x f \hat{i} + \partial_y f \hat{j}$ . If we overlay a quiver plot of  $F$  over a “color density” plot representing the  $f$ , we can increase our intuition about the gradient.

First, load some Python packages.

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from matplotlib.ticker import LogLocator
from matplotlib.colors import *
```

Now we define some symbolic variables and functions.

```
x, y = sp.symbols('x y', real=True)
```

Rather than repeat code, let’s write a single function `grad_plotter_2D()` to make several of these plots.

```
def grad_plotter_2D(
    field=x*y, grid_width=3, grid_decimate_x=8, grid_decimate_y=8,
    norm=None, # Density plot normalization
    scale=None, # Arrow length scale (auto)
    print_vector=True, mask=False, # Mask vector lengths
):
    # Define symbolics
    x, y = sp.symbols('x y', real=True)
    field = sp.sympify(field)
    # Compute vector field
    F_x = field.diff(x).simplify()
    F_y = field.diff(y).simplify()
    if field.is_constant():
        print('Warning: field is constant (no plot)')
    if print_vector:
        print(f'The gradient is:')
        print(sp.Array([F_x, F_y]))
    # Lambdify for numerics
    F_x_fun = sp.lambdify((x, y), F_x, 'numpy')
    F_y_fun = sp.lambdify((x, y), F_y, 'numpy')
    if F_x.is_constant:
```

```

    F_x_fun1 = F_x_fun # Dummy
    F_x_fun = lambda x, y: F_x_fun1(x, y) * np.ones(x.shape)
    if F_y.is_constant:
        F_y_fun1 = F_y_fun # Dummy
        F_y_fun = lambda x, y: F_y_fun1(x, y) * np.ones(x.shape)
    if not field.is_constant():
        den_fun = sp.lambdify((x, y), field, 'numpy')
    # Create grid
    w = grid_width
    Y, X = np.mgrid[-w:w:100j, -w:w:100j]
    # Evaluate numerically
    F_x_num = F_x_fun(X, Y)
    F_y_num = F_y_fun(X, Y)
    if not field.is_constant():
        den_num = den_fun(X, Y)
    # Mask F_x and F_y
    if mask:
        masking_a = np.sqrt(np.square(F_x_num) + np.square(F_y_num))
        F_x_num = np.ma.masked_where(masking_a > w / 5., F_x_num)
        F_y_num = np.ma.masked_where(masking_a > w / 5., F_y_num)
    # Plot
    if not field.is_constant():
        fig, ax = plt.subplots()
        cmap = plt.get_cmap('coolwarm')
        im = plt.pcolormesh(X, Y, den_num, cmap=cmap, norm=norm)
        plt.colorbar()
        dx = grid_decimate_y
        dy = grid_decimate_x
        plt.quiver(
            X[::dx, ::dy], Y[::dx, ::dy],
            F_x_num[::dx, ::dy], F_y_num[::dx, ::dy],
            units='xy', scale=scale
        )
        plt.title(f'$f(x,y) = {sp.latex(field)}$')
    return fig, ax
return 1, 1

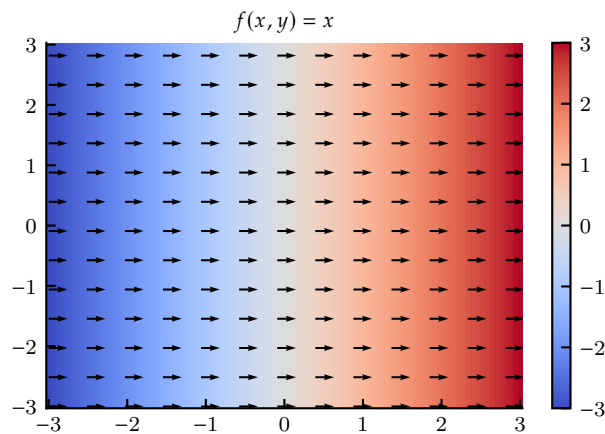
```

Let's inspect several cases. While considering the following plots, remember that they all have zero curl!

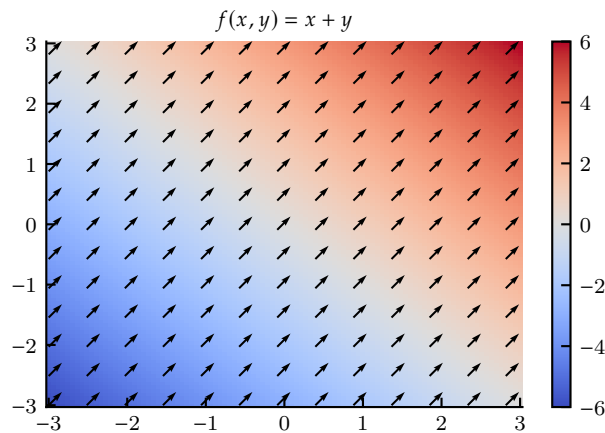
```

fig, ax = grad_plotter_2D(field=x)
plt.draw()

```

Figure 5.11. Gradient of  $f(x, y) = x$ 

```
fig, ax = grad_plotter_2D(field=x+y)
plt.draw()
```

Figure 5.12. Gradient of  $f(x, y) = x + y$ 

```
fig, ax = grad_plotter_2D(field=1)
```

**5.3.3.1 Gravitational Potential** Gravitational potentials have the form of  $1/\text{distance}$ . Let's check out the gradient.

```
fig, ax = grad_plotter_2D(
    field=1/sp.sqrt(x**2+y**2),
    norm=SymLogNorm(linthresh=.3, linscale=.3), mask=True,
)
plt.draw()
```

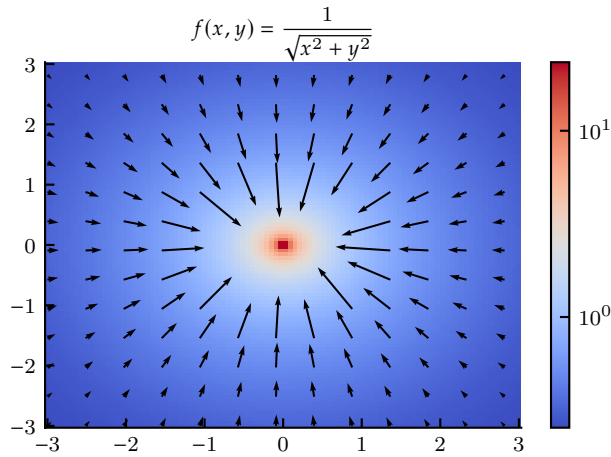


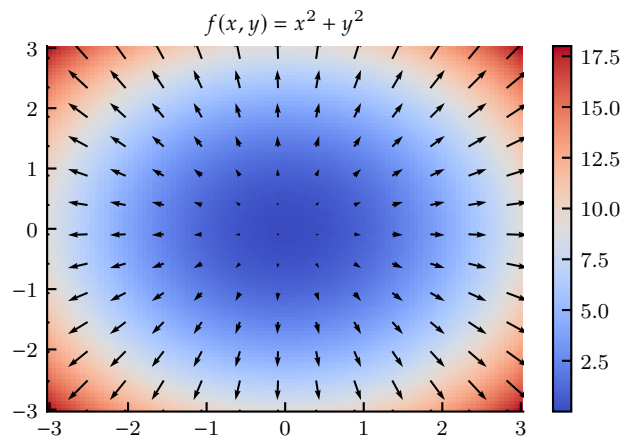
Figure 5.13. Gradient of  $f(x, y) = 1/\sqrt{x^2 + y^2}$

**5.3.3.2 Conic Section Fields** Gradients of **conic section** fields can be explored. The following is called a **parabolic field**.

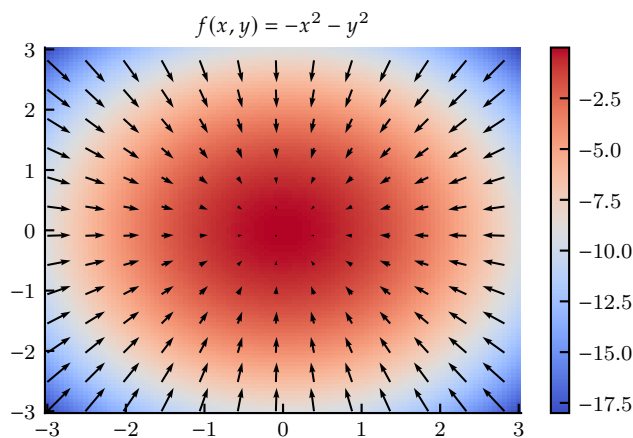
```
fig, ax = grad_plotter_2D(field=x**2)
plt.draw()
```

The following are called **elliptic fields**.

```
fig, ax = grad_plotter_2D(field=x**2 + y**2)
plt.draw()
```

Figure 5.14. Gradient of  $f(x, y) = x^2 + y^2$ 

```
fig, ax = grad_plotter_2D(field=-x**2 - y**2)
plt.draw()
```

Figure 5.15. Gradient of  $f(x, y) = -x^2 - y^2$ 

The following is called a **hyperbolic field**.

```
fig, ax = grad_plotter_2D(field=x**2 - y**2)
plt.show()
```

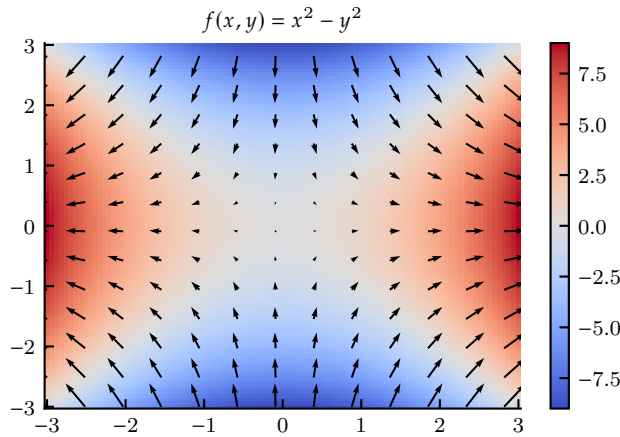


Figure 5.16. Gradient of  $f(x, y) = x^2 - y^2$

### 5.3.4 Conservative Force Fields

A force field is **conservative** if it is the gradient of a potential function. This means no energy is lost by a body acted on by a conservative force field, and that if the field is solely responsible for the body's trajectory through the field, if the body returns to its starting position, it has the same potential energy and kinetic energy with which it started. The two most common conservative force fields encountered in engineering are gravitational fields and static electric fields.

A conservative force field  $\mathbf{f}$  can be derived from its potential energy function  $U$  via the gradient

$$\mathbf{f} = -\text{grad } U.$$

This means the force field always points in the direction of greatest *decrease* of potential energy, known colloquially as “downhill.”

#### Example 5.1

In many engineering mechanics problems, a body experiences gravitational forcing from the Earth (or another planet). However, in many cases, the centers of mass of the body and the Earth do not change significantly, relative to their overall distance. In such cases, we approximate the gravitational potential energy to be linear. For a body with mass  $m$ , the gravitational potential energy is approximately

$$U_g = mgh,$$

where  $g$  is a constant acceleration ( $g \approx 9.81 \text{ m/s}^2$  on Earth's surface) and  $h$  is the vertical (i.e., with Earth downward) distance to some arbitrary reference position. Computing the force field  $\mathbf{f}_g$  in Cartesian coordinates with  $y$  the vertical coordinate pointed upward,

$$\mathbf{f}_g = -\text{grad } U_g = -mg\hat{j}.$$

This constant force is the familiar “force of gravity” we encounter in so many mechanics problems. This is a uniform field always pointing downward. Note that we not only ignore small changes in magnitude here, we also ignore small changes in direction because a more complete model of the gravitational field always points toward the center of the other mass.

With reference to figure 5.17, suppose a bead with mass  $m = 1 \text{ kg}$  is connected to a rigid circular hoop of radius  $R = 1 \text{ m}$ , oriented in the  $x$ - $y$  plane, with a constant gravitational force with acceleration  $g = 9.81 \text{ m/s}^2$  in the negative  $y$  direction. Parameterize the curve by its arclength and compute the trajectory of the bead from an initial position, assuming the hoop is frictionless. When the bead returns to its initial point, does it have the same potential energy? Now include a viscous damping force with damping coefficient  $b$ . Recompute the trajectory and comment on the nonconservative nature of the viscous damping force. As  $t \rightarrow \infty$ , how much work have each of the gravitational force field and damping force performed?

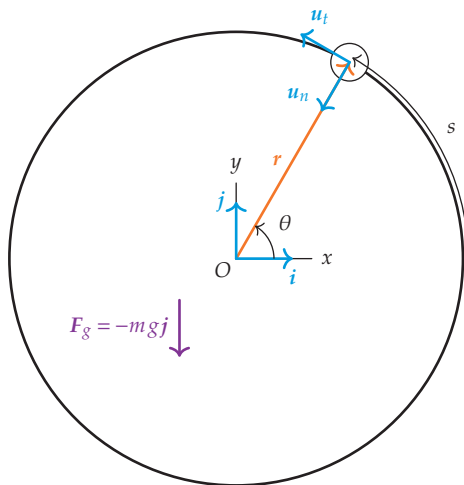


Figure 5.17. A bead on a hoop.

Our approach is to apply Newton's second law in a tangential and normal basis (TNB basis or Serret–Frenet frame) with unit vectors  $\mathbf{u}_t$  and  $\mathbf{u}_n$ , as shown in figure 5.17 (Hibbeler 2015; § 13.5). The advantage of using this coordinate system is that the variable (normal) force applied to the bead by the hoop is decoupled from the force along the (tangential) trajectory of the bead.

The position vector  $\mathbf{r}$  has its tail at the origin  $O$  of the Cartesian coordinate system and its head at the center of the bead. The easiest parameter to describe the circular curve with is the angle  $\theta$  from the positive  $x$  axis. This parametric representation of  $\mathbf{r}$  is, in Cartesian coordinates,

$$\mathbf{r}(\theta) = \begin{bmatrix} R \cos \theta \\ R \sin \theta \end{bmatrix}.$$

However, when working with a TNB basis, we should parameterize the position vector with the arclength  $s$  (O'Reilly 2019; § 3.1.2). In general, the process of changing parameters from parameter  $\theta$  to arclength  $s$  generally proceeds as follows. Compute the arclength  $s$  from its definition (Kreyszig 2011; p. 385), for some  $a \in \mathbb{R}$ ,

$$s = \int_a^\theta \sqrt{\mathbf{r}'(\tilde{\theta}) \cdot \mathbf{r}'(\tilde{\theta})} d\tilde{\theta},$$

then solve for  $\theta(s)$ . The arclength parameter  $s$  can now replace  $\theta$  by substitution,

$$\mathbf{r}(\theta) \rightarrow \mathbf{r}(\theta(s)) \rightarrow \mathbf{r}(s).$$

However in our case the (circular) arclength has the well-known relationship with the corresponding angle,

$$s = R\theta.$$

Solving for  $\theta(s)$ ,  $\theta(s) = s/R$ . So the arclength parameterization of the position vector is

$$\mathbf{r}(s) = \begin{bmatrix} R \cos(s/R) \\ R \sin(s/R) \end{bmatrix}.$$

From here, we proceed in Python. Begin by loading the necessary packages.

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp, cumulative_trapezoid
```

Define a symbolic parametric representation of the position vector  $\mathbf{r}(s)$  in Cartesian coordinates. In our code, we will use  $T$  instead of  $\theta$ .

```
R = sp.symbols("R", positive=True) # Radius of the hoop
s = sp.symbols("s", real=True) # Arc length
r_s = sp.Matrix([R * sp.cos(s/R), R * sp.sin(s/R)])
```



Now we can compute the gravitational force in the tangential direction. The gravitational force is  $F_g = -mg\hat{j}$ , where  $m$  is the mass of the bead. The unit tangent vector  $u_t$  can be conveniently computed from the arclength parameterization of  $r$  as


$$u_t = \frac{d\mathbf{r}}{ds} = \begin{bmatrix} -\sin(s/R) \\ \cos(s/R) \end{bmatrix}.$$

The gravitational force in the tangential direction is then

$$F_g(s) \cdot u_t = F_g \cdot \frac{d\mathbf{r}}{ds} = \begin{bmatrix} 0 \\ -mg \end{bmatrix} \cdot \begin{bmatrix} -\sin(s/R) \\ \cos(s/R) \end{bmatrix} = -mg \cos(s/R).$$

Computing this expression symbolically:

```
m, g = sp.symbols("m, g", positive=True) # Mass, grav. acceleration
F_g = sp.Matrix([0, -m * g]) # Grav. force (Cartesian coordinates)
u_t = r_s.diff(s) # Unit tangent vector (Cartesian coordinates)
# This has unit length because r_s is parameterized by arc length
F_g_t = F_g.dot(u_t) # Grav. force in the tangential direction
```

  $-gm \cos\left(\frac{s}{R}\right)$

Convert the symbolic expression to a Python function for numerical evaluation.

```
params = {R: 1, m: 1, g: 9.81} # Parameters
F_g_t_fun = sp.lambdify((s), F_g_t.subs(params), "numpy")
```

Apply Newton's second law to the mass to form a dynamic system state space model. The state vector is  $\begin{bmatrix} s & v \end{bmatrix}^\top$ , where  $s$  is the arc length and  $v$  is the velocity in the tangential direction. To define the state space model, we need to compute the time derivatives of the state variables. By definition, the velocity is the rate of change of arc length with respect to time, so

$$\frac{ds}{dt} = v.$$

So this is the first equation in our system of ODEs comprising the state space model. The tangential acceleration is defined as the rate of change of velocity with respect to time,

$$a = \frac{dv}{dt}.$$

Applying Newton's second law to the bead in the tangential direction, we have

$$ma = F_g(s) \cdot u_t = -mg \cos(s/R).$$

Substituting the definition of acceleration and the gravitational force, we have

$$\frac{dv}{dt} = -g \cos(s/R).$$

This is the second equation in our state space model. Define the state space model in Python:

```
def system(t, X, params):
    """Bead on a hoop dynamic system, sans damping."""
    s, v = X # Unpack the state vector
    ds_dt = v # Velocity in the tangential direction
    dv_dt = 1/m.subs(params) * F_g_t_fun(s)
        # Acceleration in the tangential direction
    return [ds_dt, dv_dt]
```

Numerically solve the system of ODEs for an initial state vector  $[s_0, v_0]$ .

```
X0 = [sp.pi/3, 0] # Initial state vector [s0, v0]
t = np.linspace(0, 10, 201) # Time array
X_t = solve_ivp(
    system, (t[0], t[-1]),
    X0, t_eval=t, args=(params,)
).y # Solve the system
```

Now write a function to plot the trajectory of the bead on the hoop through time.

```
def plot_trajectory(t, X_t):
    """Plot the trajectory of the bead on the hoop through time."""
    fig, ax = plt.subplots(2, 1, sharex=True)
    ax[0].plot(t, X_t[0])
    ax[0].set_ylabel("Arc length $s$ (m)")
    ax[1].plot(t, X_t[1])
    ax[1].set_ylabel("Velocity $v$ (m/s)")
    ax[1].set_xlabel("Time $t$ (s)")
    return fig
```

Plot the trajectory of the bead on the hoop through time.

```
fig = plot_trajectory(t, X_t)
plt.draw()
```

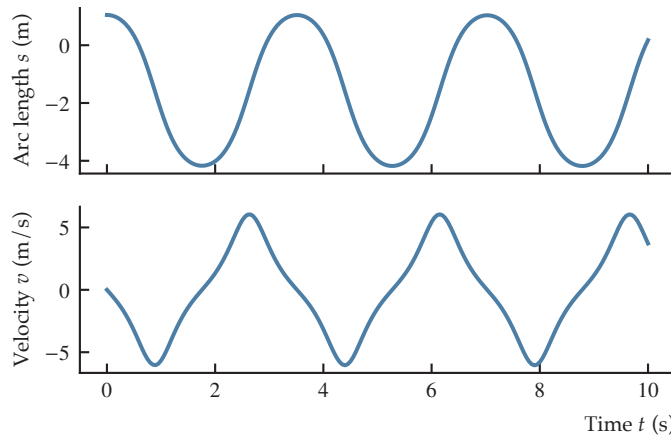


Figure 5.18. Arc length  $s$  and velocity  $v$  of the bead on the hoop through time without damping.

Now write a function to plot the trajectory of the bead on the hoop through space and time.

```
def plot_trajectory_space_time(t, X_t, r_s):
    """Plot the trajectory of the bead through space and time"""
    # First define a function to compute the position vector r(s)
    # from the state variable arclength s
    r_s_fun = sp.lambdify((s), r_s.subs(params), "numpy")
    r_s_t = r_s_fun(X_t[0]) # Position vector r(s) for each s in X_t
    fig = plt.figure()
    ax = fig.add_subplot(projection="3d")
    t_normalized = (t - np.min(t)) / (1.2*np.max(t) - np.min(t))
    for i in range(len(t) - 1): # Each segment with a different color
        ax.plot(
            r_s_t[0][0][i:i+2],
            r_s_t[1][0][i:i+2],
            [t[i], t[i+1]],
            color=plt.cm.viridis(t_normalized[i])
        )
    ticks = np.array([-1, 0, 1])*params[R]
    ax.set_xticks(ticks)
    ax.set_yticks(ticks)
    ax.set_xlabel("$x$ (m)")
    ax.set_ylabel("$y$ (m)")
    ax.set_zlabel("Time $t$ (s)")
    ax.view_init(elev=150, azimuth=0, roll=90)
    return fig
```

Plot the trajectory of the bead on the hoop through space and time.

```
fig = plot_trajectory_space_time(t, X_t, r_s)
plt.draw()
```

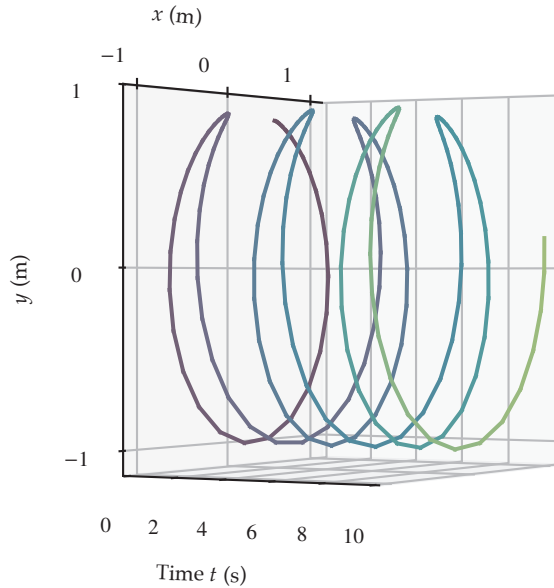


Figure 5.19. Trajectory of the bead on the hoop through space and time without damping.

Now add a viscous damping force in the anti-tangential direction.

```
b = sp.symbols("b", positive=True) # Damping coefficient
v = sp.symbols("v", real=True) # Velocity
F_d_t = -b * v # Damping force in the tangential direction
params[b] = 0.1 # Damping coefficient
F_d_t_fun = sp.lambdify((v), F_d_t.subs(params), "numpy") # Numerical
```

Apply Newton's second law to the mass to form a dynamic system state space model.

```
def system_damped(t, X, params):
    """Bead on a hoop dynamic system, with damping."""
    s, v = X # Unpack the state vector
    ds_dt = v # Velocity in the tangential direction
    dv_dt = 1/m.subs(params) * (F_g_t_fun(s) + F_d_t_fun(v))
    # Acceleration in the tangential direction
    return [ds_dt, dv_dt]
```

Numerically solve the system of ODEs for the same initial state.

```
X_t_damped = solve_ivp(
    system_damped, (t[0], t[-1]),
    X0, t_eval=t, args=(params,))
).y # Solve the system
```

Plot the trajectory of the bead on the hoop through time with damping.

```
fig = plot_trajectory(t, X_t_damped)
plt.draw()
```

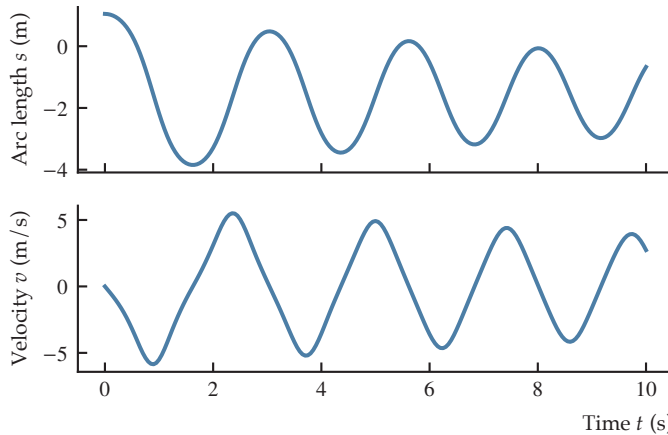


Figure 5.20. Arc length  $s$  and velocity  $v$  of the bead on the hoop through time with damping.

Plot the trajectory of the bead on the hoop through space and time with damping.

```
fig = plot_trajectory_space_time(t, X_t_damped, r_s)
plt.draw()
```

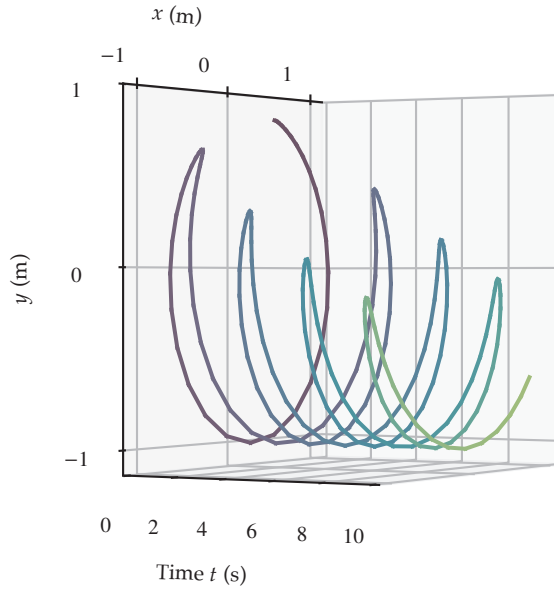


Figure 5.21. Trajectory of the bead on the hoop through space and time with damping.

Now consider the work done by the gravitational and damping forces on the bead as  $t \rightarrow \infty$ . From the state space model, in steady state,  $a = v = 0$ . This corresponds to the bead at rest at the bottom of the hoop, with  $s = -R\pi/2$ .

As we have already established, the gravitational force is conservative, so the work done by it is independent of the path taken. The work done by the damping force is given by

$$W_d = \int_{s_0}^{s_1} \mathbf{F}_d \cdot \mathbf{u}_t ds.$$

Substituting the expression for the damping force and the unit tangent vector, we have

$$W_d = \int_{s_0}^{s_1} -bv ds = -b \int_{s_0}^{s_1} v ds.$$

Or, changing the variable of integration to time via the substitution  $ds = v dt$ ,

$$W_d = -b \int_{t_0}^{t_1} v^2 dt.$$

We can evaluate this integral numerically by simulating the system for a relatively long time to get  $v(t)$ .

```
t_long = np.linspace(0, 100, 3001) # Long time array
X_t_long = solve_ivp(
    system_damped, (t_long[0], t_long[-1]),
    X0, t_eval=t_long, args=(params,))
).y # Solve the system
```

Compute the work done by the damping force.

```
W_d = -params[b] * \
    cumulative_trapezoid(X_t_long[1]**2, t_long, initial=0)
print(f"The work done by the damping force is approximately "
      f"{W_d[-1]:.2f} J.")
```

| The work done by the damping force is approximately -17.91 J.

Plot the work done by the damping force through time.

```
fig, ax = plt.subplots()
ax.plot(t_long, W_d)
ax.set_xlabel("Time $t$ (s)")
ax.set_ylabel("Damping Force Work $W_d$ (J)")
plt.show()
```

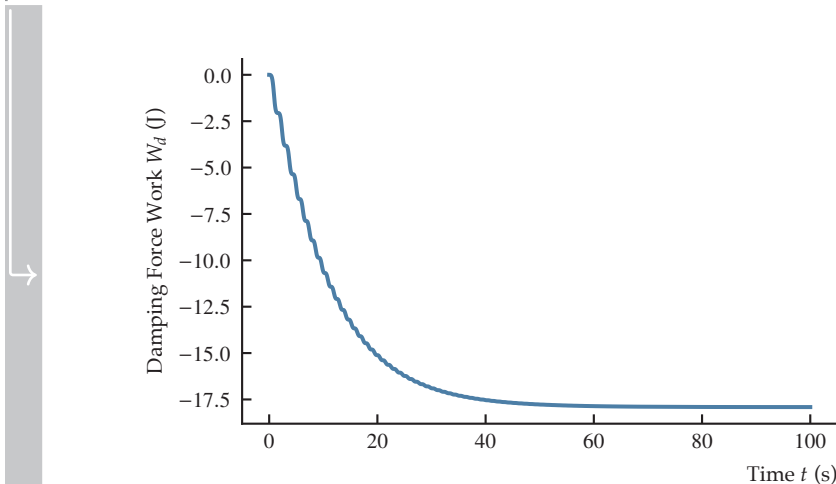


Figure 5.22. Work done by the damping force on the bead through time.

Now compute the work done by the gravitational force. Begin by computing the height of the bead at the initial and final states as follows:

```
h0 = R * sp.sin(X0[0]/R) # Height, initial state
h1 = R * sp.sin(-sp.pi/2) # Height, final state
```

If the damping force was not present and the bead started from rest ( $T_0 = 0$ ), in the downward position the kinetic energy would be equal to the difference in potential energy from the initial position; that is,  $T_1 = U_0 - U_1$ . The difference in kinetic energy from the initial position would be equal to the work done by the gravitational force,

$$W_g = T_1 - T_0 = U_0 - U_1 = mgh_0 - mgh_1 = mg(h_0 - h_1).$$

Regardless of the presence of the damping force or the path taken, by the work-energy theorem the work done by the gravitational force is equal to this change in kinetic energy of the bead. Let's compute this work done by the gravitational force.

```
| W_g = (m * g * (h0 - h1)).evalf(subs=params)
| 18.3057092111253
```

We expect that over time the work done by the damping force will be equal in magnitude to the work done by the gravitational force in moving the bead from its initial position to the bottom of the hoop.

## 5.4 Stokes and Divergence Theorems



Two theorems allow us to exchange certain integrals in  $\mathbb{R}^3$  for others that are easier to evaluate.

### 5.4.1 The Divergence Theorem

The **divergence theorem** asserts the equality of the surface integral of a vector field  $F$  and the **triple integral** of  $\operatorname{div} F$  over the volume  $V$  enclosed by the surface  $S$  in  $\mathbb{R}^3$ . That is,

$$\iint_S \mathbf{F} \cdot \mathbf{n} \, dS = \iiint_V \operatorname{div} \mathbf{F} \, dV.$$

Caveats are that  $V$  is a closed region bounded by the **orientable**<sup>4</sup> surface  $S$  and that  $F$  is continuous and continuously differentiable over a region containing  $V$ . This theorem makes some intuitive sense: we can think of the divergence inside the volume “accumulating” via the triple integration and equating the corresponding surface integral. For more on the divergence theorem, see (Kreyszig 2011; § 10.7) and (Schey 2005; pp. 45-52).

A lovely application of the divergence theorem is that, for any quantity of conserved stuff (mass, charge, spin, etc.) distributed in a spatial  $\mathbb{R}^3$  with time-dependent density  $\rho: \mathbb{R}^4 \rightarrow \mathbb{R}$  and velocity field  $\mathbf{v}: \mathbb{R}^4 \rightarrow \mathbb{R}^3$ , the divergence theorem can be

4. A surface is orientable if a consistent normal direction can be defined. Most surfaces are orientable, but some, notably the Möbius strip, cannot be. See (Kreyszig 2011; § 10.6) for more.



applied to find that

$$\partial_t \rho = -\operatorname{div}(\rho \mathbf{v}),$$

which is a more general form of a **continuity equation**, one of the governing equations of many physical phenomena. For a derivation of this equation, see (pp. 49-52).

#### 5.4.2 The Kelvin-Stokes' Theorem

The **Kelvin-Stokes' theorem** asserts the equality of the circulation of a vector field  $\mathbf{F}$  over a closed curve  $C$  and the surface integral of  $\operatorname{curl} \mathbf{F}$  over a surface  $S$  that has boundary  $C$ . That is, for  $\mathbf{r}(t)$  a parameterization of  $C$  and surface normal  $\mathbf{n}$ ,

$$\oint_C \mathbf{F}(\mathbf{r}(t)) \cdot \mathbf{r}'(t) dt = \iint_S \mathbf{n} \cdot \operatorname{curl} \mathbf{F} dS.$$

Caveats are that  $S$  is **piecewise smooth**,<sup>5</sup> its boundary  $C$  is a piecewise smooth simple closed curve, and  $\mathbf{F}$  is continuous and continuously differentiable over a region containing  $S$ . This theorem is also somewhat intuitive: we can think of the divergence over the surface “accumulating” via the surface integration and equating the corresponding circulation. For more on the Kelvin-Stokes' theorem, see (Kreyszig 2011; § 10.9) and (Schey 2005; pp. 93-102).

#### 5.4.3 Related Theorems

**Greene's theorem** is a two-dimensional special case of the Kelvin-Stokes' theorem. It is described by (Kreyszig 2011; § 10.9).

It turns out that all of the above theorems (and the fundamental theorem of calculus, which relates the derivative and integral) are special cases of the **generalized Stokes' theorem** defined by differential geometry. We would need a deeper understanding of differential geometry to understand this theorem. For more, see (Lee 2012; Ch. 16).

5. A surface is *smooth* if its normal is continuous everywhere. It is *piecewise smooth* if it is composed of a finite number of smooth surfaces.

## 5.5 Problems



**Problem 5.1** Consider a vector field  $F: \mathbb{R}^3 \rightarrow \mathbb{R}^3$  defined in Cartesian coordinates  $(x, y, z)$  as

$$F = [x^2 - y^2, y^2 - z^2, z^2 - x^2]. \quad (5.22)$$

- Compute the divergence of  $F$ .
- Compute the curl of  $F$ .
- Prove that, in a simply connected region of  $\mathbb{R}^3$ , line integrals of  $F$  are path-dependent.
- Prove that  $F$  is *not* the gradient of a potential (scalar) function (i.e., that it does not have gradience, as we've called it).

**Problem 5.2** The altitude of  $(x, y)$  points on a nearby mountain are modeled on the domain  $-2 \leq x \leq 2, -2 \leq y \leq 2$  as,

$$f(x, y) = 2 - \frac{x^2}{4} + \cos\left(\frac{\pi}{2}y\right).$$

Using this model of the mountain:

- Find the 3 dimensional path you would travel on if you were to start from the trailhead at  $(x, y) = (-1, -1.5)$  and head in a straight line to the top of the mountain at  $(0, 0)$ .
- Given the definition of work  $W = \int_C F(r) \cdot dr$ , write the equation for  $F(r)$  from the acceleration of gravity and assuming a mass of 50 kg.
- Solve for the work to climb the mountain on your path from part **a**.
- Once you get to the trailhead your friend wants to take a different route that they think will take less work. Prove that it takes the same amount of work, no matter what route you take to the top of the mountain.
- On your way up the mountain you notice you have altitude sickness at location  $(-1, -0.75)$  and need to get to a lower altitude as quickly as possible. What direction should you go to descend the fastest? Write your answer as a vector pointing in the direction you should go.

**Problem 5.3** Consider a vector field  $F: \mathbb{R}^3 \rightarrow \mathbb{R}^3$  defined in Cartesian coordinates  $(x, y, z)$  as

$$F = [-3x^2 \quad -3y^2 \quad 0]^T. \quad (5.23)$$

- Compute the divergence of  $F$ .
- Compute the curl of  $F$ .

- c. Prove that, in a simply connected region of  $\mathbf{R}^3$ , line integrals of  $\mathbf{F}$  are path-independent.
- d. Prove that  $\mathbf{F}$  is the gradient of a potential (scalar) function (i.e., that it has a gradient, as we've called it).
- e. Identify a potential function  $\phi$  for which  $\mathbf{F}$  is the gradient. Is this the only such function?



# 6 Fourier and Orthogonality



In this chapter we will explore Fourier series and transforms.

## 6.1 Fourier Series



Fourier series are mathematical series that can represent a periodic signal as a sum of sinusoids at different amplitudes and frequencies.

They are useful for solving for the response of a system to periodic inputs. However, they are probably most important *conceptually*: they are our gateway to thinking of signals in the **frequency domain**—that is, as functions of *frequency* (not time). To represent a function as a Fourier series is to *analyze* it as a sum of sinusoids at different frequencies<sup>1</sup>  $\omega_n$  and amplitudes  $a_n$ . Its **frequency spectrum** is the functional representation of amplitudes  $a_n$  versus frequency  $\omega_n$ .

Let's begin with the definition.

### Definition 6.1

The *Fourier analysis* of a periodic function  $y(t)$  is, for  $n \in \mathbb{N}_0$ , period  $T$ , and angular frequency  $\omega_n = 2\pi n/T$ ,

$$a_0 = \frac{2}{T} \int_T y(t) dt$$

$$a_n = \frac{2}{T} \int_T y(t) \cos(\omega_n t) dt$$

$$b_n = \frac{2}{T} \int_T y(t) \sin(\omega_n t) dt.$$

1. It's important to note that the symbol  $\omega_n$ , in this context, is not the natural frequency, but a frequency indexed by integer  $n$ .

The *Fourier synthesis* of a periodic function  $y(t)$  with analysis components  $a_n$  and  $b_n$  corresponding to  $\omega_n$  is

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(\omega_n t) + b_n \sin(\omega_n t).$$

Let's consider the complex form of the Fourier series, which is equivalent to definition 6.1. It may be helpful to review Euler's formula(s)—see appendix C.4.

### Definition 6.2

The *Fourier analysis* of a periodic function  $y(t)$  is, for  $n \in \mathbb{N}_0$ , period  $T$ , and angular frequency  $\omega_n = 2\pi n/T$ ,

$$c_n = \frac{1}{T} \int_T y(t) e^{-j\omega_n t} dt \quad \text{for } n \in \mathbb{Z}$$

The *Fourier synthesis* of a periodic function  $y(t)$  with analysis components  $c_n$  corresponding to  $\omega_n$  is

$$y(t) = \sum_{n=-\infty}^{\infty} c_n e^{j\omega_n t}.$$

We call the integer  $n$  a **harmonic** and the frequency associated with it,

$$\omega_n = 2\pi n/T,$$

the **harmonic frequency**. There is a special name for the first harmonic ( $n = 1$ ): the **fundamental frequency**. It is called this because all other frequency components are integer multiples of it.

It is also possible to convert between the two representations above.

### Definition 6.3

The complex Fourier analysis of a periodic function  $y(t)$  is, for  $n \in \mathbb{Z}$  and  $a_n$  and  $b_n$  as defined above,

$$c_n = \begin{cases} \frac{1}{2} (a_n - jb_n) & n \geq 0 \\ \frac{1}{2} (a_{|n|} + jb_{|n|}) & n < 0. \end{cases}$$

The sinusoidal Fourier analysis of a periodic function  $y(t)$  is, for  $n \in \mathbb{N}_0$  and  $c_n$  as defined above,

$$a_n = c_n + c_{-n} \text{ and}$$

$$b_n = j(c_n - c_{-n}).$$

The **harmonic amplitude**  $C_n$  is, for  $n \in \mathbb{N}_0$ ,

$$\begin{aligned} C_n &= \sqrt{a_n^2 + b_n^2} \\ &= 2\sqrt{c_n c_{-n}}. \end{aligned}$$

A **magnitude line spectrum** is a graph of the harmonic amplitudes as a function of the harmonic frequencies. The **harmonic phase** is

$$\begin{aligned} \theta_n &= -\arctan_2(b_n, a_n) && \text{(see appendix C.2.11)} \\ &= \arctan_2(\Im(c_n), \Re(c_n)). && (6.1) \end{aligned}$$

The illustration of figure 6.1 shows how sinusoidal components sum to represent a square wave. A line spectrum is also shown.

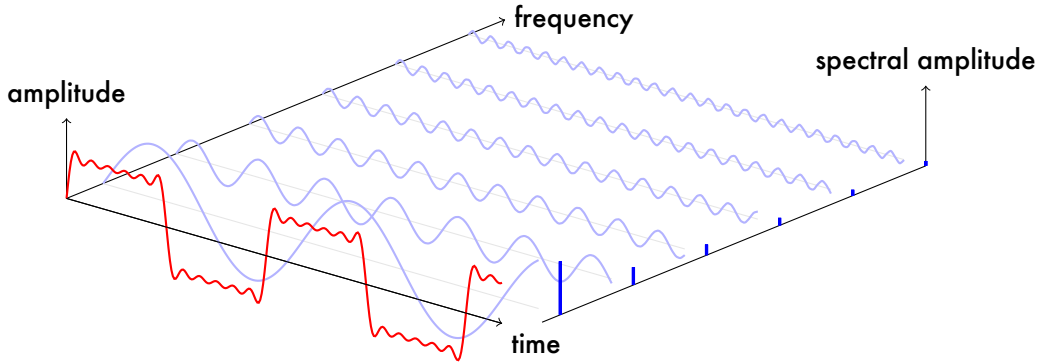


Figure 6.1. A partial sum of Fourier components of a square wave shown through time and frequency. The spectral amplitude shows the amplitude of the corresponding Fourier component.

Let us compute the associated spectral components in the following example.

### Example 6.1

Compute the first five harmonic amplitudes that represent the line spectrum for a square wave in the figure above.

Assume a square wave with amplitude 1. Compute  $a_n$ :

$$\begin{aligned} a_n &= \frac{2}{T} \int_{-T/2}^{T/2} y(t) \cos(2\pi nt/T) dt \\ &= -\frac{2}{T} \int_{-T/2}^0 \cos(2\pi nt/T) dt + \frac{2}{T} \int_0^{T/2} \cos(2\pi nt/T) dt \\ &= 0 \text{ because cosine is even.} \end{aligned}$$

Compute  $b_n$ :

$$\begin{aligned} b_n &= \frac{2}{T} \int_{-T/2}^{T/2} y(t) \sin(2\pi nt/T) dt \\ &= -\frac{2}{T} \int_{-T/2}^0 \sin(2\pi nt/T) dt + \frac{2}{T} \int_0^{T/2} \sin(2\pi nt/T) dt \\ &= \frac{2}{n\pi} (1 - \cos(n\pi)) \\ &= \begin{cases} 0 & n \text{ even} \\ \frac{4}{n\pi} & n \text{ odd} \end{cases} . \end{aligned}$$

Therefore,

$$C_n = \sqrt{a_n^2 + b_n^2}$$

$$C_0 = 0 \text{ (even)}$$

$$C_1 = \frac{4}{\pi}$$

$$C_2 = 0 \text{ (even)}$$

$$C_3 = \frac{4}{3\pi}$$

$$C_4 = 0 \text{ (even)}$$

$$C_5 = \frac{4}{5\pi} .$$



## 6.2 Partial Sums of Fourier Series



A **partial sum** of a Fourier series is the truncated synthesis

$$y(t) \approx \frac{a_0}{2} + \sum_{n=1}^N a_n \cos(\omega_n t) + b_n \sin(\omega_n t),$$

where  $a_n$  and  $b_n$  are the trigonometric series coefficients of definition 6.1 and  $N \in \mathbb{N}$  is a finite positive integer. The complex form is

$$y(t) \approx \sum_{n=-N}^N c_n e^{j\omega_n t},$$

where  $c_n$  are the complex series coefficients of definition 6.2.

Partial sums are often necessary approximations for concrete calculations.

### Example 6.2

Consider the function  $y(t)$  one period of which shown in figure 6.2. Compute a partial sum approximation and plot it.

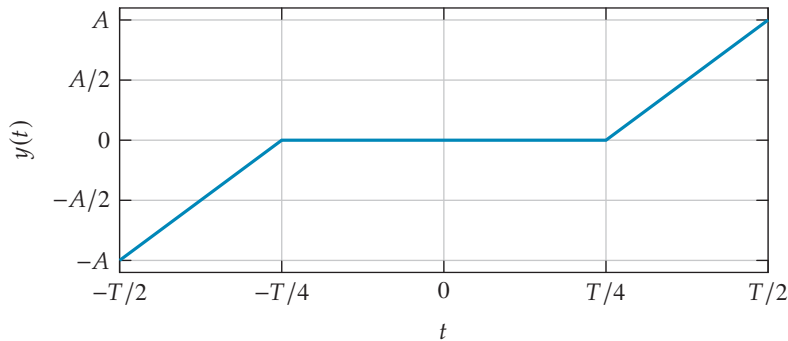


Figure 6.2. A period of the function  $y(t)$ .

We proceed in Python. First, load packages.

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
```

Over the period  $[-T/2, T/2]$ , the function  $y(t)$  is defined as:

$$y(t) = \begin{cases} (4A/T)t + A & -T/2 \leq t < -T/4 \\ 0 & -T/4 \leq t < T/4 \\ (4A/T)t - A & T/4 \leq t < T/2. \end{cases}$$

We will use a partial sum of the complex Fourier series to approximate the function  $y(t)$ .

We begin symbolically. Define the symbolic variables.

```
A, T = sp.symbols('A, T', real=True, positive=True)
t = sp.symbols('t', real=True, nonnegative=True)
n = sp.symbols('n', integer=True)
w_n = 2 * sp.pi * n / T
```

Define the function  $y(t)$ .

```
y_list = []
y_list.append(4 * A / T * t + A) # -T/2 <= t < -T/4
y_list.append(0 * A) # -T/4 <= t < T/4
y_list.append(4 * A / T * t - A) # T/4 <= t < T/2
lims = [-T/2, -T/4, T/4, T/2]
```

Compute the complex Fourier series coefficients  $c_n$  (i.e., perform the Fourier series analysis).

```
c_n = 0 # Initialize the coefficient
for i in range(len(y_list)):
    c_n += (1 / T) * sp.integrate(
        y_list[i] * sp.exp(-sp.I * w_n * t), (t, lims[i], lims[i + 1])
    )
c_n = c_n.simplify()
```

Letting  $A = 1$  and  $T = 1$ , we can compute the harmonic amplitude and phase spectrum.

```
N = 10
params = {A: 1, T: 1}
c_n_num = np.full(2 * N + 1, np.nan, dtype=complex)
n_num = np.arange(-N, N + 1)
for n_ in n_num:
    i = n_ + N
    c_n_num[i] = c_n.subs({n: n_}).evalf(subs=params)
```

Let's plot the amplitude and phase spectrum.

```
fig, ax = plt.subplots(2, 1, sharex=True)
ax[0].stem(n_num, np.abs(c_n_num))
ax[0].set_title('Amplitude spectrum')
ax[1].stem(n_num, np.angle(c_n_num))
ax[1].set_title('Phase spectrum')
ax[1].set_xlabel('Harmonic $n$')
plt.draw()
```

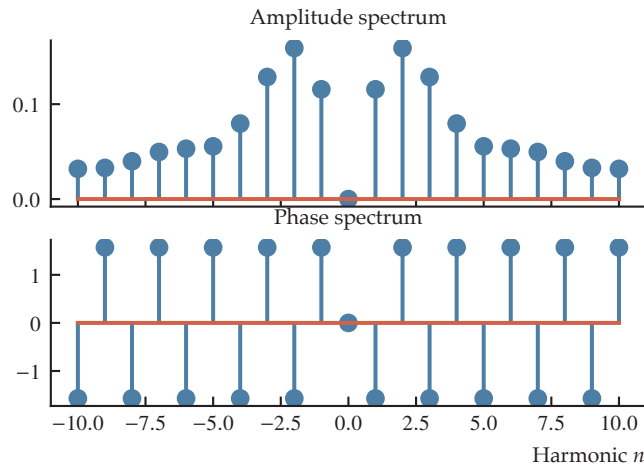


Figure 6.3. Amplitude and phase spectrum of the complex Fourier series.

Now, we will compute the partial sum of the complex Fourier series up to the  $N$ -th harmonic.

```
y_N = sp.Sum(c_n * sp.exp(sp.I * w_n * t), (n, -N, N)).doit()
```

Let's plot the function  $y(t)$  and the partial sum of the complex Fourier series.

```

t_num = np.linspace(-1, 1, 1001)
y_N_num = sp.lambdify(t, y_N.subs(params), 'numpy')(t_num)
fig, ax = plt.subplots()
ax.plot(t_num, y_N_num, label='Partial sum')
for i in range(len(y_list)):
    y_fun = sp.lambdify(t, y_list[i].subs(params), 'numpy')
    lims_ = [float(lim.evalf(subs=params)) for lim in lims[i:i + 2]]
    t_ = np.linspace(lims_[0], lims_[1], 11)
    y_ = y_fun(t_)
    if type(y_) != np.ndarray: # Happens when y(t) is a constant
        y_ = y_ * np.ones(t_.shape)
    if i == 0:
        ax.plot(t_, y_, 'r--', label='$y(t)$')
    else:
        ax.plot(t_, y_, 'r--')
ax.set_xlabel('$t$')
ax.set_ylabel('$y(t)$')
ax.legend()
plt.show()

```

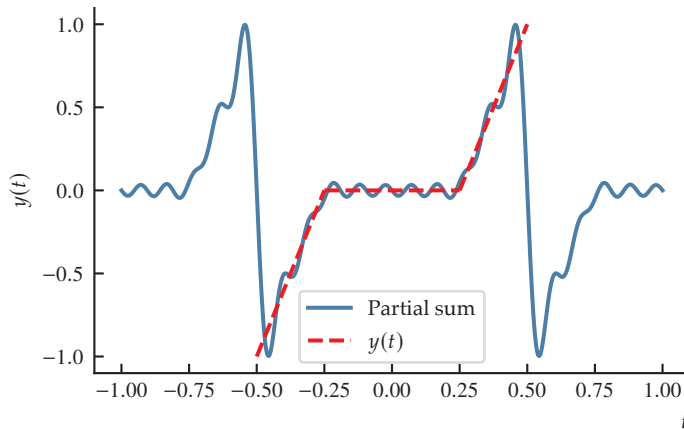


Figure 6.4. Function  $y(t)$  and partial sum of the complex Fourier series.

### 6.3 Fourier Transform



We begin with the usual loading of modules.

```
import numpy as np # for numerics
import sympy as sp # for symbolics
import matplotlib.pyplot as plt # for plots!
```

Let's consider a periodic function  $f$  with period  $T$  ( $T$ ). Each period, the function has a triangular pulse of width  $\delta$  (`pulse_width`) and height  $\delta/2$ .

```
period = 15 # period
pulse_width = 2 # pulse width
```

First, we plot the function  $f$  in the time domain. Let's begin by defining  $f$ .

```
def pulse_train(t,T,pulse_width):
    f = lambda x:pulse_width/2-abs(x) # pulse
    tm = np.mod(t,T)
    if tm <= pulse_width/2:
        return f(tm)
    elif tm >= T-pulse_width/2:
        return f(-(tm-T))
    else:
        return 0
```

Now, we develop a numerical array in time to plot  $f$ .

```
N = 151 # number of points to plot
tpp = np.linspace(-period/2,5*period/2,N) # time values
fpp = np.array(np.zeros(tpp.shape))
for i,t_now in enumerate(tpp):
    fpp[i] = pulse_train(t_now,period,pulse_width)
```

Now we plot.

```
fig, ax = plt.subplots()
ax.plot(tpp,fpp,'b-',linewidth=2) # plot
plt.xlabel('time (s)')
plt.xlim([-period/2,3*period/2])
plt.xticks(
    [0,period],
    [0,'$T='+str(period)+'$ s']
)
plt.yticks([0,pulse_width/2],['0','$\delta/2$'])
plt.draw()
```

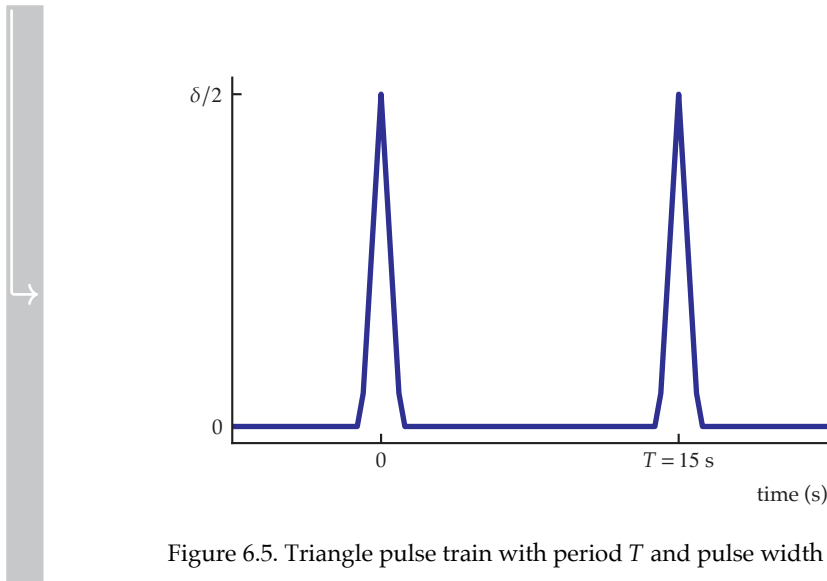


Figure 6.5. Triangle pulse train with period  $T$  and pulse width  $\delta$ .

Consider the following argument. Just as a Fourier series is a frequency domain representation of a periodic signal, a Fourier transform is a frequency domain representation of an *aperiodic* signal (we will rigorously define it in a moment). The Fourier series components will have an analog, then, in the Fourier transform. Recall that they can be computed by integrating over a period of the signal. If we increase that period infinitely, the function is effectively aperiodic. The result (within a scaling factor) will be the Fourier transform analog of the Fourier series components.

Let us approach this understanding by actually computing the Fourier series components for increasing period  $T$  using definition 6.1. We'll use sympy to compute the Fourier series cosine and sine components  $a_n$  and  $b_n$  for component  $n$  ( $n$ ) and period  $T$  ( $T$ ).

```
x, a_0, a_n, b_n = sp.symbols('x, a_0, a_n, b_n', real=True)
delta, T = sp.symbols('delta, T', positive=True)
n = sp.symbols('n', nonnegative=True)
an = sp.integrate(
    2/T*(delta/2-sp.Abs(x))*sp.cos(2*sp.pi*n/T*x),
    (x,-delta/2, delta/2) # otherwise zero
).simplify()
bn = 2/T*sp.integrate(
    (delta/2-sp.Abs(x))*sp.sin(2*sp.pi*n/T*x),
    (x, -delta/2, delta/2) # otherwise zero
).simplify()
print(sp.Eq(a_n,an), sp.Eq(b_n,bn))
```

```
Eq(a_n, Piecewise((T*(1 - cos(pi*delta*n/T))/(pi**2*n**2), n > 0),
↪ (delta**2/(2*T), True))) Eq(b_n, 0)
```

Furthermore, let us compute the *harmonic amplitude* (f\_harmonic\_amplitude):

$$C_n = \sqrt{a_n^2 + b_n^2} \quad (6.2)$$

which we have also scaled by a factor  $T/\delta$  in order to plot it with a convenient scale.

```
C_n = sp.symbols('C_n', positive=True)
cn = sp.sqrt(an**2+bn**2)
print(sp.Eq(C_n, cn))

Eq(C_n, Piecewise((T*Abs(cos(pi*delta*n/T) - 1)/(pi**2*n**2), n > 0),
↪ (delta**2/(2*T), True)))
```

Now we lambdify the symbolic expression for a numpy function.

```
cn_f = sp.lambdify((n, T, delta), cn)
```

Now we can plot. Write a function to plot pulses in the time domain with the corresponding frequency spectrum.

```
def plot_pulses_and_spectrum(T, pulse_width, omega_max):
    n_max = round(omega_max*T/(2*np.pi)) # max harmonic
    n_a = np.linspace(0,n_max,n_max+1)
    omega = 2*np.pi*n_a/T
    fig, ax = plt.subplots(1, 2)
    plt.sca(ax[0])
    for i in range(0, 3):
        tpp = np.linspace(-T/2, 5*T/2,N)
        fpp = np.array(np.zeros(tpp.shape))
        for i,t_now in enumerate(tpp):
            fpp[i] = pulse_train(t_now, T, pulse_width)
        plt.plot(tpp, fpp, 'b-', linewidth=2)
    plt.xlim([-T/2, 3*T/2])
    plt.xticks([0, T], [0, '$T='+str(T)+'$ s'])
    plt.yticks([0, pulse_width/2], ['0', '$\delta/2$'])
    plt.xlabel('time (s)')
    plt.sca(ax[1])
    plt.stem(
        omega, cn_f(n_a, T, pulse_width)*T/pulse_width, 'bo-'
    )
    plt.xlim([0, omega_max])
    plt.ylim([0, 1.1])
    plt.xlabel('Frequency $\omega$ (rad/s)')
    plt.ylabel('$C_n T/\delta$')
    return fig
```

Now we plot the pulses and their spectra for  $T \in [5, 15, 25]$  rad/s and  $\delta = 2$ .

```
omega_max = 12 # Maximum frequency to plot
fig = plot_pulses_and_spectrum(5, pulse_width, omega_max)
plt.draw()
```

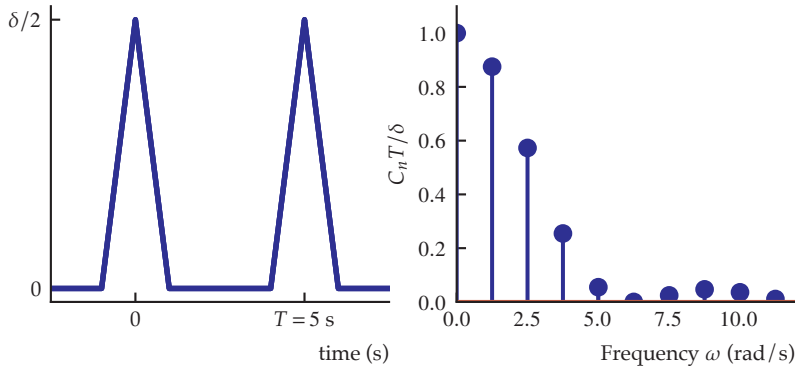


Figure 6.6. Triangle pulse train with period  $T$  and pulse width  $\delta$  and its Fourier series components for  $T = 5$  s.

```
fig = plot_pulses_and_spectrum(15, pulse_width, omega_max)
plt.draw()
```

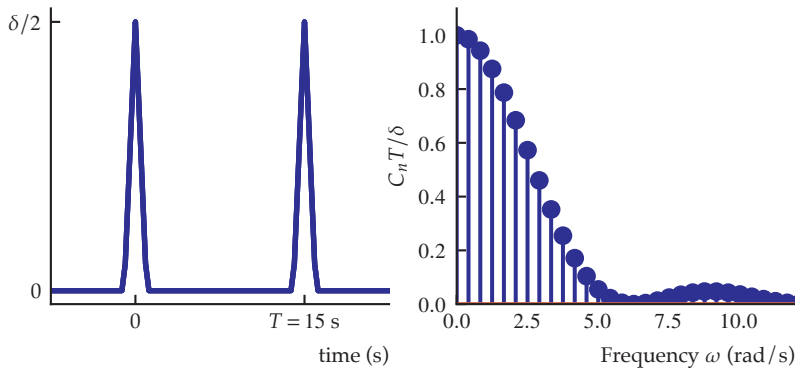


Figure 6.7. Triangle pulse train with period  $T$  and pulse width  $\delta$  and its Fourier series components for  $T = 15$  s.

```
fig = plot_pulses_and_spectrum(25, pulse_width, omega_max)
plt.draw()
```



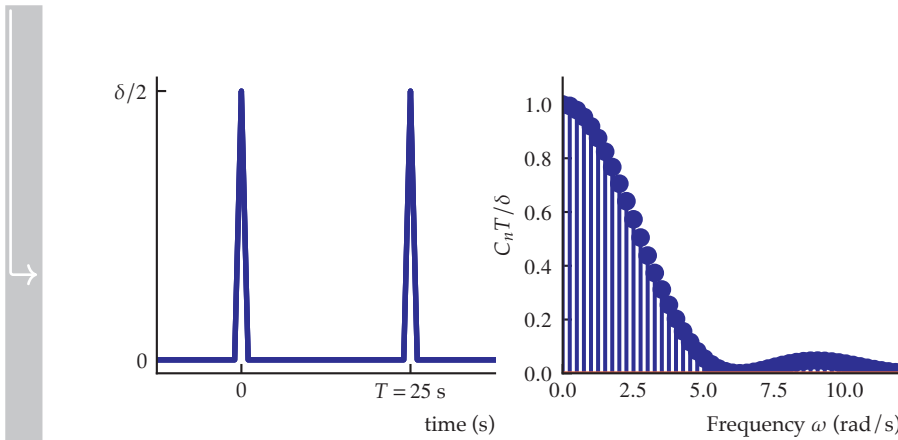


Figure 6.8. Triangle pulse train with period  $T$  and pulse width  $\delta$  and its Fourier series components for  $T = 25$  s.

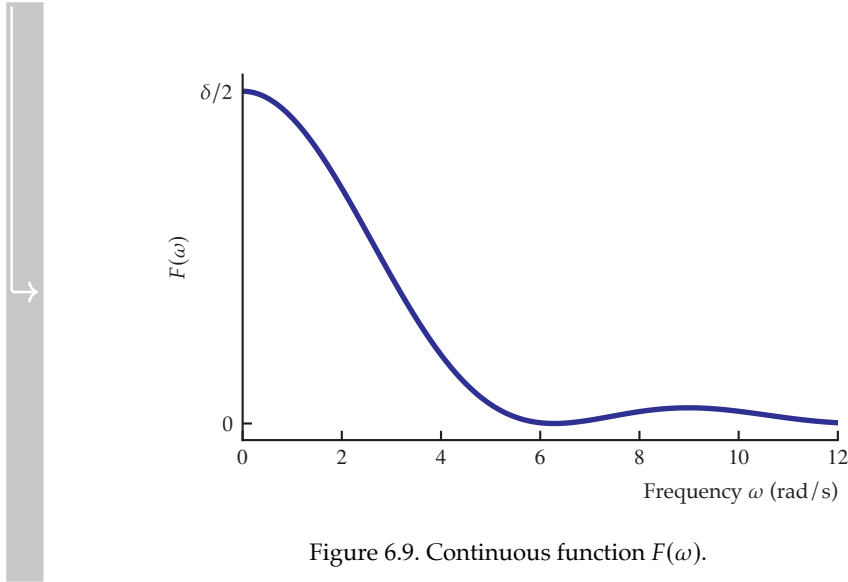
The line spectra are shown in the right-hand columns of the plots above. Note that with our chosen scaling, as  $T$  increases, the line spectra reveal a distinct waveform.

Let  $F$  be the continuous function of angular frequency  $\omega$

$$F(\omega) = \frac{\delta}{2} \cdot \frac{\sin^2(\omega\delta/4)}{(\omega\delta/4)^2}. \quad (6.3)$$

First, we plot it.

```
def F(w):
    return pulse_width/2*np.sin(w*pulse_width/4)**2 / \
           (w*pulse_width/4)**2
N = 201 # number of points to plot
wpp = np.linspace(0.0001, omega_max,N)
Fpp = []
for i in range(0,N):
    Fpp.append(F(wpp[i])) # build array of function values
fig, ax = plt.subplots()
plt.plot(wpp, Fpp, 'b-', linewidth=2) # plot
plt.xlim([0, omega_max])
plt.yticks([0, pulse_width/2], ['0', '$\delta/2$'])
plt.xlabel('Frequency $\omega$ (rad/s)')
plt.ylabel('$F(\omega)$')
plt.show()
```

Figure 6.9. Continuous function  $F(\omega)$ .

The plot of  $F$  is clearly emerging from the preceding line spectra as the period  $T$  increases.

Now we are ready to define the Fourier transform and its inverse. We will define the Fourier transform in two ways: as a trigonometric transform and as a complex transform. We begin with the trigonometric transform and its inverse.

#### Definition 6.4: Fourier Transform (Trigonometric)

Fourier transform (analysis):

$$A(\omega) = \int_{-\infty}^{\infty} y(t) \cos(\omega t) dt \quad (6.4)$$

$$B(\omega) = \int_{-\infty}^{\infty} y(t) \sin(\omega t) dt. \quad (6.5)$$

Inverse Fourier transform (synthesis):

$$y(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} A(\omega) \cos(\omega t) d\omega + \frac{1}{2\pi} \int_{-\infty}^{\infty} B(\omega) \sin(\omega t) d\omega. \quad (6.6)$$

The Fourier transform is a generalization of the Fourier series to aperiodic functions (i.e., functions with infinite period). The complex form of the Fourier transform is more convenient for analysis and computation, as we will see.

## Definition 6.5: Fourier Transform (Complex)

Fourier transform  $\mathcal{F}$  (analysis):

$$\mathcal{F}(y(t)) = Y(\omega) = \int_{-\infty}^{\infty} y(t)e^{-j\omega t} dt. \quad (6.7)$$

Inverse Fourier transform  $\mathcal{F}^{-1}$  (synthesis):

$$\mathcal{F}^{-1}(Y(\omega)) = y(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} Y(\omega)e^{j\omega t} d\omega. \quad (6.8)$$

So now we have defined the Fourier transform. There are many applications, including solving differential equations and *frequency domain* representations—called *spectra*—of *time domain* functions.

There is a striking similarity between the Fourier transform and the Laplace transform, with which you are already acquainted. In fact, the Fourier transform is a special case of a Laplace transform with Laplace transform variable  $s = j\omega$  instead of having some real component. Both transforms convert differential equations to algebraic equations, which can be solved and inversely transformed to find time-domain solutions. The Laplace transform is especially important to use when an input function to a differential equation is not absolutely integrable and the Fourier transform is undefined (for example, our definition will yield a transform for neither the unit step nor the unit ramp functions). However, the Laplace transform is also preferred for *initial value problems* due to its convenient way of handling them. The two transforms are equally useful for solving steady state problems. Although the Laplace transform has many advantages, for spectral considerations, the Fourier transform is the only game in town.

A table of Fourier transforms and their properties can be found in appendix B.2.

## Example 6.3

Consider the aperiodic signal  $y(t) = u_s(t)e^{-at}$  with  $u_s$  the unit step function and  $a > 0$ . The signal is plotted below. Derive the complex frequency spectrum and plot its magnitude and phase.

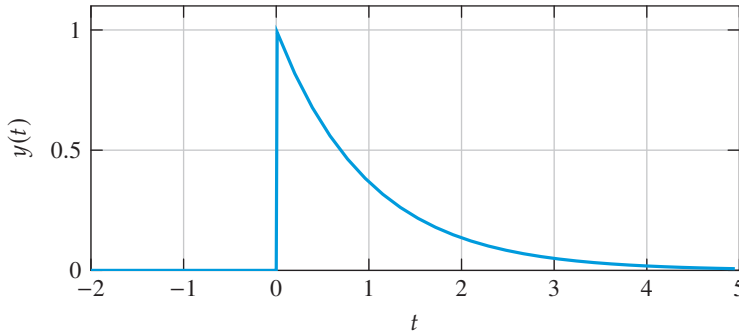


Figure 6.10. An aperiodic signal.

The signal is aperiodic, so the Fourier transform can be computed from equation (6.7):

$$\begin{aligned}
 Y(\omega) &= \int_{-\infty}^{\infty} y(t) e^{j\omega t} dt \\
 &= \int_{-\infty}^{\infty} u_s(t) e^{-at} e^{j\omega t} dt && \text{(def. of } y) \\
 &= \int_0^{\infty} e^{-at} e^{j\omega t} dt && (u_s \text{ effect}) \\
 &= \int_0^{\infty} e^{(-a+j\omega)t} dt && \text{(multiply)} \\
 &= \frac{1}{-a+j\omega} e^{(-a+j\omega)t} \Big|_0^{\infty} dt && \text{(antiderivative)} \\
 &= \frac{1}{-a+j\omega} \left( \lim_{t \rightarrow \infty} e^{(-a+j\omega)t} - e^0 \right) && \text{(evaluate)} \\
 &= \frac{1}{-a+j\omega} \left( \lim_{t \rightarrow \infty} e^{-at} e^{j\omega t} - 1 \right) && \text{(arrange)} \\
 &= \frac{1}{-a+j\omega} ((0)(\text{complex with mag } \leq 1) - 1) && \text{(limit)} \\
 &= \frac{-1}{-a+j\omega} && \text{(consequence)} \\
 &= \frac{1}{a-j\omega} \\
 &= \frac{a+j\omega}{a+j\omega} \cdot \frac{1}{a-j\omega} && \text{(rationalize)}
 \end{aligned}$$

$$= \frac{a + j\omega}{a^2 + \omega^2}.$$

The magnitude and phase of this complex function are straightforward to compute:

$$\begin{aligned} |Y(\omega)| &= \sqrt{\Re(Y(\omega))^2 + \Im(Y(\omega))^2} \\ &= \frac{1}{a^2 + \omega^2} \sqrt{a^2 + \omega^2} \\ &= \frac{1}{\sqrt{a^2 + \omega^2}} \\ \angle Y(\omega) &= \arctan(\omega/a). \end{aligned}$$

Now we can plot these functions of  $\omega$ . Setting  $a = 1$  (arbitrarily), we obtain the plots of figure 6.11.

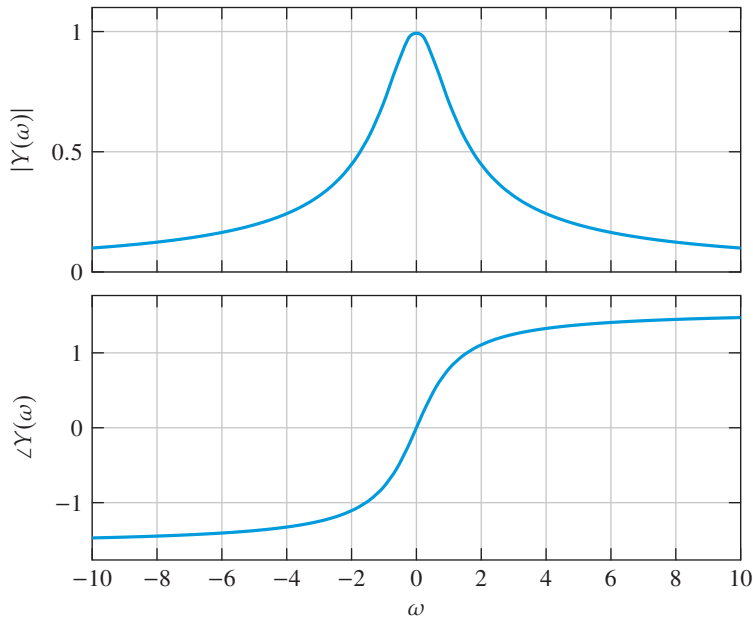


Figure 6.11. The magnitude and phase of the Fourier transform.

## 6.4 Generalized Fourier Series and Orthogonality



Let  $f : \mathbb{R} \rightarrow \mathbb{C}$ ,  $g : \mathbb{R} \rightarrow \mathbb{C}$ , and  $w : \mathbb{R} \rightarrow \mathbb{C}$  be complex functions. For square-integrable<sup>2</sup>  $f$ ,  $g$ , and  $w$ , the **inner product** of  $f$  and  $g$  with **weight function**  $w$  over the interval  $[a, b] \subseteq \mathbb{R}$  is<sup>3</sup>

$$\langle f, g \rangle_w = \int_a^b f(x) \overline{g(x)} w(x) dx$$

where  $\overline{g}$  denotes the complex conjugate of  $g$ . The inner product of functions can be considered analogous to the inner (or dot) product of vectors.

The fourier series components can be found by a special property of the sin and cos functions called **orthogonality**. In general, functions  $f$  and  $g$  from above are *orthogonal* over the interval  $[a, b]$  iff

$$\langle f, g \rangle_w = 0$$

for weight function  $w$ . Similar to how a set of orthogonal vectors can be a basis for a vector space, a set of orthogonal functions can be a **basis** for a **function space**: a vector space of functions from one set to another (with certain caveats).

In addition to some sets of sinusoids, there are several other important sets of functions that are orthogonal. For instance, sets of **legendre polynomials** (Kreyszig 2011; § 5.2) and **bessel functions** (§ 5.4) are orthogonal.

As with sinusoids, the orthogonality of some sets of functions allows us to compute their series components. Let functions  $f_0, f_1, \dots$  be orthogonal with respect to weight function  $w$  on interval  $[a, b]$  and let  $a_0, a_1, \dots$  be real constants. A **generalized fourier series** is (§ 11.6)

$$f(x) = \sum_{m=0}^{\infty} a_m f_m(x)$$

and represents a function  $f$  as a convergent series. It can be shown that the **Fourier components**  $a_m$  can be computed from

$$a_m = \frac{\langle f, f_m \rangle_w}{\langle f_m, f_m \rangle_w}. \quad (6.9)$$

In keeping with our previous terminology for fourier series, section 6.4 and equation (6.9) are called general fourier **synthesis** and **analysis**, respectively.

For the aforementioned legendre and bessel functions, the generalized fourier series are called **fourier-legendre** and **fourier-bessel series** (§ 11.6). These and the standard fourier series (section 6.1) are of particular interest for the solution of partial differential equations (chapter 8).

2. A function  $f$  is square-integrable if  $\int_{-\infty}^{\infty} |f(x)|^2 dx < \infty$ .

3. This definition of the inner product can be extended to functions on  $\mathbb{R}^2$  and  $\mathbb{R}^3$  domains using double- and triple-integration. See (Schey 2005; p. 261).

### Example 6.4

The **Bessel functions of the first kind** are series solutions to the Bessel differential equation (§ 5.4), which models boundary value problems associated with circular membranes in polar coordinates. The  $n$ th Bessel function is defined as

$$J_\nu(x) = x^\nu \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{2^{2m+\nu} m! \Gamma(\nu + m + 1)},$$

where we have used the following definitions:

- $\nu$  is a real number called the **order** of the Bessel function
- $\Gamma$  is the **gamma function** defined as  $\Gamma(\nu + 1) = \int_0^\infty e^{-t} t^\nu dt$  for  $\nu > -1$ . The gamma function is a generalization of the factorial function to real numbers.

The Bessel functions  $J_n(k_{n,m}x)$ , for nonnegative integer  $n$ , are orthogonal with respect to the weight function  $w(x) = x$  on the interval  $[0, R]$  for  $R > 0$  (kreyzig2011). Here

$$k_{n,m} = \frac{\alpha_{n,m}}{R}, \quad (6.10)$$

where  $\alpha_{n,m}$  are the zeros of the Bessel function  $J_n$ . This orthogonality allows us to compute the Fourier-Bessel series components of a function  $f(x)$  on  $[0, R]$  from equation (6.9).

In this example, we will explore the definition of the Bessel function by plotting  $J_0, J_1, J_{1/2}$ , and  $J_{-1/2}$ . Furthermore, we will compute the first several Fourier-Bessel series components of the function  $f(x) = 1 - x^3$  on the interval  $x \in [0, 1]$  and plot partial sums of the series.

We proceed in Python. First, load packages.

```
import numpy as np
import sympy as sp
import scipy
from scipy.special import jn_zeros
import matplotlib.pyplot as plt
```

Define the Bessel functions of the first kind of orders 0 and 1.

```
x = sp.symbols('x', nonnegative=True)
J_0 = sp.besselj(0, x)
J_1 = sp.besselj(1, x)
```

Plot these functions.

```

x_plt = np.linspace(0, 20, 1001)
J_0_fun = sp.lambdify(x, J_0, modules=['numpy', 'scipy'])
J_1_fun = sp.lambdify(x, J_1, modules=['numpy', 'scipy'])
fig, ax = plt.subplots()
ax.plot(x_plt, J_0_fun(x_plt), label='$J_0(x)$')
ax.plot(x_plt, J_1_fun(x_plt), label='$J_1(x)$')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.set_xlabel('$x$')
ax.set_ylabel('$J_n(x)$')
ax.legend()
plt.draw()

```

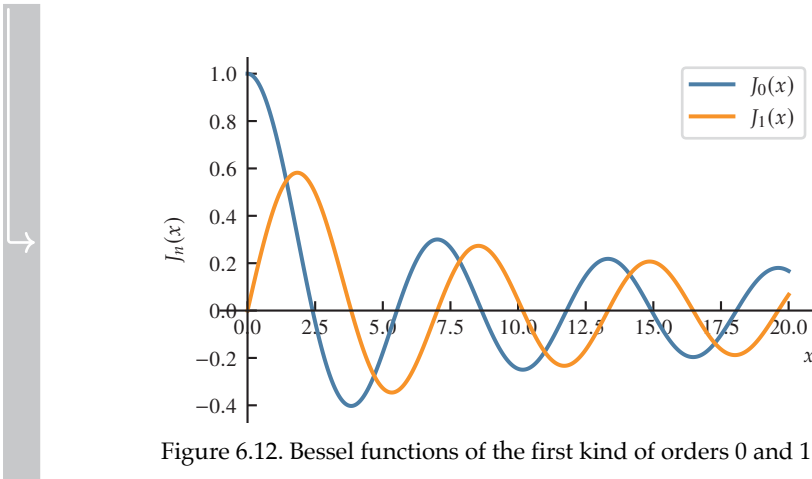


Figure 6.12. Bessel functions of the first kind of orders 0 and 1.

The plot shows the Bessel functions of the first kind of orders 0 and 1. The functions are oscillatory and decay to zero as  $x$  increases. They resemble sinusoids with decreasing amplitude.

Next, we will plot the Bessel functions of the first kind of orders  $1/2$  and  $-1/2$ .

```

J_half = sp.besselj(1/2, x)
J_half_n = sp.besselj(-1/2, x)

```

Plot these functions.



```

J_half_fun = sp.lambdify(x, J_half, modules=['numpy', 'scipy'])
J_half_n_fun = sp.lambdify(x, J_half_n, modules=['numpy', 'scipy'])
fig, ax = plt.subplots()
ax.plot(x_plt, J_half_fun(x_plt), label='$J_{1/2}(x)$')
ax.plot(x_plt, J_half_n_fun(x_plt), label='$J_{-1/2}(x)$')
ax.set_ylim(-1, 1)
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.set_xlabel('$x$')
ax.set_ylabel('$J_n(x)$')
ax.legend()
plt.draw()

```

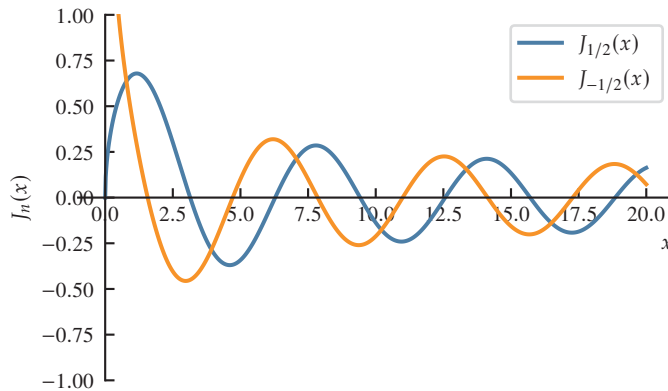


Figure 6.13. Bessel functions of the first kind of orders  $1/2$  and  $-1/2$ .

These functions are similar to the Bessel functions of orders 0 and 1. However,  $J_{-1/2}(x)$  has a singularity at  $x=0$ .

Note that, like sinusoids, all these Bessel functions produce zeros at certain values of  $x$ . The zeros are crucial to the orthogonality of the Bessel functions, as the introduction to this example explains. Therefore, we must be able to compute these zeros. Kreyszig (2011; appendix 5) provides a table for the first several zeros of the Bessel functions. Alternatively, we can use the `scipy` package's `special` module to compute the zeros, as follows for  $J_0(x)$ .

```

n_zeros = 5
alpha_0 = jn_zeros(0, n_zeros)

```

Turning to the Fourier-Bessel series, we will consider the function  $f(x) = 1 - x^3$  on the interval  $[0, 1]$ . The Fourier-Bessel series analysis of this function computes

the coefficients  $a_m$  from equation (6.9). The coefficients are given by

$$a_m = \frac{\langle f, f_m \rangle_w}{\langle f_m, f_m \rangle_w},$$

where  $f_m = J_0(k_{0,m}x)$  and  $k_{0,m}$  are given by equation (6.10) and  $w(x) = x$  is the weight function. The inner products are computed as

$$\langle f, f_m \rangle_w = \int_0^R f(x) f_m(x) w(x) dx, \quad (6.11)$$

$$\langle f_m, f_m \rangle_w = \int_0^R f_m(x)^2 w(x) dx. \quad (6.12)$$

Computing these integrals, we obtain the following:

```
R = 1
m = sp.symbols('m', integer=True, positive=True)
k_0_m = sp.symbols('k_0_m', real=True, positive=True)
f = 1 - x**3
f_m = J_0.subs(x, k_0_m * x)
w = x
inner_f_f_m = sp.integrate(f * f_m * w, (x, 0, R))
inner_f_m_f_m = sp.integrate(f_m**2 * w, (x, 0, R))
a_m = inner_f_f_m / inner_f_m_f_m
```

Convert the symbolic expression for  $a_m$  to a numerical function of  $k_{0,m}$  and compute the coefficients for the first several values of  $k_{0,m}$ .

```
a_m_fun = sp.lambdify(k_0_m, a_m, modules=['numpy', 'mpmath'])
alpha_0 = np.zeros_like(alpha_0)
for i, alpha in enumerate(alpha_0): # Vectorization fails; workaround
    a_m[i] = a_m_fun(alpha/R)
print(a_m_)
```

```
| [ 1.27215302 -0.33478735  0.09593875 -0.04835341  0.02549602]
```

Next, we compute the partial sum of the Fourier-Bessel series up to  $m = 5$ . The partial sum is given by

$$f_N(x) = \sum_{m=1}^N a_m J_0(k_{0,m}x).$$

We will evaluate this sum numerically.

```
N = n_zeros
f_N = 0
for i in range(N):
    f_N += a_m[i] * sp.besselj(0, alpha_0[i] * x)
f_N_fun = sp.lambdify(x, f_N, modules=['numpy', 'scipy'])
```

Plot the function  $f(x)$  and the partial sum  $f_N(x)$ .

```
x_plt = np.linspace(0, R, 101)
fig, ax = plt.subplots()
ax.plot(x_plt, 1 - x_plt**3, label='$f(x)$')
ax.plot(x_plt, f_N_fun(x_plt),
        label='Partial sum $f_N(x)$', linestyle='--')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.set_xlabel('$x$')
ax.set_ylabel('$f(x)$')
ax.legend()
plt.show()
```

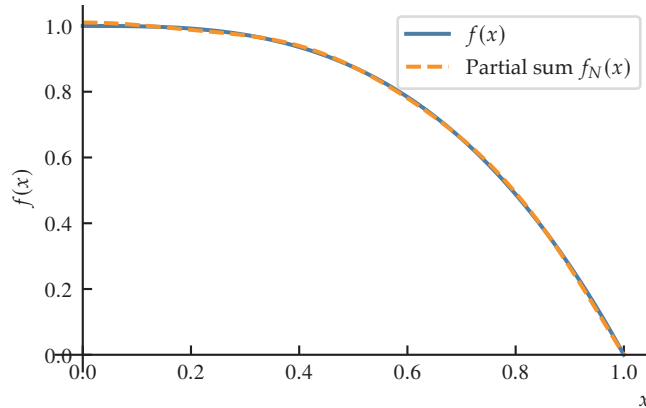


Figure 6.14. Function  $f(x)$  and the partial sum of the Fourier-Bessel series.

The plot shows the function  $f(x) = 1 - x^3$  and the partial sum of the Fourier-Bessel series up to  $m = 5$ . The partial sum approximates the function quite well, capturing its general shape.

## 6.5 Problems



**Problem 6.1** 🐉STANISLAW Explain, in your own words (supplementary drawings are ok), what the *frequency domain* is, how we derive models in it, and why it is useful.

**Problem 6.2** 🐉PUG Consider the function

$$f(t) = 8 \cos(t) + 6 \sin(2t) + \sqrt{5} \cos(4t) + 2 \sin(4t) + \cos(6t - \pi/2).$$

(a) Find the (harmonic) magnitude and (harmonic) phase of its Fourier series components. (b) Sketch its magnitude and phase spectra. *Hint: no Fourier integrals are necessary to solve this problem.*

**Problem 6.3** 🐉PONYO Consider the function with  $a > 0$

$$f(t) = e^{-a|t|}.$$

From the transform definition, derive the Fourier transform  $F(\omega)$  of  $f(t)$ . Simplify the result such that it is clear the expression is real (no imaginary component).

**Problem 6.4** 🐉SEESAW Consider the periodic function  $f: \mathbb{R} \rightarrow \mathbb{R}$  with period  $T$  defined for one period as

$$f(t) = at \quad \text{for } t \in (-T/2, T/2] \quad (6.13)$$

where  $a, T \in \mathbb{R}$ . Perform a fourier series analysis on  $f$ . Letting  $a = 5$  and  $T = 1$ , plot  $f$  along with the partial sum of the fourier series synthesis, the first 50 nonzero components, over  $t \in [-T, T]$ .

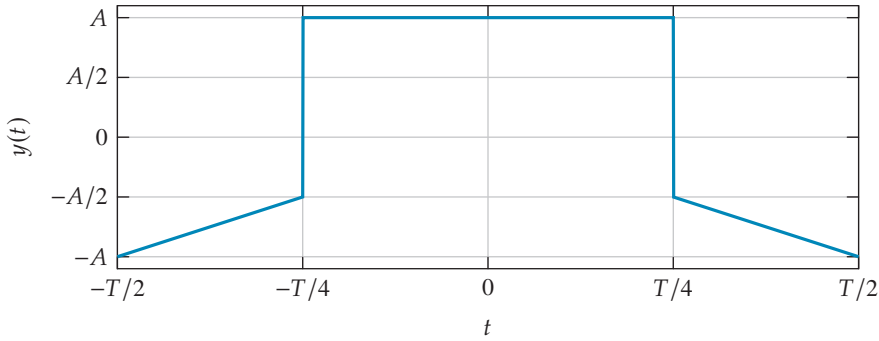




Figure 6.15. one period  $T$  of the function  $y(t)$ . Every line that appears straight is so.

**Problem 6.5**  Consider a periodic function  $y(t)$  with some period  $T \in \mathbb{R}$  and some parameter  $A \in \mathbb{R}$  for which one period is shown in figure 6.15.

- Perform a *trigonometric* Fourier series analysis of  $y(t)$  and write the Fourier series  $Y(\omega)$ .
- Plot the harmonic amplitude spectrum of  $Y(\omega)$  for  $A = T = 1$ . Consider using computing software.
- Plot the phase spectrum of  $Y(\omega)$  for  $A = T = 1$ . Consider using computing software.

**Problem 6.6**  Consider the function  $f : \mathbb{R} \rightarrow \mathbb{R}$  defined as


$$f(t) = \begin{cases} a - a|t|/T & \text{for } t \in [-T, T] \\ 0 & \text{otherwise} \end{cases} \quad (6.14)$$

where  $a, T \in \mathbb{R}$ . Perform a fourier transform analysis on  $f$ , resulting in  $F(\omega)$ . Plot  $F$  for various  $a$  and  $T$ .

**Problem 6.7**  Consider the function  $f : \mathbb{R} \rightarrow \mathbb{R}$  defined as


$$f(t) = ae^{-b|t-T|} \quad (6.15)$$

where  $a, b, T \in \mathbb{R}$ . Perform a fourier transform analysis on  $f$ , resulting in  $F(\omega)$ . Plot  $F$  for various  $a$ ,  $b$ , and  $T$ .

**Problem 6.8**  Consider the function  $f : \mathbb{R} \rightarrow \mathbb{R}$  defined as

$$f(t) = a \cos \omega_0 t + b \sin \omega_0 t \quad (6.16)$$

where  $a, b, \omega_0 \in \mathbb{R}$  constants. Perform a fourier transform analysis on  $f$ , resulting in  $F(\omega)$ .<sup>4</sup>

**Problem 6.9**  **SECRETS** This exercise encodes a “secret word” into a sampled waveform for decoding via a *discrete fourier transform* (DFT). The nominal goal of the exercise is to decode the secret word. Along the way, plotting and interpreting the DFT will be important.

First, load relevant packages.

```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, Markdown, Latex
```

We define two functions: `letter_to_number` to convert a letter into an integer index of the alphabet (a becomes 1, b becomes 2, etc.) and `string_to_number_list` to convert a string to a list of ints, as shown in the example at the end.

```
def letter_to_number(letter):
    return ord(letter) - 96

def string_to_number_list(string):
    out = [] # list
    for i in range(0, len(string)):
        out.append(letter_to_number(string[i]))
    return out # list

print(f"aces = { string_to_number_list('aces') }")

| aces = [1, 3, 5, 19]
```

Now, we encode a code string code into a signal by beginning with “white noise,” which is *broadband* (appears throughout the spectrum) and adding to it *sin* functions with amplitudes corresponding to the letter assignments of the code and harmonic corresponding to the position of the letter in the string. For instance, the string ‘bad’ would be represented by noise plus the signal

$$2 \sin 2\pi t + 1 \sin 4\pi t + 4 \sin 6\pi t. \quad (6.17)$$

Let’s set this up for secret word ‘chupcabra’.

```
N = 2000
Tm = 30
```

4. It may be alarming to see a Fourier transform of a periodic function! Strictly speaking, it does not exist; however, if we extend the transform to include the *distribution* (not actually a function) Dirac  $\delta(\omega)$ , the modified-transform does exist and is given in table B.2.

4. Python code in this section was generated from a Jupyter notebook named `random_signal_fft.ipynb` with a `python3` kernel.

```

T = float(Tm)/float(N)
fs = 1/T
x = np.linspace(0, Tm, N)
noise = 4*np.random.normal(0, 1, N)
code = 'chupcabra' # the secret word
code_number_array = np.array(string_to_number_list(code))
y = np.array(noise)
for i in range(0,len(code)):
    y = y + code_number_array[i]*np.sin(2.*np.pi*(i+1.)*x)

```

For proper decoding, later, it is important to know the fundamental frequency of the generated data.

```

print(f"fundamental frequency = {fs} Hz")
| fundamental frequency = 66.66666666666667 Hz

```

Now, we plot.

```

fig, ax = plt.subplots()
plt.plot(x,y)
plt.xlim([0,Tm/4])
plt.xlabel('time (s)')
plt.ylabel('$y_n$')
plt.show()

```

Finally, we can save our data to a numpy file `secrets.npy` to distribute our message.

```

| np.save('secrets',y)

```

Now, I have done this (for a different secret word!) and saved the data; download it here:

<https://math.ricopic.one/sg>

In order to load the `.npy` file into *Python*, we can use the following command.

```

| secret_array = np.load('secrets.npy')

```

Your job is to (a) perform a DFT, (b) plot the spectrum, and (c) decode the message! Here are a few hints.

1. Use `from scipy import fft` to do the DFT.
2. Use a hanning window to minimize the end-effects. See `numpy.hanning` for instance. The `fft` call might then look like

```

| 2*fft(np.hanning(N)*secret_array)/N

```

where `N = len(secret_array)`.



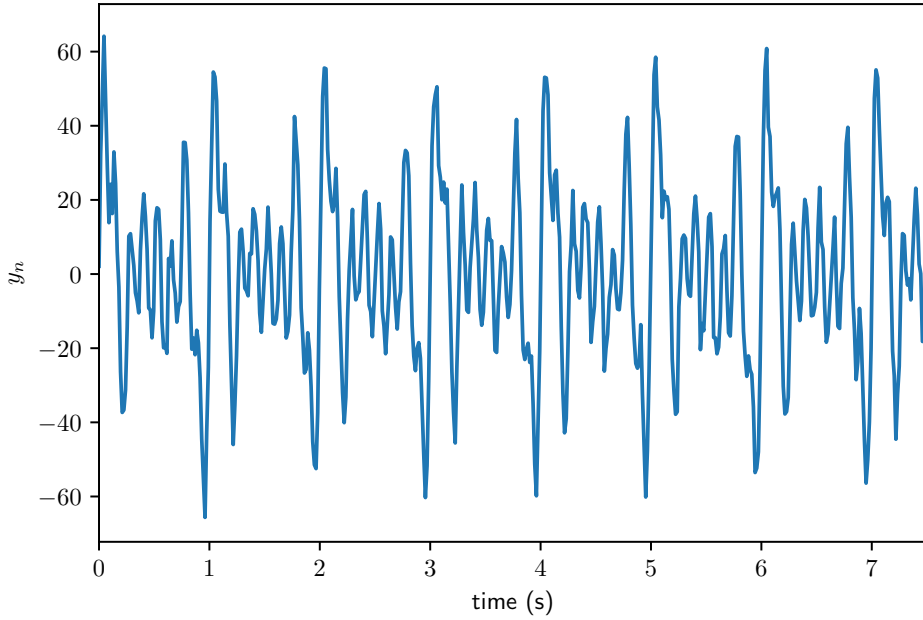


Figure 6.16. the chupacabra signal.

3. Use only the *positive* spectrum; you can lop off the negative side and double the positive side.

**Problem 6.10** 🍷🍷🍷 Derive a fourier transform property for expressions including function  $f : \mathbb{R} \rightarrow \mathbb{R}$  for

$$f(t) \cos(\omega_0 t + \psi)$$

where  $\omega_0, \psi \in \mathbb{R}$ .

**Problem 6.11** 🍷🍷🍷 Consider the function  $f : \mathbb{R} \rightarrow \mathbb{R}$  defined as

$$f(t) = a u_s(t) e^{-bt} \cos(\omega_0 t + \psi) \quad (6.18)$$

where  $a, b, \omega_0, \psi \in \mathbb{R}$  and  $u_s(t)$  is the unit step function. Perform a fourier transform analysis on  $f$ , resulting in  $F(\omega)$ . Plot  $F$  for various  $a, b, \omega_0, \psi$  and  $T$ .

**Problem 6.12** 🍷🍷🍷 Consider the function  $f : \mathbb{R} \rightarrow \mathbb{R}$  defined as

$$f(t) = g(t) \cos(\omega_0 t) \quad (6.19)$$



where  $\omega_0 \in \mathbb{R}$  and  $g : \mathbb{R} \rightarrow \mathbb{R}$  will be defined in each part below. Perform a fourier transform analysis on  $f$  for each  $g$  below for  $\omega_1 \in \mathbb{R}$  a constant and consider how things change if  $\omega_1 \rightarrow \omega_0$ .

- a.  $g(t) = \cos(\omega_1 t)$
- b.  $g(t) = \sin(\omega_1 t)$

**Problem 6.13** 🐍**SAVAGE** An instrument called a “lock-in amplifier” can measure a sinusoidal signal  $A \cos(\omega_0 t + \psi) = a \cos(\omega_0 t) + b \sin(\omega_0 t)$  at a known frequency  $\omega_0$  with exceptional accuracy even in the presence of significant noise  $N(t)$ . The workings of these devices can be described in two operations: first, the following operations on the signal with its noise,  $f_1(t) = a \cos(\omega_0 t) + b \sin(\omega_0 t) + N(t)$ ,

$$f_2(t) = f_1(t) \cos(\omega_1 t) \quad \text{and} \quad f_3(t) = f_1(t) \sin(\omega_1 t). \quad (6.20)$$

where  $\omega_0, \omega_1, a, b \in \mathbb{R}$ . Note the relation of this operation to the Fourier transform analysis of problem 6.12. The key is to know with some accuracy  $\omega_0$  such that the instrument can set  $\omega_1 \approx \omega_0$ . The second operation on the signal is an aggressive low-pass filter. The filtered  $f_2$  and  $f_3$  are called the *in-phase* and *quadrature* components of the signal and are typically given a complex representation

$$(\text{in-phase}) + j (\text{quadrature}).$$

Explain with fourier transform analyses on  $f_2$  and  $f_3$

- a. what  $F_2 = \mathcal{F}(f_2)$  looks like,
- b. what  $F_3 = \mathcal{F}(f_3)$  looks like,
- c. why we want  $\omega_1 \approx \omega_0$ ,
- d. why a low-pass filter is desirable, and
- e. what the time-domain signal will look like.

**Problem 6.14** 🐍**STRAWMAN** Consider again the lock-in amplifier explored in problem 6.13. Investigate the lock-in amplifier numerically with the following steps.

- a. Generate a noisy sinusoidal signal at some frequency  $\omega_0$ . Include enough broadband white noise that the signal is invisible in a time-domain plot.
- b. Generate  $f_2$  and  $f_3$ , as described in problem 6.13.
- c. Apply a time-domain discrete low-pass filter to each  $f_2 \mapsto \phi_2$  and  $f_3 \mapsto \phi_3$ , such as `scipy.signal.sosfiltfilt`, to complete the lock-in amplifier operation. Plot the results in time and as a complex (polar) plot.
- d. Perform a discrete fourier transform on each  $f_2 \mapsto F_2$  and  $f_3 \mapsto F_3$ . Plot the spectra.
- e. Construct a frequency domain low-pass filter  $F$  and apply it (multiply!) to each  $F_2 \mapsto F'_2$  and  $F_3 \mapsto F'_3$ . Plot the filtered spectra.
- f. Perform an inverse discrete fourier transform to each  $F'_2 \mapsto f'_2$  and  $F'_3 \mapsto f'_3$ . Plot the results in time and as a complex (polar) plot.

- g. Compare the two methods used, i.e. time-domain filtering versus frequency-domain filtering.

# 7 Ordinary Differential Equations



An **ordinary differential equation** is one with (ordinary) derivatives of functions of a single variable each—time, in many applications. These typically describe quantities in some sort of **lumped-parameter** way: mass as a “point particle,” a spring’s force as a function of time-varying displacement across it, a resistor’s current as a function of time-varying voltage across it. Given the simplicity of such models in comparison to the wildness of nature, it is quite surprising how well they work for a great many phenomena. For instance, electronics, rigid body mechanics, population dynamics, bulk fluid mechanics, and bulk heat transfer can be lumped-parameter modeled.

Three good texts on PDEs for further study are Kreyszig (2011; chapters 1–6), TODO.

## 7.1 SISO Linear Systems



In engineering, we often consider the design, mathematical modeling, or analysis of machines, circuits, biological populations, etc. We call these, in aggregate, **systems**. The vast majority of systems we consider are **dynamic**: they change over time. We can analyze such systems by writing mathematical representations of appropriate physical laws.

### 7.1.1 Dynamic Systems

For instance, a simple machine might have a link pinned and actuated by a motor at one end, as shown in figure 7.1. The angle  $\theta$  of the link might change with time, depending on the external forces acting on it, which include the motor torque. We could apply Newton’s laws to describe this motion.

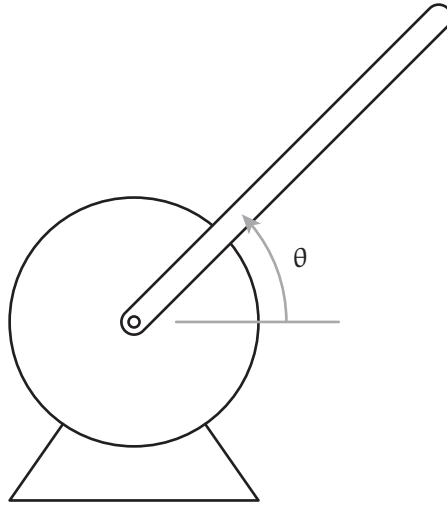


Figure 7.1. a simple machine consisting of a motor and a link.

Assuming the link's weight creates a moment about the motor shaft much smaller than the torque  $T$  applied by the motor, and letting the link have mass moment of inertia  $I$  about the motor shaft, Newton's second law in its angular form yields

$$T = I \frac{d^2 \theta}{dt^2}.$$

What type of mathematical object is this? The derivatives make it a **differential equation** with independent variable time  $t$  and dependent variables (functions of time)  $\theta$  and  $T$ . The derivatives are all ordinary derivatives and not partial derivatives, so it is an **ordinary differential equation** (ODE). Let's assume the torque  $T$  applied by the motor is known and that the angle  $\theta$  is unknown. The unknown dependent variable  $\theta$  and its derivatives enter the differential equation *linearly*, making it a **linear ordinary differential equation**.

#### Box 7.1

##### Course connections: Differential Equations, Computer Applications in Engineering

In Differential Equations, you spend a lot of time studying ODEs. This primer focuses on a specific subset of material from that course and presents one unified way to solve all such problems. This primer is, of course, no substitute for the course.

In Computer Applications in Engineering, you learned some fundamental numerical techniques for solving ODEs. These techniques apply directly to the ODEs presented in this primer, which focuses on *analytic* instead of *numerical* solutions.

**Dynamic systems** that can be effectively described by such equations are called **linear dynamic systems**. Note that many are *approximately* linear. Therefore, we spend a great deal of time analyzing linear dynamic systems. In fact, early courses in physics and engineering—covering topics going by names such as *mechanics*, *electronics*, and *dynamics*—mostly consist of learning physical laws that have precisely this form. We have seen that, in at least one case (and, in fact, in many others), Newton’s second law yields a linear system description. In electronics, one can effectively describe the voltage-current  $v$ – $i$  relationships of discrete components such as capacitors and inductors with simple differential equations; letting a capacitor’s capacitance be  $C$  and an inductor’s inductance be  $L$ :

$$\frac{dv}{dt} = \frac{1}{C}i \quad \text{and} \quad \frac{di}{dt} = \frac{1}{L}v.$$

As we know, circuits often consist of several such components and can be described by combining these sorts of simple equations to form those that are more complex.

### Box 7.2

#### **Course connections: Physics I, Physics II, Dynamics**

In Physics I and Dynamics, you were often applying Newton’s second law to derive a system of equations. These equations were, in fact, ODEs!

In Physics II, you performed circuit analyses. Whenever a capacitor or an inductor were included in the circuit, the resulting equations were ODEs!

### Box 7.3

#### **Course connections: Mechatronics, System Dynamics and Control, Heat Transfer, Vibration Theory, etc.**

In a great many of your Mechanical Engineering courses, you will encounter linear ODEs. Investing your time in this primer will pay dividends throughout. Note that more advanced solution techniques for multiple-input, multiple output (MIMO) systems, nonlinear systems, and distributed systems described by partial differential equations are beyond the scope of this primer. Where such systems arise in the ME curriculum, solution techniques will be discussed. However, throughout the curriculum, it is often assumed that you can solve linear ODEs without too much trouble.

### 7.1.2 Inputs

These system descriptions in the form linear ODEs often include “dependent” variables (meaning they’re dependent on time) that can be considered *independent of the system’s dynamics*. They are therefore prescribed externally and “input” to the system, thereby getting their name: **inputs**.

What is and what is not an input depend on the system definition. For instance, in our motor-link example, above, we made the nebulous statement that the motor torque  $T$  was “known” and the angle  $\theta$  was “unknown.” Stated a bit more precisely,  $T$  was taken to be an *input*, whereas  $\theta$  was not. This means the motor itself was *not* part of the system described by the ODE. However, we could have included it in the system. This would mean that  $T$  is *internal* to the system, which would require the application of additional physical laws to describe its electronic circuitry. The choice between these two options (and among others) depends on our design and analytical needs.

A great number of systems of engineering interest have a **single input**. In our motor-link example, our single input was the torque  $T$ . In many electronic systems, a single voltage source supplies external power, and so is taken to be the system’s single input. Even systems of great complexity can often be described as single input systems.

When a system has a single input, we will often use  $u$  to denote this variable.

### 7.1.3 Outputs

When designing and analyzing a system, certain dependent variables will be of particular interest. We call such variables **outputs**.

Often, only a single variable is of interest. In such cases, we say we have a **single output** system and we often denote the output with the symbol  $y$ . For instance, perhaps in our motor-link example we are interested in the angle  $\theta$ , which we would then call an output.

It turns out we’re ignoring another class of variable<sup>1</sup> that we’ll learn more about in Mechatronics. For now, let’s assume that we’re interested in every dependent variable, other than inputs, in our system.

An objection might be raised, here: how can it be that a single output  $y$  can describe the output of many systems if we’re also going to take every dependent variable as an output? Won’t more variables be required to describe the dynamics? For instance, if, in the motor-link example, we take the system to include the motor, we’ll probably need the voltage and current therein to describe it. Together with  $\theta$ , that’s *three* outputs!

1. Variables of this class are called *state variables*.

It turns out this can be assuaged by algebraic relationships among the variables. For instance, the current through the motor windings is proportional to the torque. Through such relationships, we can have our cake (our single output) and eat most of it (eliminate extra dependent variables), too. The catch is that the **order** (highest derivative) of the differential equation describing a system typically increases through this process of variable elimination.

#### Box 7.4

##### Course connections: Mechatronics and System Dynamics and Control

We will learn in Mechatronics that we can express a system's dynamics as  $n \in \mathbb{N}$  coupled first-order equations or as a single  $n$ th-order equation. In System Dynamics and Control, we'll learn to solve the coupled equations. In this primer, we'll learn to solve the single  $n$ th-order equation.

### 7.1.4 SISO Linear Systems

The result of all this is that we frequently encounter **single-input, single-output** (SISO) linear systems. The input-output dynamics of all these systems can be described by the linear ODE with constant coefficients<sup>2</sup>  $a_i, b_j \in \mathbb{R}$  (which include such system parameters as mass and spring constants, capacitances and resistances, etc.), order  $n$ , and  $m \leq n$  for  $n, m \in \mathbb{N}_0$ —as:

$$\begin{aligned} \frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_1 \frac{dy}{dt} + a_0 y = \\ b_m \frac{d^m u}{dt^m} + b_{m-1} \frac{d^{m-1} u}{dt^{m-1}} + \cdots + b_1 \frac{du}{dt} + b_0 u. \end{aligned} \quad (7.1)$$

The rest of this primer will take as its primary goal the description of a solution technique for equation (7.1). **Solutions** will be functions  $y(t)$  that satisfy equation (7.1) in terms of parameters  $a_i, b_i$ , and input  $u(t)$ , only.

### 7.2 A Unique Solution Exists

We're not yet sure if a solution even **exists** for equation (7.1), and if it does, if it is **unique**—meaning it's the only solution.



2. The restriction of the coefficients to temporal constants means we can add the qualifier “time-invariant” to such systems, which we will do in Mechatronics.

### 7.2.1 Existence and Uniqueness

Rather than proving the existence and uniqueness of a solution, we will simply consider a theorem that states conditions under which existence and uniqueness do hold. In other words: *a unique solution exists*, and we'll explore the conditions for which this is true.

Let the **forcing function**  $f$  be the “right-hand side” of equation (7.1):

$$f(t) \equiv b_m \frac{d^m u}{dt^m} + b_{m-1} \frac{d^{m-1} u}{dt^{m-1}} + \cdots + b_1 \frac{du}{dt} + b_0 u.$$

#### Theorem 7.1

**Existence and Uniqueness** A solution  $y(t)$  of equation (7.1) *exists* and is *unique* for  $t \geq t_0$  if and only if both the following are specified:

1.  $n$  initial conditions

$$y(t_0), \left. \frac{dy}{dt} \right|_{t=t_0}, \dots, \left. \frac{d^{n-1}y}{dt^{n-1}} \right|_{t=t_0}$$

2. a continuous forcing function  $f(t)$  for  $t \geq t_0$ .

Assuming this theorem can be proved, and it can (**Finan2018**), we need only the initial conditions and the forcing function to guarantee ourselves there is a unique solution. Let's think about what this means in terms of the dynamics of a system. In a sense, if we know its initial state and the input or forcing<sup>3</sup>, how it will behave for the rest of time is determined. When I say “in a sense,” I mean that insofar as the system is well-described by equation (7.1). The determinist implications of this must be understood to be approximate and limited in scope. I don't want to be responsible for creating a bunch of determinists (**Hoefer2016**).

Note however, that, given initial conditions and forcing, we only know *that* a unique solution exists, not *what* that solution is or *how* to find it.

### 7.2.2 Outlining a Solution Technique

It turns out that, given a forcing function and no initial conditions, several potential solutions can satisfy the ODE equation (7.1); conversely, given certain initial conditions and no forcing function, several potential solutions satisfy the ODE. It is only when both initial conditions and a forcing function are given that a unique solution exists. It can be shown (**Kreyszig2010**) that the **general solution**  $y_g$  (also called the *total solution*)—actually a “family” of solutions with unknown constants—to

3. Usually, if we know the input  $u$ , it is trivial to apply ?? to find the forcing function  $f$ . However, note that  $u$  must be differentiable  $m$  times.



equation (7.1) is equal to the sum of two solutions that are often relatively easy to obtain:

1. the **homogeneous solution**  $y_h$ , another family of solutions, this time to equation (7.1) with  $f(t) = 0$ , and
2. the **particular solution**  $y_p$ , which satisfies equation (7.1) sans initial conditions.

That is,

$$y_g(t) = y_h(t) + y_p(t).$$

Methods for deriving homogeneous and particular solutions are the topics of (**lec:homogeneous\_solution**) and (**lec:particular\_solution**).

The general solution  $y_g$  is still a *family* of solutions that all satisfy equation (7.1) for a given forcing function  $f$ . It only becomes the *unique* solution, which we call the **specific solution** and typically denote simply  $y$  or (occasionally)  $y_s$ , once the initial conditions are applied to  $y_g$ .

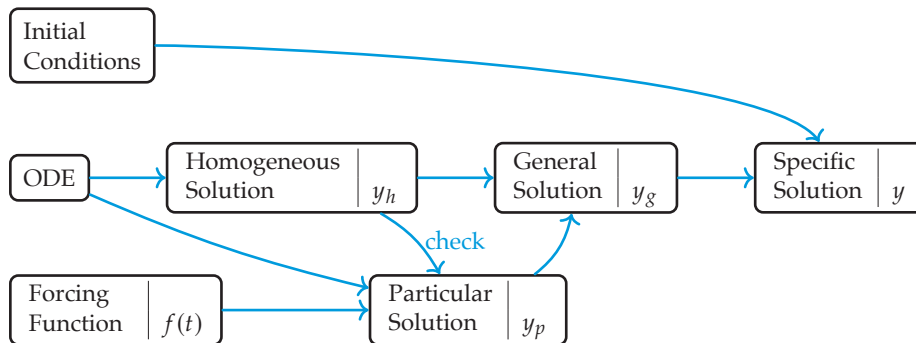


Figure 7.2. A diagram of the solution technique. Each arrow signifies that the block at its tail is supplied to and precedes the block at its head. The first column includes everything required to obtain a unique solution. The homogeneous  $y_h$  and particular  $y_p$  solutions of the second column sum to the general solution  $y_g$ . Applying the initial conditions to this yields the specific solution  $y$ .

The diagram of figure 7.2 illustrates this solution technique, with each arrow signifying that the block at its tail is supplied to and precedes the block at its head. Lectures proceed with the diagram:

- (**lec:homogeneous\_solution**) describes how to obtain the homogeneous solution  $y_h$  from equation (7.1) with the forcing function  $f(t) = 0$ ;

- **(lec:particular\_solution)** describes how to derive the particular solution  $y_p$  from equation (7.1) without the initial conditions for common forcing functions by a method called *undetermined coefficients*;
- **(lec:general\_solution)** blows your mind by summing the homogeneous and particular solutions to obtain the general solution  $y_g$ ; lest we be accused of dereliction of our duty to appear smarter than Business majors, this lecture also applies the initial conditions to the general solution to find constants introduced in the homogeneous solution to fully solve the differential equation—i.e., to obtain the specific solution  $y$ .

### Box 7.5

#### Course connection: Differential Equations

The technique outlined here is probably quite similar to one described in your Differential Equations course. Terminology and notation may be different, so it may be worth correlating this primer with your previous coursework and text.

### 7.3 Homogeneous Solution



The **homogeneous solution** (also called the *complementary solution*) to equation (7.1) is the family of solutions<sup>4</sup> that satisfy equation (7.1) with the forcing function  $f(t) = 0$ , but is not restricted by the initial conditions. The equation is

$$\frac{d^n y}{dt^n} + a_{n-1} \frac{d^{n-1} y}{dt^{n-1}} + \cdots + a_1 \frac{dy}{dt} + a_0 y = 0. \quad (7.2)$$

What function might solve this, the **homogeneous equation**?

The natural exponential with base  $e$  is a good candidate, since it is its own derivative. It turns out that a **linear combination** (weighted sum) of **linearly independent** exponentials<sup>5</sup> is the family of solutions we're looking for.

4. This family of solutions is actually the *general* solution of the homogeneous equation, equation (7.2). Lest it be confused with the general solution of equation (7.1), we avoid calling it this in the following.

5. The special case of repeated characteristic equation roots requires a slight modification of this statement, as we'll see. In this case, some of the exponentials pick up extra factors.

### 7.3.1 Characteristic Equation and Its Roots

It can be shown that, for an exponential function  $Ce^{\lambda t}$  with complex  $C$  and  $\lambda$ , the latter must satisfy

$$\lambda^n + a_{n-1}\lambda^{n-1} + \cdots + a_1\lambda + a_0 = 0, \quad (7.3)$$

called the **characteristic equation**.

It has  $n$  solutions or **roots**  $\lambda_i$ .

When these roots are all **distinct**—meaning none is equal to another—the **homogeneous solution**  $y_h$  to equation (7.1) is

$$y_h(t) = \sum_{i=1}^n C_i e^{\lambda_i t}. \quad (7.4)$$

### 7.3.2 Repeated Roots

If a root is not distinct, it is said to have **multiplicity**  $\mu$  equal to the number of its instances.

So a root that appears thrice has multiplicity three.

This multiplicity causes the linear combination of exponentials to be **degenerate** or **linearly dependent**.

This is easily remedied, however, by augmenting the solution of equation (7.4) with a polynomial term in  $t$ , as follows.

Let there be  $n'$  distinct roots, each with multiplicity  $\mu_i$ .

Then the solution is:<sup>6</sup>

$$y_h(t) = \sum_{i=1}^{n'} \sum_{k=1}^{\mu_i} C_{ik} t^{(k-1)} e^{\lambda_{ik} t}. \quad (7.5)$$

Note that, as we would expect, when all roots are distinct, the factor  $t$  in each term has exponent zero,  $t^0 = 1$ , and we recover equation (7.4).

6. We use a double subscript  $ik$ , here, meaning the  $i$ th distinct root and the  $k$ th copy. Don't be alarmed: it's just to make sure there are enough distinct constants. If one prefers, the constants can be numbered 1 through  $n$ , but it's difficult to write that in summation form.

### 7.3.3 What Have We Done?

We have found the homogeneous solution to equation (7.1), which we know sums with another term to form its general solution.

We should probably consider what this homogeneous solution looks like.

It's a sum of weighted *complex* exponential functions of time.

Complex solutions to the characteristic equation, equation (7.3), always arise in complex conjugate pairs  $\sigma \pm j\omega$ , yielding terms like

$$e^{(\sigma+j\omega)t} + e^{(\sigma-j\omega)t} = e^{\sigma t} (e^{j\omega t} + e^{-j\omega t}) = 2e^{\sigma t} \cos(\omega t).$$

This last identity is from a form of **Euler's formula**.

The result shows us the possible forms of terms in the homogeneous solution:

- For a real root,  $\omega = 0$  and a real exponential results (note, also, that the 2 goes away).
- For imaginary roots,  $\sigma = 0$  and a sinusoid results.
- For complex roots, a sinusoidal oscillation with an exponential envelope occurs.

Everything we know about the exponential also applies.

For instance, if  $\sigma < 0$ , an exponential *decay* results, whereas if  $\sigma > 0$ , we get an exponential *growth*.

#### Example 7.1

##### A homogeneous solution

Find the homogeneous solution for the equation

$$\frac{d^5 y}{dt^5} + 14 \frac{d^4 y}{dt^4} + 81 \frac{d^3 y}{dt^3} + 248 \frac{d^2 y}{dt^2} + 408 \frac{dy}{dt} + 288 y = f(t).$$

First, we write the characteristic equation:

$$\lambda^5 + 14\lambda^4 + 81\lambda^3 + 248\lambda^2 + 408\lambda + 288 = 0.$$

Your favorite root finder (e.g. Matlab, Wolfram Alpha, many calculators) will yield, in arbitrary order:

$$\lambda_{11} = -4,$$

$$\lambda_{21,22} = -3,$$

$$\lambda_{31,41} = -2 \pm j2.$$

So  $\lambda_{11}$  is a real root,  $\lambda_{21}$  and  $\lambda_{22}$  are repeated real roots with multiplicity  $\mu = 2$ , and  $\lambda_{31}$  and  $\lambda_{41}$  are complex conjugate roots.

So the homogeneous solution is, *à la* equation (7.5):

$$y_h(t) = \sum_{i=1}^{n'} \sum_{j=1}^{\mu_i} C_i t^{(j-1)} e^{\lambda_i t}$$

$$= C_{11} e^{\lambda_{11} t} + C_{21} e^{\lambda_{21} t} + C_{22} t e^{\lambda_{22} t} + C_{31} e^{\lambda_{31} t} + C_{41} e^{\lambda_{41} t}.$$

Now that we've applied our equation, we can drop the double-subscripting, if we like:

$$y_h(t) = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} + C_3 t e^{\lambda_3 t} + C_4 e^{\lambda_4 t} + C_5 e^{\lambda_5 t},$$

where we map indices as follows:

$$1 \rightarrow 11,$$

$$2 \rightarrow 21,$$

$$3 \rightarrow 21,$$

$$4 \rightarrow 31,$$

$$5 \rightarrow 41.$$

Once we're comfortable dealing with multiplicity, we can skip the strict application of equation (7.5) and use single subscripts.

## 7.4 Particular Solution



What effect does the forcing function  $f$  have on the solution?

How might we solve for this effect, called the **particular solution**?

One answer to the latter question will be given in this lecture: using the **method of undetermined coefficients**.

Before we turn to this method, please recognize that other methods, such as the *method of variation* or *Laplace transforms* apply to more general forms of forcing  $f$  (although the integrals that accompany each may be unknown).

When choosing the method of undetermined coefficients, we limit the scope of applicability to systems subjected to forcing functions that are complex exponentials (which include sinusoids) or polynomials.

The principle of *superposition*, discussed in the Mechatronics course, allows us to construct solutions for linear systems subject to linear combinations of complex exponentials and polynomials.

### 7.4.1 Method of Undetermined Coefficients

The method is:

1. Based on the form of the forcing function, **propose** an appropriate solution that includes undetermined coefficients (being careful to propose a solution linearly independent of the homogeneous solution),
2. Substitute this proposed solution into the ODE, and
3. Determine the undetermined coefficients by solving the algebraic system of equations that results from equating terms on each side of the equation.

If there is, in fact, a solution to the algebraic system—that is, for the undetermined coefficients—*our proposed solution is our particular solution*, with coefficients now determined.

However, if there is no solution,<sup>7</sup>, our proposed solution is *not* our particular solution.

### 7.4.2 Some Suggested Solution Proposals

How can one propose a solution?

There are no clear answers other than “be clever or use known solutions.”

As remarkably unsatisfying as this is, we can still rejoice in being let off the hook, since we are certainly not clever.

As mentioned above, this method only really works if the forcing function is a complex exponential or a polynomial (but this can be extended, using superposition, to a large class of problems of interest).

(**tab:undetermined\_coefficients**) is provided as a guide, but it essentially boils down to: if  $f$  is a complex exponential, propose that  $y_p$  is a complex exponential; if  $f$  is a polynomial, propose that  $y_p$  is a polynomial.

**Suggested particular solutions  $y_p(t)$  (with undetermined coefficients) to propose for various forcing functions  $f$ .**

Let  $k$ ,  $\lambda$ ,  $\omega$ , and  $\phi$  be real constants and  $n$  be a positive integer. Furthermore, let  $K_i$  be the *undetermined coefficients*.

$k$	$K_1$	$0$
$kt^n$	$K_n t^n + K_{n-1} t^{n-1} + \cdots + K_0$	$0$
$ke^{\lambda t}$	$K_1 e^{\lambda t}$	$\lambda$
$ke^{j\omega t}$	$K_1 e^{j\omega t}$	$j\omega$
$k \cos(\omega t + \phi)$	$K_1 \cos(\omega t) + K_2 \sin(\omega t)$	$j\omega$
$k \sin(\omega t + \phi)$	$K_1 \cos(\omega t) + K_2 \sin(\omega t)$	$j\omega$

7. One should not simply throw up one's hands at a certain point and declare “there's no solution!” Rather, one should prove that there is none.

### 7.4.3 The Parenthetical Caveat

The only caveat, here, is the parenthetical warning from the three-step method about choosing a linearly independent solution.

This is a result of a theorem we have not considered, here, but suffice it to say that, in order for our *general* solution to simply be the sum of the homogeneous and particular solutions, as we will propose in the next lecture, these two must be linearly independent.

We will not only skip the details of why this is the case, but also the details of how to deal with it, opting instead for a simple recipe.

The “test values” in (**tab:undetermined\_coefficients**) are to test whether or not the particular solution is a component of the homogeneous solution.

If the test value is equal to any root of the characteristic equation of multiplicity  $\mu$ , then the proposed solution should be multiplied by  $t^\mu$ .

#### Example 7.2

##### A particular solution

Find the particular solution for the equation

$$\frac{d^5 y}{dt^5} + 14 \frac{d^4 y}{dt^4} + 81 \frac{d^3 y}{dt^3} + 248 \frac{d^2 y}{dt^2} + 408 \frac{dy}{dt} + 288y = f(t),$$

which is the same as that of (**ex:homogeneous1**), with

$$f(t) = a \cos(\omega t),$$

with  $a \in \mathbb{R}$  and  $\omega = 5 \text{ rad/s}$ .

We needn't re-solve for the roots of the characteristic equation, since we did that in (**ex:homogeneous1**).

They were

$$\lambda_{11} = -4,$$

$$\lambda_{21,22} = -3,$$

$$\lambda_{31,41} = -2 \pm j2.$$

We now consult (**tab:undetermined\_coefficients**).

The forcing function is in the fifth row with  $k = a$ ,  $\omega = \omega$ , and  $\phi = 0$ .

So our proposed particular solution is

$$y_p(t) = K_1 \cos(\omega t) + K_2 \sin(\omega t),$$

where  $K_1$  and  $K_2$  are our undetermined coefficients.

Note that our table indicates we should test that  $j\omega = j5$  is not a root of the characteristic equation, which it is not ( $j5 \neq \lambda_i$ ).

We now substitute  $y \mapsto y_p$  into the ODE:

$$\frac{d^5 y_p}{dt^5} + 14 \frac{d^4 y_p}{dt^4} + 81 \frac{d^3 y_p}{dt^3} + 248 \frac{d^2 y_p}{dt^2} + 408 \frac{dy_p}{dt} + 288 y_p = a \cos(\omega t).$$

The five derivatives are:

$$\frac{dy_p}{dt} = -K_1 \omega \sin(\omega t) + K_2 \omega \cos(\omega t)$$

$$\frac{d^2 y_p}{dt^2} = -K_1 \omega^2 \cos(\omega t) - K_2 \omega^2 \sin(\omega t)$$

$$\frac{d^3 y_p}{dt^3} = K_1 \omega^3 \sin(\omega t) - K_2 \omega^3 \cos(\omega t)$$

$$\frac{d^4 y_p}{dt^4} = K_1 \omega^4 \cos(\omega t) + K_2 \omega^4 \sin(\omega t)$$

$$\frac{d^5 y_p}{dt^5} = -K_1 \omega^5 \sin(\omega t) + K_2 \omega^5 \cos(\omega t)$$

Substituting into the ODE, we collect cosine and sine terms and equate them with the right-hand side:

$$K_2 \omega^5 + 14K_1 \omega^4 - 81K_2 \omega^3 - 248K_1 \omega^2 + 408K_2 \omega + K_1 = a$$

$$-K_1 \omega^5 + 14K_2 \omega^4 + 81K_1 \omega^3 - 248K_2 \omega^2 - 408K_1 \omega + K_2 = 0$$

This is a linear system of two equations in two unknowns  $K_1$  and  $K_2$ . Solving it (with  $\omega = 5$  rad/s) gives:

$$K_1 \approx (82.00 \cdot 10^{-6})a, \quad K_2 \approx (-159.4 \cdot 10^{-6})a$$

Since there *is* an algebraic solution, we know our proposed particular solution is, in fact, a particular solution!

And, of course, our coefficients  $K_1$  and  $K_2$  are now determined.



## 7.5 General and Specific Solutions



We posited in (lec:unique\_solution\_exists) that the **general solution** to the ODE equation (7.1) is

$$y_g(t) = y_h(t) + y_p(t). \quad (7.6)$$

We have not and will not prove this, but simply propose it to be the case. Working through a proof of this from, for instance, your differential equations textbook is of some value.

So, we already have  $y_h$  and  $y_p$ , so finding  $y_g$  is trivial.

What type of object is  $y_g$ ?

The particular solution contributes only determined coefficients, but the homogeneous solution contributes  $n$  “unknown” constants  $C_i$ .

This means  $y_g$  inherits those constants and therefore is a *family* of solutions.

This leads us to our final step: applying the initial conditions to find the specific constants  $C_i$  and thereby our **specific solution**  $y$ .

“Applying” the initial conditions is simply to subject  $y_g$  to each of them. For instance, if we have two initial conditions, such as

$$y(0) = 2 \quad \text{and} \quad \left. \frac{dy}{dt} \right|_{t=0} = 0,$$

we construct two algebraic equations

$$y_g(0) = 2 \quad \text{and} \quad \left. \frac{dy_g}{dt} \right|_{t=0} = 0,$$

which is a system of algebraic equations from which the two unknown constants  $C_1$  and  $C_2$  (from the homogeneous solution) can be solved.

### Example 7.3

#### A general and a specific solution

Find the general solution for the equation

$$\frac{d^2 y}{dt^2} + 5 \frac{dy}{dt} + 6y = f(t),$$

with

$$f(t) = a \cos(\omega t),$$

where  $a \in \mathbb{R}$  and  $\omega = 5 \text{ rad/s}$ .

Apply the following initial conditions to obtain a specific solution:

$$y(0) = 3 \quad \text{and} \quad \left. \frac{dy}{dt} \right|_{t=0} = 0.$$

First, let's find the homogeneous solution via the characteristic equation:

$$\lambda^2 + 5\lambda + 6 = 0$$

and its roots:

$$\lambda_{1,2} = -2, -3.$$

Both of these are real, so:

$$y_h(t) = C_1 e^{-2t} + C_2 e^{-3t}.$$

The particular solution should have the form:

$$y_p(t) = K_1 \cos(\omega t) + K_2 \sin(\omega t),$$

where  $K_1$  and  $K_2$  are our undetermined coefficients.

Substituting this into the ODE requires two derivatives:

$$\frac{dy_p}{dt} = -K_1 \omega \sin(\omega t) + K_2 \omega \cos(\omega t)$$

$$\frac{d^2 y_p}{dt^2} = -K_1 \omega^2 \cos(\omega t) - K_2 \omega^2 \sin(\omega t)$$

Substitution leads to the following algebraic system by equating cosine and sine terms:

$$-K_1 \omega^2 + 5K_2 \omega + 6K_1 = a$$

$$-K_2 \omega^2 - 5K_1 \omega + 6K_2 = 0$$

Solving gives:

$$K_1 = -\frac{a(\omega^2 - 6)}{\omega^4 + 13\omega^2 + 36} = -\frac{19a}{986}$$

$$K_2 = \frac{5a\omega}{\omega^4 + 13\omega^2 + 36} = \frac{25a}{986}$$

So the particular solution is:

$$y_p(t) = \frac{a}{986} (-19 \cos(5t) + 25 \sin(5t)).$$

The general solution is:

$$\begin{aligned}y_g(t) &= y_h(t) + y_p(t) \\&= C_1 e^{-2t} + C_2 e^{-3t} + \frac{a}{986}(-19 \cos(5t) + 25 \sin(5t)).\end{aligned}$$

Applying the initial conditions:

$$\begin{aligned}C_1 + C_2 - \frac{19a}{986} &= 3 \\-2C_1 - 3C_2 + \frac{125a}{986} &= 0\end{aligned}$$

Solving:

$$C_1 = \frac{-2a}{29} + 9, \quad C_2 = \frac{3a}{34} - 6$$

So the specific solution is:

$$y(t) = \left(\frac{-2a}{29} + 9\right) e^{-2t} + \left(\frac{3a}{34} - 6\right) e^{-3t} + \frac{a}{986}(-19 \cos(5t) + 25 \sin(5t)).$$

## 7.6 Problems



**Problem 7.1** For the following sets of parameters, find the homogeneous solution  $y_h$  for equation (7.1) with the order  $n$  and coefficients  $a_i$  given.

1.  $n = 2, a_1 = -1, a_0 = -2$
2.  $n = 2, a_1 = 6, a_0 = 9$
3.  $n = 2, a_1 = 10, a_0 = 34$
4.  $n = 5, a_4 = -7, a_3 = 32, a_2 = -124, a_1 = 256, a_0 = -192$

**Problem 7.2** For the following sets of parameters, find the particular solution  $y_p$  for equation (7.1) with the order  $n$ , coefficients  $a_i$ , and forcing function  $f$  given.

1.  $n = 2, a_1 = -1, a_0 = -2, f(t) = 3$
2.  $n = 2, a_1 = 6, a_0 = 9, f(t) = 5e^{-3t}$
3.  $n = 1, a_0 = 2, f(t) = 2 \cos(3t)$
4.  $n = 3, a_2 = 5, a_1 = 16, a_0 = 80, f(t) = t + 2$

**Problem 7.3** For the following sets of parameters, find the specific solution  $y$  for equation (7.1) with the order  $n$ , coefficients  $a_i$ , forcing function  $f$ , and initial conditions given.

Note that the homogeneous and particular solutions from problem 7.2 apply to these problems, so they need not be re-derived.

1.  $n = 2, a_1 = -1, a_0 = -2, f(t) = 3, y(0) = 2, dy/dt|_{t=0} = 0$
2.  $n = 2, a_1 = 6, a_0 = 9, f(t) = 5e^{-3t}, y(0) = 0, dy/dt|_{t=0} = 0$
3.  $n = 1, a_0 = 2, f(t) = 2 \cos(3t), y(0) = 4$
4.  $n = 3, a_2 = 5, a_1 = 16, a_0 = 80, f(t) = t + 2, y(0) = 0, dy/dt|_{t=0} = 1, d^2y/dt^2|_{t=0} = 0$

# 8 Partial Differential Equations



An **ordinary differential equation** is one with (ordinary) derivatives of functions of a single variable each—time, in many applications. These typically describe quantities in some sort of **lumped-parameter** way: mass as a “point particle,” a spring’s force as a function of time-varying displacement across it, a resistor’s current as a function of time-varying voltage across it. Given the simplicity of such models in comparison to the wildness of nature, it is quite surprising how well they work for a great many phenomena. For instance, electronics, rigid body mechanics, population dynamics, bulk fluid mechanics, and bulk heat transfer can be lumped-parameter modeled.

However, as we saw in chapter 5, there are many phenomena of which we require more detailed models. These include the following:

- Detailed fluid mechanics
- Detailed heat transfer
- Solid mechanics
- Electromagnetism
- Quantum mechanics

In many cases, what is required to account for is the **time-varying spatial distribution** of a quantity. In fluid mechanics, we treat a fluid as having quantities such as density and velocity that vary continuously over space and time. Deriving the governing equations for such phenomena typically involves vector calculus; we observed in chapter 5 that statements about quantities like the divergence (e.g., continuity) can be made about certain scalar and vector fields. Such statements are governing equations (e.g., the continuity equation) and they are **partial differential equations** (PDEs) because the quantities of interest, called **dependent variables** (e.g., density and velocity), are both temporally and spatially varying (temporal and spatial variables are therefore called **independent variables**).

In this chapter, we explore the **analytic solution** of PDEs. This is related to but distinct from the **numeric solution** (i.e., simulation) of PDEs, which is another

important topic. Many PDEs have no known analytic solution, so for these numeric solution is the best available option.<sup>1</sup> However, it is important to note that the insight one can gain from an analytic solution is often much greater than that from a numeric solution. This is easily understood when one considers that a numeric solution is an approximation for a specific set of initial and boundary conditions. Typically, very little can be said of what would happen in general, although this is often what we seek to know. So, despite the importance of numeric solution, one should always prefer an analytic solution.

Three good texts on PDEs for further study are Kreyszig (2011; chapter 12), Strauss (2007), and Haberman (2018).

## 8.1 Classifying PDEs



PDEs often have an infinite number of solutions; however, when applying them to physical systems, we usually assume that a deterministic, or at least a probabilistic, sequence of events will occur. Therefore, we impose additional constraints on a PDE, usually in the form of

1. **Initial conditions**, values of independent variables over all space at an initial time and
2. **Boundary conditions**, values of independent variables (or their derivatives) over all time.

Ideally, imposing such conditions leaves us with a **well-posed problem**, which has three aspects (Bove, Colombini, and Santo 2006; § 1.5):

**Existence** There exists at least one solution.

**Uniqueness** There exists at most one solution.

**Stability** If the PDE, boundary conditions, or initial conditions are changed slightly, the solution changes only slightly.

As with ODEs, PDEs can be **linear** or **nonlinear**; that is, the dependent variables and their derivatives can appear in only linear combinations (linear PDE) or in one or more nonlinear combination (nonlinear PDE). As with ODEs, there are more known analytic solutions to linear PDEs than nonlinear PDEs.

The **order** of a PDE is the order of its highest derivative. A great many physical models can be described by **second-order PDEs** or systems thereof. Let  $u$  be an independent scalar variable, a function of  $m$  temporal and spatial variables  $x_i$ . A second-order linear PDE has the form, for coefficients  $\alpha$ ,  $\beta$ ,  $\gamma$ , and  $\delta$ , and real

1. There are some analytic techniques for gaining insight into PDEs for which there are no known solutions, such as considering the *phase space*. This is an active area of research; for more, see Bove, Colombini, and Santo (2006).

functions of  $x_i$ , (Strauss 2007; § 1.6)

$$\underbrace{\sum_{i=1}^m \sum_{j=1}^m \alpha_{ij} \partial_{x_i x_j}^2 u}_{\text{second-order terms}} + \underbrace{\sum_{k=1}^m (\gamma_k \partial_{x_k} u + \delta_k u)}_{\text{first- and zeroth-order terms}} = \underbrace{f(x_1, \dots, x_m)}_{\text{forcing}} \quad (8.1)$$

where  $f$  is called a **forcing function**. When  $f$  is zero, equation (8.1) is called **homogeneous**. We can consider the coefficients  $\alpha_{ij}$  to be components of a matrix  $A$  with rows indexed by  $i$  and columns indexed by  $j$ . There are four prominent classes defined by the eigenvalues of  $A$ :

**Elliptic** The eigenvalues all have the same sign

**Parabolic** The eigenvalues have the same sign except one that is zero

**Hyperbolic** Exactly one eigenvalue has the opposite sign of the others

**Ultrahyperbolic** There are at least two eigenvalues of each sign

The first three of these have received extensive treatment. They are named after conic sections due to the similarity the equations have with polynomials when derivatives are considered analogous to powers of polynomial variables. For instance, here is a case of each of the first three classes,

$$\partial_{xx}^2 u + \partial_{yy}^2 u = 0 \quad (\text{elliptic})$$

$$\partial_{xx}^2 u - \partial_{yy}^2 u = 0 \quad (\text{hyperbolic})$$

$$\partial_{xx}^2 u - \partial_t u = 0. \quad (\text{parabolic})$$

When  $A$  depends on  $x_i$ , it may have multiple classes across its domain. In general, this equation and its associated initial and boundary conditions do not comprise a well-posed problem; however several special cases have been shown to be well-posed. Thus far, the most general statement of existence and uniqueness is the **cauchy-kowalevski theorem** for **cauchy problems**.

## 8.2 Boundary Value Problems



Before we introduce an important solution method for PDEs in section 8.3, we consider an *ordinary* differential equation that will arise in that method when dealing with a single spatial dimension  $x$ : the **sturm-liouville (S-L) differential equation**. Let  $p, q, \sigma$  be functions of  $x$  on open interval  $(a, b)$ . Let  $X$  be the dependent variable and  $\lambda$  constant. The **regular S-L problem** is the S-L ODE<sup>2</sup>

$$\frac{d}{dx}(pX') + qX + \lambda\sigma X = 0 \quad (8.2)$$

with boundary conditions

$$\beta_1 X(a) + \beta_2 X'(a) = 0 \quad (8.3)$$

$$\beta_3 X(b) + \beta_4 X'(b) = 0 \quad (8.4)$$

with coefficients  $\beta_i \in \mathbb{R}$ . This is a type of **boundary value problem**.

This problem has nontrivial solutions, called **eigenfunctions**  $X_n(x)$  with  $n \in \mathbb{Z}_+$ , corresponding to specific values of  $\lambda = \lambda_n$  called **eigenvalues**.<sup>3</sup> There are several important theorems proven about this (see (Haberman 2018; § 5.3)). Of greatest interest to us are that

1. There exist an infinite number of eigenfunctions  $X_n$  (unique within a multiplicative constant)
2. There exists a unique corresponding *real* eigenvalue  $\lambda_n$  for each eigenfunction  $X_n$
3. The eigenvalues can be ordered as  $\lambda_1 < \lambda_2 < \dots$
4. Eigenfunction  $X_n$  has  $n - 1$  zeros on open interval  $(a, b)$
5. The eigenfunctions  $X_n$  form an orthogonal basis with respect to weighting function  $\sigma$  such that any piecewise continuous function  $f : [a, b] \rightarrow \mathbb{R}$  can be represented by a generalized fourier series on  $[a, b]$

This last theorem will be of particular interest in section 8.3.

2. For the S-L problem to be *regular*, it has the additional constraints that  $p, q, \sigma$  are continuous and  $p, \sigma > 0$  on  $[a, b]$ . This is also sometimes called the sturm-liouville eigenvalue problem. See (Haberman 2018; § 5.3) for the more general (non-regular) S-L problem and (§ 7.4) for the multi-dimensional analog.

3. These eigenvalues are closely related to, but distinct from, the “eigenvalues” that arise in systems of linear ODEs.



### 8.2.1 Types of Boundary Conditions

Boundary conditions of the sturm-liouville kind equation (8.3) have four sub-types:

**Dirichlet** for just  $\beta_2, \beta_4 = 0$ ,

**Neumann** for just  $\beta_1, \beta_3 = 0$ ,

**Robin** for all  $\beta_i \neq 0$ , and

**Mixed** if  $\beta_1 = 0, \beta_3 \neq 0$ ; if  $\beta_2 = 0, \beta_4 \neq 0$ .

There are many problems that are *not* regular sturm-liouville problems. For instance, the right-hand sides of equation (8.3) are zero, making them **homogeneous boundary conditions**; however, these can also be nonzero. Another case is **periodic boundary conditions**:

$$X(a) = X(b) \quad (8.5)$$

$$X'(a) = X'(b). \quad (8.6)$$

#### Example 8.1

Consider the differential equation

$$X'' + \lambda X = 0$$

with dirichlet boundary conditions on the boundary of the interval  $[0, L]$

$$X(0) = 0 \quad \text{and} \quad X(L) = 0.$$

Solve for the eigenvalues and eigenfunctions.

This is a sturm-liouville problem, so we know the eigenvalues are real. The well-known general solution to the ODE is

$$X(x) = \begin{cases} k_1 + k_2 x & \lambda = 0 \\ k_1 e^{j\sqrt{\lambda}x} + k_2 e^{-j\sqrt{\lambda}x} & \text{otherwise} \end{cases}$$

with real constants  $k_1, k_2$ . The solution must also satisfy the boundary conditions. Let's apply them to the case of  $\lambda = 0$  first:

$$X(0) = 0 \Rightarrow k_1 + k_2(0) = 0 \Rightarrow k_1 = 0$$

$$X(L) = 0 \Rightarrow k_1 + k_2(L) = 0 \Rightarrow k_2 = -k_1/L.$$

Together, these imply  $k_1 = k_2 = 0$ , which gives the *trivial solution*  $X(x) = 0$ , in which we aren't interested. We say, then, that for nontrivial solutions,  $\lambda \neq 0$ . Now let's check  $\lambda < 0$ . The solution becomes

$$\begin{aligned} X(x) &= k_1 e^{-\sqrt{|\lambda|x}} + k_2 e^{\sqrt{|\lambda|x}} \\ &= k_3 \cosh(\sqrt{|\lambda|x}) + k_4 \sinh(\sqrt{|\lambda|x}) \end{aligned}$$

where  $k_3$  and  $k_4$  are real constants. Again applying the boundary conditions:

$$X(0) = 0 \Rightarrow k_3 \cosh(0) + k_4 \sinh(0) = 0 \Rightarrow k_3 + 0 = 0 \Rightarrow k_3 = 0$$

$$X(L) = 0 \Rightarrow 0 \cosh(\sqrt{|\lambda|}L) + k_4 \sinh(\sqrt{|\lambda|}L) = 0 \Rightarrow k_4 \sinh(\sqrt{|\lambda|}L) = 0.$$

However,  $\sinh(\sqrt{|\lambda|}L) \neq 0$  for  $L > 0$ , so  $k_4 = k_3 = 0$ —again, the trivial solution. Now let's try  $\lambda > 0$ . The solution can be written

$$X(x) = k_5 \cos(\sqrt{\lambda}x) + k_6 \sin(\sqrt{\lambda}x).$$

Applying the boundary conditions for this case:

$$X(0) = 0 \Rightarrow k_5 \cos(0) + k_6 \sin(0) = 0 \Rightarrow k_5 + 0 = 0 \Rightarrow k_5 = 0$$

$$X(L) = 0 \Rightarrow 0 \cos(\sqrt{\lambda}L) + k_6 \sin(\sqrt{\lambda}L) = 0 \Rightarrow k_6 \sin(\sqrt{\lambda}L) = 0.$$

Now,  $\sin(\sqrt{\lambda}L) = 0$  for

$$\sqrt{\lambda}L = n\pi \Rightarrow$$

$$\lambda = \left(\frac{n\pi}{L}\right)^2. \quad (n \in \mathbb{Z}_+)$$

Therefore, the only nontrivial solutions that satisfy both the ODE and the boundary conditions are the *eigenfunctions*

$$X_n(x) = \sin\left(\sqrt{\lambda_n}x\right) \quad (8.7)$$

$$= \sin\left(\frac{n\pi}{L}x\right) \quad (8.8)$$

with corresponding *eigenvalues*

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2.$$

Note that because  $\lambda > 0$ ,  $\lambda_1$  is the lowest eigenvalue.

### Plotting the Eigenfunctions

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

Set  $L = 1$  and compute values for the first four eigenvalues `lambda_n` and eigenfunctions `X_n`.

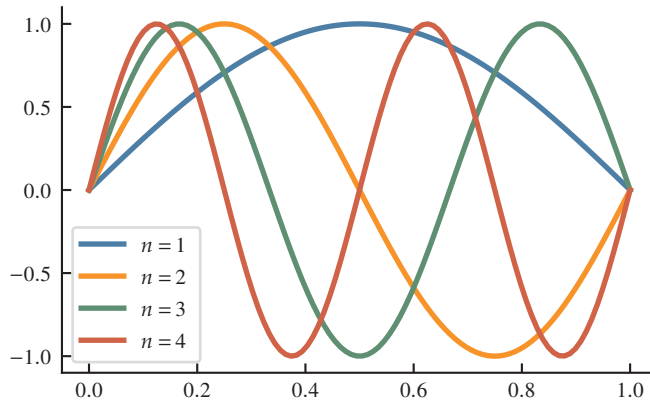
```

L = 1
x = np.linspace(0, L, 100)
n = np.linspace(1, 4, 4, dtype=int)
lambda_n = (n*np.pi/L)**2
X_n = np.zeros([len(n), len(x)])
for i, n_i in enumerate(n):
    X_n[i, :] = np.sin(np.sqrt(lambda_n[i])*x)

Plot the eigenfunctions.

fig, ax = plt.subplots()
for i, n_i in enumerate(n):
    ax.plot(x, X_n[i, :], linewidth=2, label='$n = '+str(n_i)+'$')
plt.legend()
plt.show()

```

Figure 8.1. Eigenfunctions  $X_n(x)$ .

We see that the fourth of the S-L theorems appears true:  $n - 1$  zeros of  $X_n$  exist on the open interval  $(0, 1)$ .

### 8.2.2 Irregular Sturm-Liouville Problems

Sturm-Liouville problems that do not satisfy the regularity conditions are called **irregular**. The nice theorems for regular S-L problems may not hold for irregular S-L problems. However, we can often still solve irregular S-L problems using the same methods as for regular problems.

### Example 8.2

Consider the ODE called **Bessel's equation** (Kreyszig 2011; § 5.4), for real independent variable  $s$  and dependent variable  $y(s)$ ,

$$s^2 y'' + s y' + (s^2 - \nu^2) y = 0.$$

This ODE arises in polar coordinate PDE models of circular membranes, where  $y$  is the membrane displacement and  $s = kr$ , and where  $r$  is the radius of the membrane and  $k$  is a constant sometimes called the wavenumber.

Consider the following boundary conditions:

- At radius  $r = R$ , the membrane is fixed, so  $y = 0$ .
- At radius  $r = 0$ , the membrane is free to move, but the displacement is finite, so  $|y| < \infty$ .

Prove that the Bessel equation is an irregular Sturm-Liouville problem and solve for the eigenvalues and eigenfunctions for the case  $\nu = 0$ .

We proceed to show that the Bessel equation with the given boundary conditions is an irregular Sturm-Liouville problem by first showing that the Bessel equation can be written in the form of a Sturm-Liouville problem ODE, then showing that the boundary conditions are not regular. Dividing the Bessel equation by  $s$  gives

$$s y'' + y' + \frac{s^2 - \nu^2}{s} y = 0 \implies \quad (z \neq 0)$$

$$\frac{d}{ds} (s y') + (s - \nu^2/s) y = 0. \quad (8.9)$$

So we see that this is equivalent to the Sturm-Liouville problem's ODE,

$$\frac{d}{dx} (p X') + (\lambda \sigma + q) X = 0, \quad (8.10)$$

with  $x = s$ ,  $X = y$ ,  $p(s) = s$ ,  $q(s) = -\nu^2/s$ ,  $\sigma(s) = s$ , and  $\lambda = 1$ .

Now let us consider the boundary conditions. The second boundary condition cannot be written as a linear combination of  $y$  and  $y'$  at  $s = 0$ , therefore this is an irregular Sturm-Liouville problem.

Solutions for Bessel's equation are Bessel functions of the first kind,  $J_\nu(s)$  (section 6.4), and Bessel functions of the second kind,  $Y_\nu(s)$  (§ 5.5). For  $\nu = 0$ , the Bessel equation simplifies to

$$s^2 y'' + s y' + s^2 y = 0$$

and the solutions are zeroth-order Bessel functions of the first kind,  $J_0(s)$ , and of the second kind,  $Y_0(s)$ . That is, the general solution is

$$y(s) = a J_0(s) + b Y_0(s).$$

However,  $Y_0(s)$  is singular at  $s = 0$ , so the boundary condition  $|y(0)| < \infty$  requires  $b = 0$ . Therefore, the eigenfunctions are

$$y_n(s) = J_0(s) = J_0(k_n r)$$

for eigenvalues  $\lambda_n = k_n^2$  and  $n \in \mathbb{Z}_+$ . On the boundary,  $y_n(kR) = 0$  implies that  $k_n R$  are the zeros of  $J_0(s)$ , what we called  $\alpha_{0,n}$  in example 6.4. Therefore, the eigenvalues are

$$\lambda_n = \left( \frac{\alpha_{0,n}}{R} \right)^2.$$

**Plotting the Eigenfunctions** We proceed in Python. First, load packages.

```
import numpy as np
import sympy as sp
import scipy
from scipy.special import jn_zeros
import matplotlib.pyplot as plt
```

The eigenvalues can be computed from the zeros of the Bessel function zeros  $\alpha_{0,n}$ , where  $n = 1, 2, 3, \dots$  as follows:

```
n = sp.symbols('n', integer=True, positive=True)
k, R = sp.symbols('k, R', positive=True, real=True)
r = sp.symbols('r', nonnegative=True)
J_0 = sp.besselj(0, k * r)
N_zeros = 5
alpha_0_n = jn_zeros(0, N_zeros)
params = {R: 1} # Set R = 1
lambda_n_ = (alpha_0_n / params[R])**2
print(f"The first {N_zeros} eigenvalues are:")
print(lambda_n_)
```

```
The first 5 eigenvalues are:
```

```
[ 5.78318596  30.47126234  74.88700679 139.04028443 222.93230362]
```

The eigenfunctions are given by the Bessel functions of the first kind  $J_0(k_n r)$ .

```
def k_n(lambda_n): return np.sqrt(lambda_n)
k_n_ = k_n(lambda_n_)
```

Plot the eigenfunctions.

```

r_plt = np.linspace(0, 1, 101)
print(J_0)
y_n_fun = sp.lambdify((r, k), J_0, modules=['numpy', 'scipy'])
fig, ax = plt.subplots()
for i in range(5):
    k_i = k_n[i]
    ax.plot(r_plt, y_n_fun(r_plt, k_i), label=f'$y_{i+1}(k_{i+1} r)$')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.set_xlabel('$r$')
ax.set_ylabel('Eigenfunctions $y_n(k_n r)$')
ax.legend()
plt.show()

```

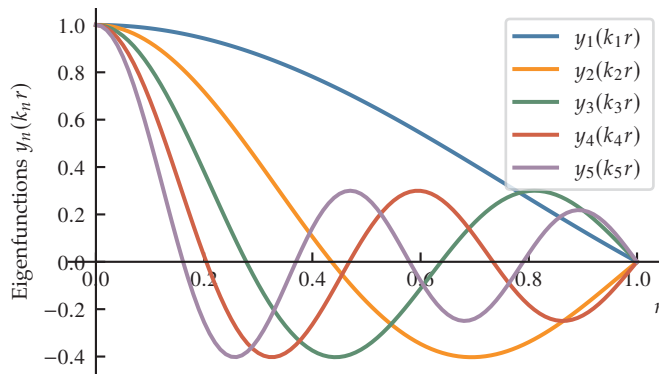


Figure 8.2. Eigenfunctions for the first few eigenvalues.

The plot shows the eigenfunctions  $y_n(k_n r)$  for the first few eigenvalues. Note that the boundary conditions are satisfied and that the eigenfunctions appear to be radial modes of vibration for a circular membrane.

### 8.3 PDE Solution by Separation of Variables



We are now ready to learn one of the most important techniques for solving PDEs: **separation of variables**. It applies only to **linear** PDEs since it will require the principle of superposition. Not all linear PDEs yield to this solution technique, but several that are important do.

The technique includes the following steps.

**assume a product solution** Assume the solution can be written as a **product solution**  $u_p$ : the product of functions of each independent variable.

**separate PDE** Substitute  $u_p$  into the PDE and rearrange such that at least one side of the equation has functions of a single independent variable. If this is possible, the PDE is called **separable**.

**set equal to a constant** Each side of the equation depends on different independent variables; therefore, they must each equal the same constant, often called  $-\lambda$ .

**repeat separation, as needed** If there are more than two independent variables, there will be an ODE in the separated variable and a PDE (with one fewer variables) in the other independent variables. Attempt to separate the PDE until only ODEs remain.

**solve each boundary value problem** Solve each boundary value problem ODE, ignoring the initial conditions for now.

**solve the time variable ODE** Solve for the general solution of the time variable ODE, sans initial conditions.

**construct the product solution** Multiply the solution in each variable to construct the product solution  $u_p$ . If the boundary value problems were sturm-liouville, the product solution is a family of **eigenfunctions** from which any function can be constructed via a generalized fourier series.

**apply the initial condition** The product solutions individually usually do not meet the initial condition. However, a generalized fourier series of them nearly always does. **Superposition** tells us a linear combination of solutions to the PDE and boundary conditions is also a solution; the unique series that also satisfies the initial condition is the unique solution to the entire problem.

#### Example 8.3

Consider the one-dimensional diffusion equation PDE<sup>a</sup>

$$\partial_t u(t, x) = k \partial_{xx}^2 u(t, x)$$

with real constant  $k$ , with dirichlet boundary conditions on interval  $x \in [0, L]$

$$u(t, 0) = 0 \quad (8.11)$$

$$u(t, L) = 0, \quad (8.12)$$

and with initial condition

$$u(0, x) = f(x),$$

where  $f$  is some piecewise continuous function on  $[0, L]$ .

*a.* For more on the diffusion or heat equation, see (Haberman 2018; § 2.3), (Kreyszig 2011; § 12.5), and (Strauss 2007; § 2.3).

**Assume a Product Solution** First, we assume a product solution of the form  $u_p(t, x) = T(t)X(x)$  where  $T$  and  $X$  are unknown functions on  $t > 0$  and  $x \in [0, L]$ .

**Separate PDE** Second, we substitute the product solution into section 8.3 and separate variables:

$$T'X = kTX'' \Rightarrow$$

$$\frac{T'}{kT} = \frac{X''}{X}.$$

So it is separable! Note that we chose to group  $k$  with  $T$ , which was arbitrary but conventional.

**Set Equal to a Constant** Since these two sides depend on different independent variables ( $t$  and  $x$ ), they must equal the same constant we call  $-\lambda$ , so we have two ODEs:

$$\frac{T'}{kT} = -\lambda \Rightarrow T' + \lambda kT = 0$$

$$\frac{X''}{X} = -\lambda \Rightarrow X'' + \lambda X = 0.$$

**Solve the Boundary Value Problem** The latter of these equations with the boundary conditions equation (8.11) is precisely the same sturm-liouville boundary value problem from (**ex:sturm\_liouville1**), which had eigenfunctions

$$X_n(x) = \sin\left(\sqrt{\lambda_n}x\right) \quad (8.13)$$

$$= \sin\left(\frac{n\pi}{L}x\right) \quad (8.14)$$

with corresponding (positive) eigenvalues

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2.$$

**Solve the Time Variable ODE** The time variable ODE is homogeneous and has the familiar general solution

$$T(t) = ce^{-k\lambda t}$$



with real constant  $c$ . However, the boundary value problem restricted values of  $\lambda$  to  $\lambda_n$ , so

$$T_n(t) = ce^{-k(n\pi/L)^2 t}.$$

**Construct the Product Solution** The product solution is

$$\begin{aligned} u_p(t, x) &= T_n(t)X_n(x) \\ &= ce^{-k(n\pi/L)^2 t} \sin\left(\frac{n\pi}{L}x\right). \end{aligned}$$

This is a family of solutions that each satisfy only exotically specific initial conditions.

**Apply the Initial Condition** The initial condition is  $u(0, x) = f(x)$ . The eigenfunctions of the boundary value problem form a fourier series that satisfies the initial condition on the interval  $[0, L]$  if we extend  $f$  to be periodic and odd over  $x$  (Kreyszig 2011; p. 550); we call the extension  $f^*$ . The odd series synthesis can be written

$$f^*(x) = \sum_{n=1}^{\infty} b_n \sin\left(\frac{n\pi}{L}x\right)$$

where the fourier analysis gives

$$b_n = \frac{2}{L} \int_0^L f^*(\chi) \sin\left(\frac{n\pi}{L}\chi\right) d\chi.$$

So the complete solution is

$$u(t, x) = \sum_{n=1}^{\infty} b_n e^{-k(n\pi/L)^2 t} \sin\left(\frac{n\pi}{L}x\right).$$

Notice this satisfies the PDE, the boundary conditions, and the initial condition!

**Plotting Solutions** If we want to plot solutions, we need to specify an initial condition  $u(0, x) = f^*(x)$  over  $[0, L]$ . We can choose anything piecewise continuous, but for simplicity let's let

$$f(x) = 1. \quad (x \in [0, L])$$

The odd periodic extension is an odd square wave. The integral section 8.3 gives

$$\begin{aligned} b_n &= \frac{4}{n\pi} (1 - \cos(n\pi)) \\ &= \begin{cases} 0 & n \text{ even} \\ \frac{4}{n\pi} & n \text{ odd.} \end{cases} \end{aligned}$$

Now we can write the solution as

$$u(t, x) = \sum_{n=1, n \text{ odd}}^{\infty} \frac{4}{n\pi} e^{-k(n\pi/L)^2 t} \sin\left(\frac{n\pi}{L}x\right).$$

**Plotting in Python** First, load some Python packages.

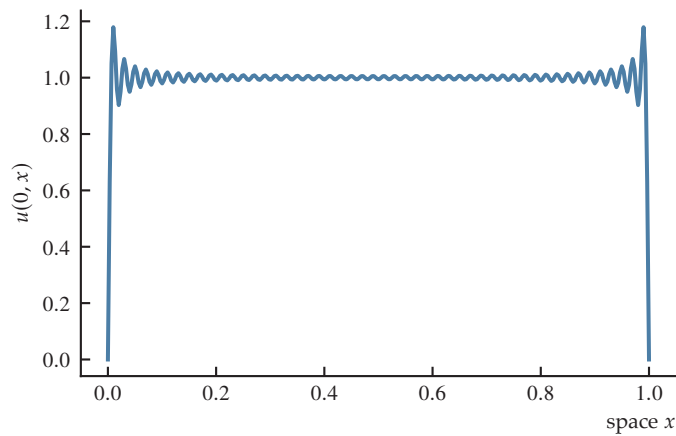
```
import numpy as np
import matplotlib.pyplot as plt
```

Set  $k = L = 1$  and sum values for the first  $N$  terms of the solution.

```
L = 1
k = 1
N = 100
x = np.linspace(0, L, 300)
t = np.linspace(0, 2*(L/np.pi)**2, 100)
u_n = np.zeros([len(t), len(x)])
for n in range(N):
    n = n+1 # because index starts at 0
    if n % 2 == 0: # even
        pass # already initialized to zeros
    else: # odd
        u_n += 4/(n*np.pi)*np.outer(
            np.exp(-k*(n*np.pi/L)**2*t),
            np.sin(n*np.pi/L*x)
        )
```

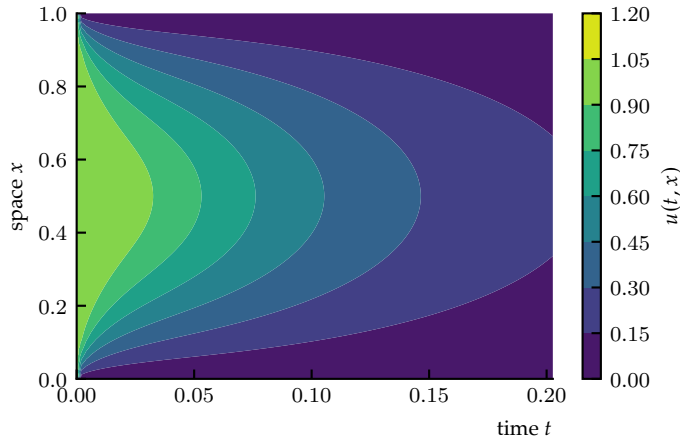
Let's first plot the initial condition.

```
fig, ax = plt.subplots()
ax.plot(x, u_n[0, :])
plt.xlabel('space $x$')
plt.ylabel('$u(0, x)$')
plt.draw()
```

Figure 8.3. Initial condition  $u(0, x)$ 

Now we plot the entire response.

```
fig, ax = plt.subplots()
plt.contourf(t,x,u_n.T)
c = plt.colorbar()
c.set_label('$u(t,x)$')
plt.xlabel('time $t$')
plt.ylabel('space $x$')
plt.show()
```

Figure 8.4. Solution  $u(t, x)$ 

We see the diffusive action proceeds as we expected.

## 8.4 The 1D Wave Equation

The one-dimensional **wave equation** is the linear PDE

$$\partial_{tt}^2 u(t, x) = c^2 \partial_{xx}^2 u(t, x).$$

with real constant  $c$ . This equation models such phenomena as strings, fluids, sound, and light. It is subject to initial and boundary conditions and can be extended to multiple spatial dimensions. For 2D and 3D examples in rectangular and polar coordinates, see (Kreyszig 2011; § 12.9 12.10) and (Haberman 2018; § 4.5 7.3).

### Example 8.4

Consider the one-dimensional wave equation PDE

$$\partial_{tt}^2 u(t, x) = c^2 \partial_{xx}^2 u(t, x) \quad (8.15)$$

with real constant  $c$  and with dirichlet boundary conditions on interval  $x \in [0, L]$

$$u(t, 0) = 0 \quad \text{and} \quad u(t, L) = 0, \quad (8.16)$$

and with initial conditions (we need two because of the second time-derivative)

$$u(0, x) = f(x) \quad \text{and} \quad \partial_t u(0, x) = g(x),$$

where  $f$  and  $g$  are some piecewise continuous functions on  $[0, L]$ .



**Assume a Product Solution** First, we assume a product solution of the form  $u_p(t, x) = T(t)X(x)$  where  $T$  and  $X$  are unknown functions on  $t > 0$  and  $x \in [0, L]$ .

**Separate PDE** Second, we substitute the product solution into equation (8.15) and separate variables:

$$T''X = c^2TX'' \Rightarrow$$

$$\frac{T''}{c^2T} = \frac{X''}{X}.$$

So it is separable! Note that we chose to group  $c$  with  $T$ , which was arbitrary but conventional.

**Set Equal to a Constant** Since these two sides depend on different independent variables ( $t$  and  $x$ ), they must equal the same constant we call  $-\lambda$ , so we have two ODEs:

$$\frac{T''}{c^2T} = -\lambda \Rightarrow T'' + \lambda c^2T = 0$$

$$\frac{X''}{X} = -\lambda \Rightarrow X'' + \lambda X = 0.$$

**Solve the Boundary Value Problem** The latter of these equations with the boundary conditions ?? is precisely the same Sturm-Liouville boundary value problem from ??, which had eigenfunctions

$$X_n(x) = \sin\left(\sqrt{\lambda_n}x\right) \quad (8.17)$$

$$= \sin\left(\frac{n\pi}{L}x\right) \quad (8.18)$$

with corresponding (positive) eigenvalues

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2.$$

**Solve the Time Variable ODE** The time variable ODE is homogeneous and, with  $\lambda$  restricted by the reals by the boundary value problem, has the familiar general solution

$$T(t) = k_1 \cos(c\sqrt{\lambda}t) + k_2 \sin(c\sqrt{\lambda}t)$$

with real constants  $k_1$  and  $k_2$ . However, the boundary value problem restricted values of  $\lambda$  to  $\lambda_n$ , so

$$T_n(t) = k_1 \cos\left(\frac{cn\pi}{L}t\right) + k_2 \sin\left(\frac{cn\pi}{L}t\right).$$

**Construct the Product Solution** The product solution is

$$\begin{aligned} u_p(t, x) &= T_n(t)X_n(x) \\ &= k_1 \sin\left(\frac{n\pi}{L}x\right) \cos\left(\frac{cn\pi}{L}t\right) + k_2 \sin\left(\frac{n\pi}{L}x\right) \sin\left(\frac{cn\pi}{L}t\right). \end{aligned}$$

This is a family of solutions that each satisfy only exotically specific initial conditions.

**Apply the Initial Conditions** Recall that superposition tells us that any linear combination of the product solution is also a solution. Therefore,

$$u(t, x) = \sum_{n=1}^{\infty} a_n \sin\left(\frac{n\pi}{L}x\right) \cos\left(\frac{cn\pi}{L}t\right) + b_n \sin\left(\frac{n\pi}{L}x\right) \sin\left(\frac{cn\pi}{L}t\right)$$

is a solution. If  $a_n$  and  $b_n$  are properly selected to satisfy the initial conditions, section 8.4 will be the solution to the entire problem. Substituting  $t = 0$  into our potential solution gives

$$u(0, x) = \sum_{n=1}^{\infty} a_n \sin\left(\frac{n\pi}{L}x\right) \quad (8.19)$$

$$\partial_t u(t, x)|_{t=0} = \sum_{n=1}^{\infty} b_n \frac{cn\pi}{L} \sin\left(\frac{n\pi}{L}x\right). \quad (8.20)$$

Let us extend  $f$  and  $g$  to be periodic and odd over  $x$ ; we call the extensions  $f^*$  and  $g^*$ . From equation (8.19), the initial conditions are satisfied if

$$f^*(x) = \sum_{n=1}^{\infty} a_n \sin\left(\frac{n\pi}{L}x\right) \quad (8.21)$$

$$g^*(x) = \sum_{n=1}^{\infty} b_n \frac{cn\pi}{L} \sin\left(\frac{n\pi}{L}x\right). \quad (8.22)$$

We identify these as two odd fourier syntheses. The corresponding fourier analyses are

$$a_n = \frac{2}{L} \int_0^L f^*(x) \sin\left(\frac{n\pi}{L}x\right) \quad (8.23)$$

$$b_n \frac{cn\pi}{L} = \frac{2}{L} \int_0^L g^*(x) \sin\left(\frac{n\pi}{L}x\right) \quad (8.24)$$

So the complete solution is equations (8.21) and (8.22) with components given by equations (8.23) and (8.24). Notice this satisfies the PDE, the boundary conditions, and the initial condition!

**Discussion** It can be shown that this series solution is equivalent to two *traveling waves* that are interfering (see (Haberman 2018; § 4.4) and (Kreyszig 2011; § 12.2)). This is convenient because computing the series solution exactly requires an infinite summation. We show in the following section that the approximation by partial summation is still quite good.

**Choosing Specific Initial Conditions** If we want to plot solutions, we need to specify initial conditions over  $[0, L]$ . Let's model a string being suddenly struck from rest as

$$\begin{aligned} f(x) &= 0 \\ g(x) &= \delta(x - \Delta L) \end{aligned}$$

where  $\delta$  is the dirac delta distribution and  $\Delta \in [0, L]$  is a fraction of  $L$  representing the location of the string being struck. The odd periodic extension is an odd pulse train. The integrals of equations (8.23) and (8.24) give

$$a_n = 0 \tag{8.25}$$

$$\begin{aligned} b_n &= \frac{2}{cn\pi} \int_0^L \delta(x - \Delta L) \sin\left(\frac{n\pi}{L}x\right) dx \\ &= \frac{2}{cn\pi} \sin(n\pi\Delta). \end{aligned} \tag{sifting property}$$

Now we can write the solution as

$$u(t, x) = \sum_{n=1}^{\infty} \frac{2}{cn\pi} \sin(n\pi\Delta) \sin\left(\frac{n\pi}{L}x\right) \sin\left(\frac{cn\pi}{L}t\right).$$

**Plotting in Python** First, load some Python packages.

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

Set  $c = L = 1$  and sum values for the first  $N$  terms of the solution for some striking location  $\Delta$ .

```

Delta = 0.1 # 0 <= Delta <= L
L = 1
c = 1
N = 150
t = np.linspace(0,30*(L/np.pi)**2,100)
x = np.linspace(0,L,150)
t_b, x_b = np.meshgrid(t,x)
u_n = np.zeros([len(x),len(t)])
for n in range(N):
    n = n+1 # because index starts at 0
    u_n += 4/(c*n*np.pi)* \
        np.sin(n*np.pi*Delta)* \
        np.sin(c*n*np.pi/L*t_b)* \
        np.sin(n*np.pi/L*x_b)

```

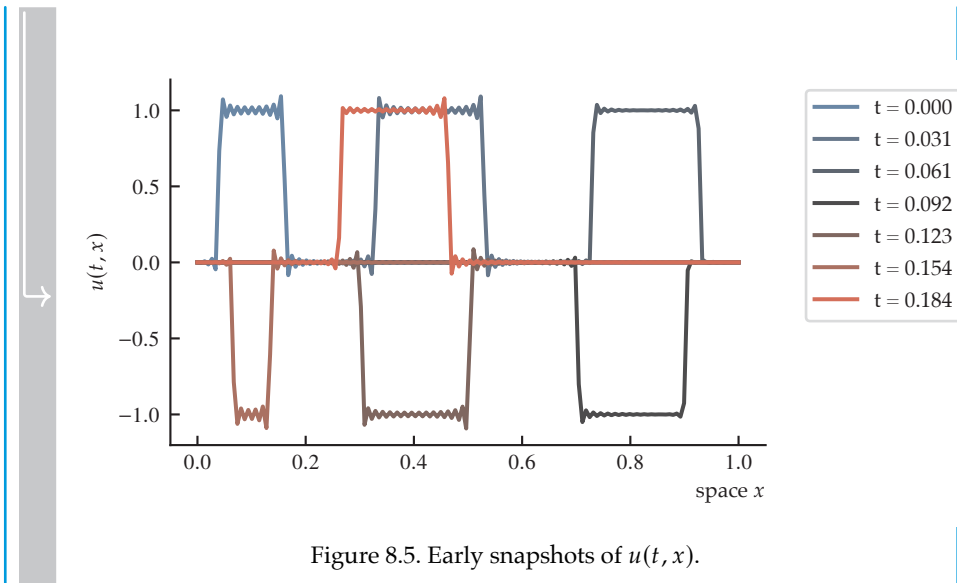
Let's first plot some early snapshots of the response.

```

import seaborn as sns
n_snaps = 7
sns.set_palette(
    sns.diverging_palette(
        240, 10, n=n_snaps, center="dark"
    )
)
fig, ax = plt.subplots()
it = np.linspace(2,77,n_snaps,dtype=int)
for i in range(len(it)):
    ax.plot(x,u_n[:,it[i]],label=f"t = {t[i]:.3f}");
lgd = ax.legend(
    bbox_to_anchor=(1.05, 1),
    loc='upper left'
)
plt.xlabel('space $x$')
plt.ylabel('$u(t,x)$')
plt.draw()

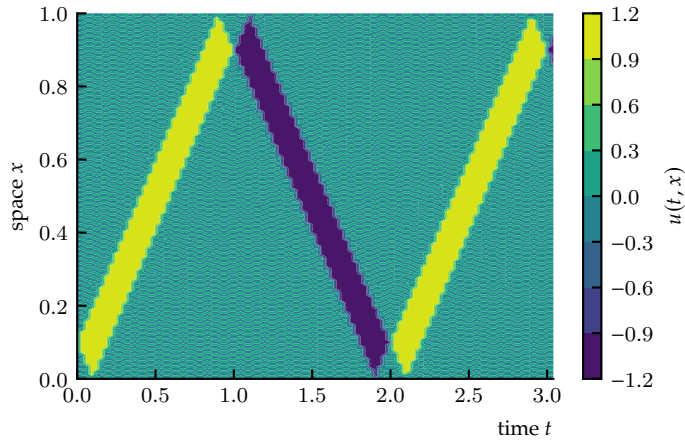
```



Figure 8.5. Early snapshots of  $u(t, x)$ .

Now we plot the entire response.

```
fig, ax = plt.subplots()
p = ax.contourf(t, x, u_n)
c = fig.colorbar(p, ax=ax)
c.set_label('$u(t,x)$')
plt.xlabel('time $t$')
plt.ylabel('space $x$')
plt.show()
```

Figure 8.6. Solution  $u(t, x)$ .

We see a wave develop and travel, reflecting and inverting off each boundary.

### 8.5 The Laplacian in Polar Coordinates and the Fourier-Bessel Series

The **Laplacian** is a differential operator often given the symbol  $\nabla^2$  or  $\Delta$ , equivalent to the divergence of the gradient. In Cartesian coordinates, the Laplacian is given by

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$$

where  $u$  is a scalar function of the coordinates  $x$ ,  $y$ , and  $z$ . In cylindrical coordinates, the Laplacian is given by

$$\nabla^2 u = \frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial u}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} + \frac{\partial^2 u}{\partial z^2}$$

where  $u$  is a scalar function of the coordinates  $r$ ,  $\theta$ , and  $z$ .

The Laplacian models many physical phenomena, such as heat conduction, mechanical vibration, and electrostatics. In this section, we will use the Laplacian to model the vibration of a circular membrane, such as a drumhead. The Laplacian can be shown to be an effective model for membranes that are thin compared to their radius and that do not resist bending (Kreyszig 2011; § 12.10). For a circular

membrane, the two-dimensional wave equation in polar coordinates is

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u = c^2 \left( \frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial u}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 u}{\partial \theta^2} \right)$$

where  $u(r, \theta, t)$  is the displacement of the membrane at radius  $r$ , angle  $\theta$ , and time  $t$ , and  $c$  is the wave speed.

Following Kreyszig (2011; § 12.10), we will explore solutions with radial symmetry, where the displacement  $u$  depends only on the radius  $r$  and time  $t$ . In this case, the wave equation simplifies to

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial u}{\partial r} \right).$$

Expanding the derivatives gives

$$\frac{\partial^2 u}{\partial t^2} = c^2 \left( \frac{\partial^2 u}{\partial r^2} + \frac{1}{r} \frac{\partial u}{\partial r} \right). \quad (8.26)$$

### Example 8.5

Consider the radially symmetric wave equation in polar coordinates, given by equation (8.26), for a circular membrane with radius  $R = 1$  and wave speed  $c = 2\pi$  rad/s. The displacement of the membrane is  $u(r, t)$ , where  $r$  is the radius and  $t$  is time. The boundary conditions are

- At the edge of the membrane,  $r = R$ , the membrane is fixed, so  $u = 0$ .
- At the center of the membrane,  $r = 0$ , the membrane is free to move, but the displacement is finite, so  $|u| < \infty$ .

The initial conditions are

- The membrane is at rest at  $t = 0$ , so  $\partial u / \partial t = 0$ .
- The membrane is displaced at  $t = 0$ , so  $u = f(r)$ , where  $f(r) = 1 - (r/R)^3$ .

Develop a series solution for the displacement of the membrane,  $u(r, t)$ .

We will assume a product solution of the form  $u_p(r, t) = W(r)G(t)$ . Substituting this into the wave equation equation (8.26) gives

$$\frac{\partial^2}{\partial t^2}(WG) = c^2 \left( \frac{\partial^2}{\partial r^2}(WG) + \frac{1}{r} \frac{\partial}{\partial r}(WG) \right) \implies \quad (8.27)$$

$$W \frac{\partial^2 G}{\partial t^2} = c^2 \left( G \frac{\partial^2 W}{\partial r^2} + \frac{1}{r} G \frac{\partial W}{\partial r} \right). \quad (8.28)$$

Using  $\dot{G} = \partial G / \partial t$  and  $W' = \partial W / \partial r$ , we rewrite this as

$$W\ddot{G} = c^2 \left( GW'' + \frac{1}{r} GW' \right),$$

which can be separated into two ordinary differential equations by dividing by  $c^2 WG$ :

$$\frac{\ddot{G}}{c^2 G} = \frac{W''}{W} + \frac{1}{r} \frac{W'}{W}.$$

The left-hand side depends only on  $t$ , and the right-hand side depends only on  $r$ . Therefore, both sides must be equal to a constant, which we will call  $-k^2$ . This gives two ordinary differential equations, one for time and one for radius. The time equation is, letting  $\lambda \equiv ck$ ,

$$\ddot{G} + \lambda^2 G = 0,$$

which is the familiar harmonic oscillator equation with solutions  $G(t) = A \cos(\lambda t) + B \sin(\lambda t)$ .

The radial equation is

$$W'' + \frac{1}{r} W' + k^2 W = 0,$$

which can be transformed into the Bessel equation (see example 8.2) by changing variables to  $s = kr$ . Apply the chain rule to find

$$W' = \frac{dW}{dr} = \frac{dW}{ds} \frac{ds}{dr} = k \frac{dW}{ds},$$

and therefore  $W'' = k^2 d^2 W / ds^2$ . Substitute these into the radial equation to get

$$k^2 \frac{d^2 W}{ds^2} + \frac{1}{r} k \frac{dW}{ds} + k^2 W = 0 \implies \frac{d^2 W}{ds^2} + \frac{1}{s} \frac{dW}{ds} + W = 0,$$

which is the Bessel equation with  $\nu = 0$ . From example 8.2, we know that the general solution to the Bessel equation is

$$W(s) = aJ_0(s) + bY_0(s),$$

where  $J_0(s)$  and  $Y_0(s)$  are order-0 Bessel functions of the first and second kind, respectively. The boundary condition  $|u| < \infty$  at  $r = 0$  requires  $b = 0$ , so the general solution for  $W(s)$  is

$$W(s) = aJ_0(s).$$

The boundary condition  $u = 0$  at  $r = R$  requires  $J_0(kR) = 0$ . Zeros of the Bessel function  $J_0(s)$  are denoted by  $\alpha_{0,m}$ , so the values of  $k$  are

$$k_m = \frac{\alpha_{0,m}}{R}$$

and the eigenfunctions are

$$u_m(r, t) = (A_m \cos(\lambda_m t) + B_m \sin(\lambda_m t)) J_0(k_m r),$$

where the eigenvalues are  $\lambda_m = ck_m$ .

Applying the initial condition  $u'(r, 0) = 0$  gives

$$\left. \frac{\partial u}{\partial t} \right|_{t=0} = -A_m \lambda_m \sin(\lambda_m 0) + B_m \lambda_m \cos(\lambda_m 0) J_0(k_m r) \implies \quad (8.29)$$

$$0 = B_m \lambda_m J_0(k_m r) \implies \quad (8.30)$$

$$B_m = 0. \quad (8.31)$$

Applying the initial condition  $u(r, 0) = f(r)$  gives

$$u(r, 0) = A_m \cos(\lambda_m 0) J_0(k_m r) \implies \quad (8.32)$$

$$= A_m J_0(k_m r). \quad (8.33)$$

We can choose  $A_m$  such that  $u(r, 0) = f(r)$  in the rare cases where  $f(r)$  is a Bessel function. Instead, we will use the orthogonality of the Bessel functions to find a series representation of  $f(r)$  in terms of the eigenfunctions  $u_m(r, t)$ ; that is, we will find  $A_m$  such that

$$f(r) = \sum_{m=1}^{\infty} A_m J_0(k_m r).$$

So  $A_m$  are the Fourier-Bessel coefficients of  $f(r)$ ,

$$A_m = \frac{\langle f, f_m \rangle_{w=r}}{\langle f_m, f_m \rangle_{w=r}},$$

with weight function  $w = r$ , interval  $[0, R]$ , and  $f_m(r) = J_0(k_m r)$ . The inner products are given from the definition of the inner product,

$$\langle f, f_m \rangle_{w=r} = \int_0^R f(r) f_m(r) r \, dr, \quad (8.34)$$

$$\langle f_m, f_m \rangle_{w=r} = \int_0^R f_m(r)^2 r \, dr. \quad (8.35)$$

We proceed in Python. First, load packages.

```
import numpy as np
import sympy as sp
import scipy
from scipy.special import jn_zeros
import matplotlib.pyplot as plt
```

The inner products are computed as follows:

```

r = sp.symbols("r", real=True, nonnegative=True)
k_m, R = sp.symbols("k_m, R", real=True, positive=True)
f = 1 - (r/R)**3 # Initial condition
f_m = sp.besselj(0, k_m * r) # Eigenfunction
w = r # Weight function of the inner product
inner_f_fm = sp.integrate(f * f_m * w, (r, 0, R)) #  $\langle f, f_m \rangle$ 
inner_fm_fm = sp.integrate(f_m**2 * w, (r, 0, R)) #  $\langle f_m, f_m \rangle$ 
A_m = (inner_f_fm / inner_fm_fm).simplify()

```

The values of  $k_m$ , eigenvalues  $\lambda_m = ck_m$ , and the series components  $A_m$  can be computed from the zeros of the Bessel function zeros  $\alpha_{0,m}$ , where  $m = 1, 2, 3, \dots$  as follows:

```

c = sp.symbols("c", real=True, positive=True)
n_zeros = 5 # Number of Bessel function zeros
params = {R: 1, c: 2*np.pi}

```

The partial sum of the series can be defined as the sum of the first  $n$  terms of the series:

```

def f_partial_sum_t(r, t, n, A_m, params):
    alphas = jn_zeros(0, n) # Bessel function zeros
    ks = alphas / params[R]
    lambdas = params[c] * ks
    A_m_ = np.zeros(n) # Numerical vals of the series comp.
    f_partial = np.zeros((len(r), len(t)))
    for i in range(n):
        A_m_[i] = A_m.subs(k_m, ks[i]).subs(params).evalf()
        f_m = scipy.special.jn(0, ks[i] * r[:, None])
        f_partial += A_m_[i] * f_m * np.cos(lambdas[i] * t)
    return f_partial

```

Plot the partial sum of the series and compare it to the initial condition.

```

r_plt = np.linspace(-params[R], params[R], 101)
t_plt = np.array([0])
f_num = 1 - (np.abs(r_plt)/params[R])**3
f_partial = f_partial_sum_t(r_plt, t_plt, n_zeros, A_m, params)
fig, ax = plt.subplots()
ax.plot(r_plt, f_num, label="$f$")
ax.plot(r_plt, f_partial, label="$f_{\mathrm{partial}}$")
ax.set_xlabel('$r$')
ax.legend()
plt.draw()

```

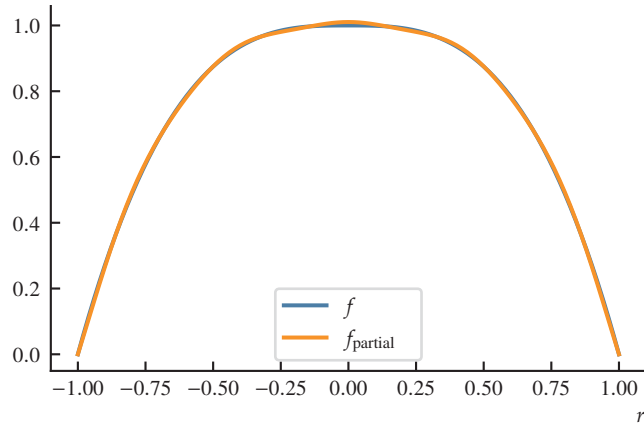


Figure 8.7. Comparison between the initial condition and the partial sum of the series.

The result is quite good, even with only five terms in the series.

We can now plot the partial sum over one period.

```
cmap = plt.get_cmap('viridis')
n_snapshots = 101
t_plt = np.linspace(0, 1, n_snapshots)
r_plt = np.linspace(-params[R], params[R], 201)
f_partial = f_partial_sum_t(r_plt, t_plt, n_zeros, A_m, params)
fig, ax = plt.subplots()
for i in range(n_snapshots):
    # Label 5 snapshots
    label = None
    if i % 20 == 0:
        label = f"$t = {t_plt[i]:.2f}$"
    ax.plot(r_plt, f_partial[:, i], label=label,
            color=cmap(i / n_snapshots), alpha=1)
ax.set_xlabel('$r$')
ax.set_ylabel('Displacement u(r, t)')
ax.legend(loc='upper right')
plt.show()
```

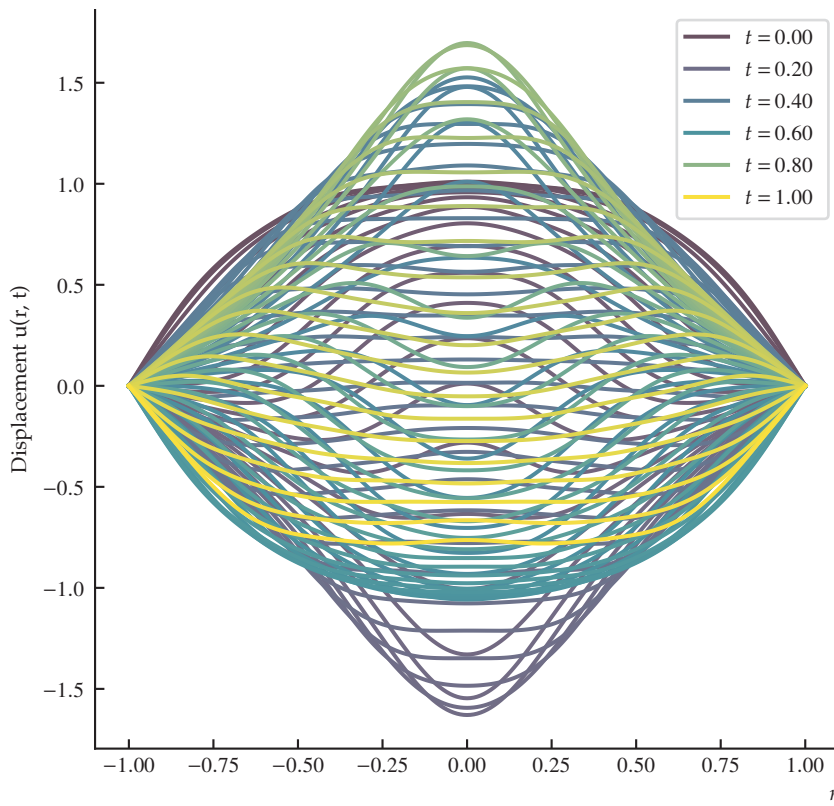



Figure 8.8. Partial sum of the series over one period.



## 8.6 Problems



**Problem 8.1**  **HORTICULTURE** The PDE of example 8.3 can be used to describe the conduction of heat along a long, thin rod, insulated along its length, where  $u(t, x)$  represents temperature. The initial and dirichlet boundary conditions in that example would be interpreted as an initial temperature distribution along the bar and fixed temperatures of the ends. Now consider the same PDE

$$\partial_t u(t, x) = k \partial_{xx}^2 u(t, x) \quad (8.36)$$

with real constant  $k$ , with mixed boundary conditions on interval  $x \in [0, L]$

$$u(t, 0) = 0 \quad (8.37a)$$

$$\partial_x u(t, x)|_{x=L} = 0, \quad (8.37b)$$

and with initial condition

$$u(0, x) = f(x), \quad (8.38)$$

where  $f$  is some piecewise continuous function on  $[0, L]$ . This represents the insulation of one end ( $L$ ) of the rod and the other end ( $0$ ) is held at a fixed temperature.

- Assume a product solution, separate variables into  $X(x)$  and  $T(t)$ , and set the separation constant to  $-\lambda$ .
- Solve the boundary value problem for its eigenfunctions  $X_n$  and eigenvalues  $\lambda_n$ .
- Solve for the general solution of the time variable ODE.
- Write the product solution and apply the initial condition  $f(x)$  by constructing it from a generalized fourier series of the product solution.
- Let  $L = k = 1$  and

$$f(x) = \begin{cases} 0 & \text{for } x \in [0, L/2) \\ 100 & \text{for } x \in [L/2, L] \end{cases} \quad (8.39)$$

as shown in figure 8.9. Compute the solution series components. Plot the sum of the first 50 terms over  $x$  and  $t$ .

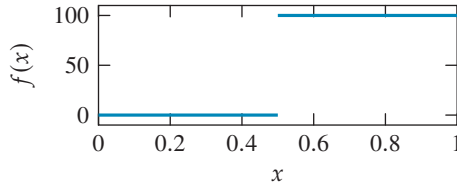



Figure 8.9. Initial condition for problem 8.1.

**Problem 8.2**  **POLTERGEIST** The PDE of example 8.3 can be used to describe the conduction of heat along a long, thin rod, insulated along its length, where  $u(t, x)$  represents temperature. The initial and dirichlet boundary conditions in that example would be interpreted as an initial temperature distribution along the bar and fixed temperatures of the ends. Now consider the same PDE

$$\partial_t u(t, x) = k \partial_{xx}^2 u(t, x) \quad (8.40)$$

with real constant  $k$ , now with *neumann* boundary conditions on interval  $x \in [0, L]$

$$\partial_x u|_{x=0} = 0 \quad \text{and} \quad \partial_x u|_{x=L} = 0, \quad (8.41a)$$

and with initial condition

$$u(0, x) = f(x), \quad (8.42)$$

where  $f$  is some piecewise continuous function on  $[0, L]$ . This represents the complete insulation of the ends of the rod, such that no heat flows from the ends (or from anywhere else).

- Assume a product solution, separate variables into  $X(x)$  and  $T(t)$ , and set the separation constant to  $-\lambda$ .
- Solve the boundary value problem for its eigenfunctions  $X_n$  and eigenvalues  $\lambda_n$ .
- Solve for the general solution of the time variable ODE.
- Write the product solution and apply the initial condition  $f(x)$  by constructing it from a generalized fourier series of the product solution.
- Let  $L = k = 1$  and  $f(x) = 100 - 200/L |x - L/2|$  as shown in figure 8.10. Compute the solution series components. Plot the sum of the first 50 terms over  $x$  and  $t$ .

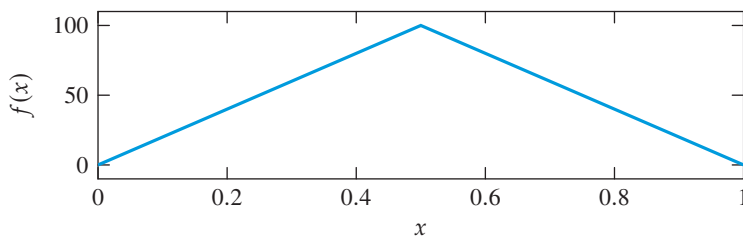



Figure 8.10. Initial condition for problem 8.2.

**Problem 8.3**  **KATHMANDU** Consider the free vibration of a uniform and relatively thin beam—with modulus of elasticity  $E$ , second moment of cross-sectional area  $I$ , and mass-per-length  $\mu$ —pinned at each end. The PDE describing this is a version of the euler-bernoulli beam equation for vertical motion  $u$ :

$$\partial_{tt}^2 u(t, x) = -\alpha^2 \partial_{xxxx}^4 u(t, x) \quad (8.43)$$

with real constant  $\alpha$  defined as

$$\alpha^2 = \frac{EI}{\mu}. \quad (8.44)$$

Pinned supports fix vertical motion such that we have boundary conditions on interval  $x \in [0, L]$

$$u(t, 0) = 0 \quad \text{and} \quad u(t, L) = 0. \quad (8.45a)$$

Additionally, pinned supports cannot provide a moment, so

$$\partial_{xx}^2 u|_{x=0} = 0 \quad \text{and} \quad \partial_{xx}^2 u|_{x=L} = 0. \quad (8.45b)$$

Furthermore, consider the initial conditions

$$u(0, x) = f(x), \quad \text{and} \quad \partial_t u|_{t=0} = 0. \quad (8.46a)$$

where  $f$  is some piecewise continuous function on  $[0, L]$ .

- Assume a product solution, separate variables into  $X(x)$  and  $T(t)$ , and set the separation constant to  $-\lambda$ .
- Solve the boundary value problem for its eigenfunctions  $X_n$  and eigenvalues  $\lambda_n$ . Assume real  $\lambda > 0$  (it's true but tedious to show).
- Solve for the general solution of the time variable ODE.
- Write the product solution and apply the initial conditions by constructing it from a generalized fourier series of the product solution.
- Let  $L = \alpha = 1$  and  $f(x) = \sin(10\pi x/L)$  as shown in figure 8.11. Compute the solution series components. Plot the sum of the first 50 terms over  $x$  and  $t$ .

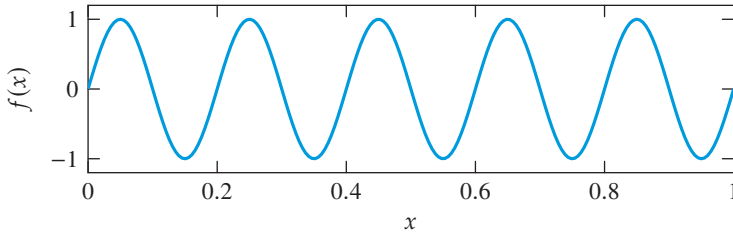


Figure 8.11. Initial condition for problem 8.3.

**Problem 8.4**  Given the 1D heat equation,

$$\frac{\partial}{\partial t} u(t, x) = \alpha \frac{\partial^2}{\partial x^2} u(t, x),$$

with boundary conditions,

$$\frac{\partial}{\partial x} u(t, x) \Big|_{x=L} = 0$$

$$u(t, 0) = 0,$$

and initial condition,

$$u(0, x) = \begin{cases} 1 & \frac{L}{3} \leq x \leq \frac{2L}{3} \\ 0 & \text{otherwise} \end{cases}$$

- show that this PDE is separable,
- solve the sturm-liouville boundary condition problem,
- find the fourier coefficients, and
- given  $L = 1$ ,  $\alpha = 1$ , and using the first 100 terms of the infinite sum, plot the solution at  $t = 0$ ,  $t = 0.01$ , and  $t = 0.1$ .

**Problem 8.5**  Consider the one-dimensional wave equation PDE

$$\partial_{tt}^2 u(t, x) = c^2 \partial_{xx}^2 u(t, x) \quad (8.47)$$

with real constant  $c$  and with dirichlet boundary conditions on interval  $x \in [0, L]$

$$u(t, 0) = 0 \quad \text{and} \quad u(t, L) = 0, \quad (8.48a)$$

and with initial conditions (we need two because of the second time-derivative)

$$u(0, x) = f(x) \quad \text{and} \quad \partial_t u(0, x) = g(x), \quad (8.49)$$

where  $f$  and  $g$  are some piecewise continuous functions on  $[0, L]$ .

Assume we can model a musical instrument's plucked string with equations (8.47) to (8.49) with the initial velocity  $g(x) = 0$  and initial displacement  $f(x)$  given in figure 8.12.

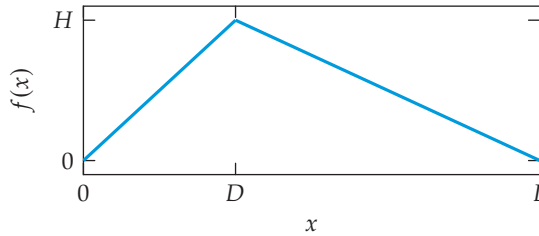


Figure 8.12. Initial condition for problem 8.5.

- a. Assume a product solution, separate variables into  $X(x)$  and  $T(t)$ , and set the separation constant to  $-\lambda$ .
- b. Solve the boundary value problem for its eigenfunctions  $X_n$  and eigenvalues  $\lambda_n$ .
- c. Solve for the general solution of the time variable ODE.
- d. Write the product solution and apply the initial conditions by constructing it from a generalized fourier series of the product solution.
- e. Let  $H = 0.5$ ,  $L = c = 1$ , and  $D = 0.3$ . Compute the solution series components. Plot the sum of the first 50 terms over  $x$  and  $t$ .



# 9 Optimization



This chapter concerns optimization mathematics. Optimization is concerned with finding one or more best solution, called an **optimal solution**, to a problem. Optimal solutions are defined with respect to a given criterion, called an **objective function**. The objective function takes a set of input values and returns a single output value that represents the quality of the solution. The goal of optimization is to find one or more input values to an objective function that produce optimal output values, either by maximizing or minimizing the output value (i.e., resulting in an **extremum**). The set of argument values that produce the optimal output value are called the set of **optimal solutions**.

The notation for extrema is as follows:

- The minimum of an objective function  $f$  is denoted by  $\min f$ . For argument  $x \in S$ , the minimum value is

$$\min_{x \in S} f(x).$$

- The maximum of an objective function  $f$  is denoted by  $\max f$ . For argument  $x \in S$ , the maximum value is

$$\max_{x \in S} f(x).$$

- The set of solutions that minimize an objective function  $f$  is denoted by  $\operatorname{argmin} f$ . For argument  $x \in S$ , the set of solutions is

$$\operatorname{argmin}_{x \in S} f(x).$$

- The set of solutions that maximize an objective function  $f$  is denoted by  $\operatorname{argmax} f$ . For argument  $x \in S$ , the set of solutions is

$$\operatorname{argmax}_{x \in S} f(x).$$

Optimization problems can be classified into two categories: **unconstrained optimization** and **constrained optimization**. In unconstrained optimization, the objective function is optimized over the entire domain of the input space, whereas

in constrained optimization, the objective function is optimized over a subset of the input space that satisfies a set of constraints. In this chapter, we will consider both unconstrained and constrained optimization problems.

## 9.1 Gradient Descent



Consider a multivariate function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  that represents some cost or value. This is called an **objective function**, and we often want to find an  $\mathbf{X} \in \mathbb{R}^n$  that yields  $f$ 's **extremum**: minimum or maximum, depending on whichever is desirable.

It is important to note however that some functions have no finite extremum. Other functions have multiple. Finding a **global extremum** is generally difficult; however, many good methods exist for finding a **local extremum**: an extremum for some region  $R \subset \mathbb{R}^n$ .

The method explored here is called **gradient descent**. It will soon become apparent why it has this name.

### 9.1.1 Stationary Points

Recall from basic calculus that a function  $f$  of a single variable had potential local extrema where  $df(x)/dx = 0$ . The multivariate version of this, for multivariate function  $f$ , is

$$\text{grad } f = \mathbf{0}. \quad (9.1)$$

A value  $\mathbf{X}$  for which equation (9.1) holds is called a **stationary point**. However, as in the univariate case, a stationary point may not be a local extremum; in these cases, it called a **saddle point**.

Consider the **hessian matrix**  $H$  with values, for independent variables  $x_i$ ,

$$H_{ij} = \partial_{x_i x_j}^2 f.$$

For a stationary point  $\mathbf{X}$ , the **second partial derivative test** tells us if it is a local maximum, local minimum, or saddle point:

**minimum** If  $H(\mathbf{X})$  is **positive definite** (all its eigenvalues are positive),

$\mathbf{X}$  is a local minimum.

**maximum** If  $H(\mathbf{X})$  is **negative definite** (all its eigenvalues are negative),

$\mathbf{X}$  is a local maximum.

**saddle** If  $H(\mathbf{X})$  is **indefinite** (it has both positive and negative eigenvalues),

$\mathbf{X}$  is a saddle point.

These are sometimes called tests for concavity: minima occur where  $f$  is **convex** and maxima where  $f$  is **concave** (i.e. where  $-f$  is convex).



It turns out, however, that solving equation (9.1) directly for stationary points is generally hard. Therefore, we will typically use an iterative technique for estimating them.

### 9.1.2 The Gradient Points the Way

Although equation (9.1) isn't usually directly useful for computing stationary points, it suggests iterative techniques that are. Several techniques rely on the insight that **the gradient points toward stationary points**. Recall from section 5.3 that  $\text{grad } f$  is a vector field that points in the direction of greatest increase in  $f$ .

Consider starting at some point  $x_0$  and wanting to move iteratively closer to a stationary point. So, if one is seeking a maximum of  $f$ , then choose  $x_1$  to be in the direction of  $\text{grad } f$ . If one is seeking a minimum of  $f$ , then choose  $x_1$  to be opposite the direction of  $\text{grad } f$ .

The question becomes: *how far*  $\alpha$  should we go in (or opposite) the direction of the gradient? Surely too-small  $\alpha$  will require more iteration and too-large  $\alpha$  will lead to poor convergence or missing minima altogether. This framing of the problem is called **line search**. There are a few common methods for choosing  $\alpha$ , called the **step size**, some more computationally efficient than others.

Two methods for choosing the step size are described below. Both are framed as minimization methods, but changing the sign of the step turns them into maximization methods.

### 9.1.3 The Classical Method

Let

$$g_k = \text{grad } f(x_k),$$

the gradient at the algorithm's current estimate  $x_k$  of the minimum. The classical method of choosing  $\alpha$  is to attempt to solve analytically for

$$\alpha_k = \underset{\alpha}{\operatorname{argmin}} f(x_k - \alpha g_k). \quad (9.2)$$

This solution approximates the function  $f$  as one varies  $\alpha$ . It is approximate because as  $\alpha$  varies, so should  $x$ . But even with  $\alpha$  as the only variable, equation (9.2) may be difficult or impossible to solve. However, this is sometimes called the "optimal" choice for  $\alpha$ . Here "optimality" refers not to practicality but to ideality. This method is rarely used to solve practical problems.

The algorithm of the classical gradient descent method can be summarized in the pseudocode of algorithm 1. It is described further in [kreyszig2011]{plaincite post="§ 22.1"}.

**Algorithm 1** Classical gradient descent

---

```

1: procedure classical_minimizer( $f, x_0, T$ )
2:   while  $\delta x > T$  do                                      $\triangleright$  until threshold  $T$  is met
3:      $g_k \leftarrow \text{grad } f(x_k)$ 
4:      $\alpha_k \leftarrow \text{argmin}_{\alpha} f(x_k - \alpha g_k)$ 
5:      $x_{k+1} \leftarrow x_k - \alpha_k g_k$ 
6:      $\delta x \leftarrow \|x_{k+1} - x_k\|$ 
7:      $k \leftarrow k + 1$ 
8:   return  $x_k$                                               $\triangleright$  the threshold was reached

```

---

**9.1.4 The Barzilai and Borwein Method**

In practice, several non-classical methods are used for choosing step size  $\alpha$ . Most of these construct criteria for step sizes that are too small and too large and prescribe choosing some  $\alpha$  that (at least in certain cases) must be in the sweet-spot in between. Barzilai and Borwein (1988) developed such a prescription, which we now present.

Let  $\Delta x_k = x_k - x_{k-1}$  and  $\Delta g_k = g_k - g_{k-1}$ . This method minimizes  $\|\Delta x - \alpha \Delta g\|^2$  by choosing

$$\alpha_k = \frac{\Delta x_k \cdot \Delta g_k}{\Delta g_k \cdot \Delta g_k}.$$

The algorithm of this gradient descent method can be summarized in the pseudocode of algorithm 2. It is described further in Barzilai and Borwein (1988).

**Algorithm 2** Barzilai and Borwein gradient descent

---

```

1: procedure barzilai_minimizer( $f, x_0, T$ )
2:   while  $\delta x > T$  do                                      $\triangleright$  until threshold  $T$  is met
3:      $g_k \leftarrow \text{grad } f(x_k)$ 
4:      $\Delta g_k \leftarrow g_k - g_{k-1}$ 
5:      $\Delta x_k \leftarrow x_k - x_{k-1}$ 
6:      $\alpha_k \leftarrow \frac{\Delta x_k \cdot \Delta g_k}{\Delta g_k \cdot \Delta g_k}$ 
7:      $x_{k+1} \leftarrow x_k - \alpha_k g_k$ 
8:      $\delta x \leftarrow \|x_{k+1} - x_k\|$ 
9:      $k \leftarrow k + 1$ 
10:  return  $x_k$                                               $\triangleright$  the threshold was reached

```

---

### Example 9.1

Consider the functions (a)  $f_1 : \mathbb{R}^2 \rightarrow \mathbb{R}$  and (b)  $f_2 : \mathbb{R}^2 \rightarrow \mathbb{R}$  defined as

$$f_1(\mathbf{x}) = (x_1 - 25)^2 + 13(x_2 + 10)^2$$

$$f_2(\mathbf{x}) = \frac{1}{2} \mathbf{x} \cdot A \mathbf{x} - \mathbf{b} \cdot \mathbf{x}$$

where

$$A = \begin{bmatrix} 20 & 0 \\ 0 & 10 \end{bmatrix} \quad \text{and} \quad (9.3)$$

$$\mathbf{b} = \begin{bmatrix} 1 & 1 \end{bmatrix}^\top. \quad (9.4)$$

Use the method of Barzilai and Borwein (1988) starting at some  $\mathbf{x}_0$  to find a minimum of each function.

First, load some Python packages.

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option("display.precision", 3) # Show only three decimal places
```

We begin by writing a class `Gradient_descent_min` to perform the gradient descent. This is not optimized for speed.

```
class Gradient_descent_min():
    """ A Barzilai and Borwein gradient descent class.

    Inputs:
        * f: Python function of x variables
        * x: list of symbolic variables (eg [x1, x2])
        * x0: list of numeric initial guess of a min of f
        * T: step size threshold for stopping the descent

    To execute the gradient descent call descend method.

    nb: This is only for gradients in cartesian
        coordinates! Further work would be to implement
        this in multiple or generalized coordinates.
        See the grad method below for implementation.
    """

    def __init__(self, f, x, x0, T):
        self.f = f
        self.x = sp.Array(x)
```

```

self.x0 = np.array(x0)
self.T = T
self.n = len(x0) # size of x
self.g = sp.lambdify(x,self.grad(f,x),'numpy')
self.xk = np.array(x0)
self.table = {}

def descend(self):
    # unpack variables
    f = self.f
    x = self.x
    x0 = self.x0
    T = self.T
    g = self.g
    # initialize variables
    N = 0
    x_k = x0
    dx = 2*T # can't be zero
    x_km1 = .9*x0-.1 # can't equal x0
    g_km1 = np.array(g(*x_km1))
    N_max = 100 # max iterations
    table_data = [[N,x0,np.array(g(*x0)),0]]
    while (dx > T and N < N_max) or N < 1:
        N += 1 # increment index
        g_k = np.array(g(*x_k))
        dg_k = g_k - g_km1
        dx_k = x_k - x_km1
        alpha_k = abs(dx_k.dot(dg_k)/dg_k.dot(dg_k))
        x_km1 = x_k # store
        x_k = x_k - alpha_k*g_k
        # save
        t_list = [N,x_k,g_k,alpha_k]
        t_list = [
            [f"{t_i:.3g}" for t_i in t] if isinstance(t,np.ndarray) \
            else t for t in t_list]
        table_data.append(t_list)
        self.xk = np.vstack((self.xk,x_k))
        # store other variables
        g_km1 = g_k
        dx = np.linalg.norm(x_k - x_km1) # check
    self.tabulater(table_data)

def tabulater(self,table_data):
    table = pd.DataFrame(table_data,columns=['N','x_k','g_k','alpha_k'])
    self.table['python'] = table
    self.table['latex'] = table.to_latex(index=False)

```

```
def grad(self,f,x): # cartesian coord's gradient
    return sp.derive_by_array(f(x),x)
```

First, consider  $f_1$ .

```
x1, x2 = sp.symbols('x1, x2')
x = sp.Array([x1, x2])
f1 = lambda x: (x[0]-25)**2 + 13*(x[1]+10)**2
gd = Gradient_descent_min(f=f1, x=x, x0=[-50,40], T=1e-8)
```

Perform the gradient descent.

```
gd.descend()
```

Print the interesting variables.

```
print(gd.table['python'])
```

	N	x_k	g_k	alpha_k
0	0	[-50, 40]	[-150, 1300]	0.000
1	1	[-43.7, -15]	[-150, 1.3e+03]	0.042
2	2	[-38.4, -10]	[-137, -131]	0.038
3	3	[-33.1, -10]	[-127, 0.124]	0.041
4	4	[25, -10]	[-116, -0.00962]	0.500
5	5	[25, -10.1]	[-0.0172, 0.115]	0.500
6	6	[25, -10]	[-1.84e-08, -1.38]	0.039
7	7	[25, -10]	[-1.7e-08, 0.00219]	0.038
8	8	[25, -10]	[-1.57e-08, 0]	0.038

Now let's lambdify the function  $f_1$  so we can plot.

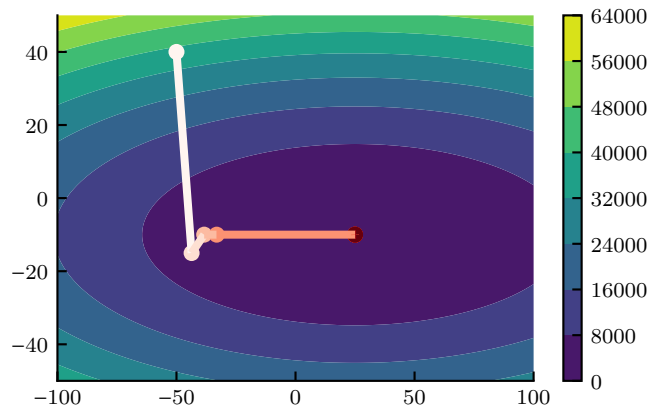
```
f1_lambda = sp.lambdify((x1, x2), f1(x), 'numpy')
```

Now let's plot a contour plot with the gradient descent overlaid.

```

fig, ax = plt.subplots()
# contour plot
X1 = np.linspace(-100,100,100)
X2 = np.linspace(-50,50,100)
X1, X2 = np.meshgrid(X1,X2)
F1 = f1_lambda(X1,X2)
plt.contourf(X1,X2,F1)
plt.colorbar()
# gradient descent plot
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.collections import LineCollection
xX1 = gd.xk[:,0]
xX2 = gd.xk[:,1]
points = np.array([xX1, xX2]).T.reshape(-1, 1, 2)
segments = np.concatenate(
    [points[:-1], points[1:]], axis=1
)
lc = LineCollection(
    segments,
    cmap=plt.get_cmap('Reds')
)
lc.set_array(np.linspace(0,1,len(xX1))) # color segs
lc.set_linewidth(3)
ax.autoscale(False) # avoid the scatter changing lims
ax.add_collection(lc)
ax.scatter(
    xX1,xX2,
    zorder=1,
    marker="o",
    color=plt.cm.Reds(np.linspace(0,1,len(xX1))),
    edgecolor='none'
)
plt.draw()

```

Figure 9.1. Gradient descent on  $f_1$ .

Now consider  $f_2$ .

```
A = sp.Matrix([[10, 0], [0, 20]])
b = sp.Matrix([[1, 1]])
def f2(x):
    X = sp.Array([x]).tomatrix().T
    return 1/2*X.dot(A*X) - b.dot(X)
gd = Gradient_descent_min(f=f2, x=x, x0=[50, -40], T=1e-8)
```

Perform the gradient descent.

```
gd.descend()
```

Print the interesting variables.

```
print(gd.table['python'])
```

	N	x_k	g_k	alpha_k
0	0	[50, -40]	[499.0, -801.0]	0.000
1	1	[17.6, 12]	[499, -801]	0.065
2	2	[8.07, -1.01]	[175, 240]	0.054
3	3	[3.62, 0.174]	[79.7, -21.2]	0.056
4	4	[0.489, -0.0468]	[35.2, 2.49]	0.089
5	5	[0.104, 0.145]	[3.89, -1.94]	0.099
6	6	[0.101, 0.00238]	[0.0381, 1.9]	0.075
7	7	[0.1, 0.05]	[0.00949, -0.952]	0.050
8	8	[0.1, 0.05]	[0.00474, 9.58e-05]	0.050
9	9	[0.1, 0.05]	[0.00237, -2.38e-09]	0.100
10	10	[0.1, 0.05]	[1.93e-06, 2.37e-09]	0.100
11	11	[0.1, 0.05]	[0, -2.37e-09]	0.100

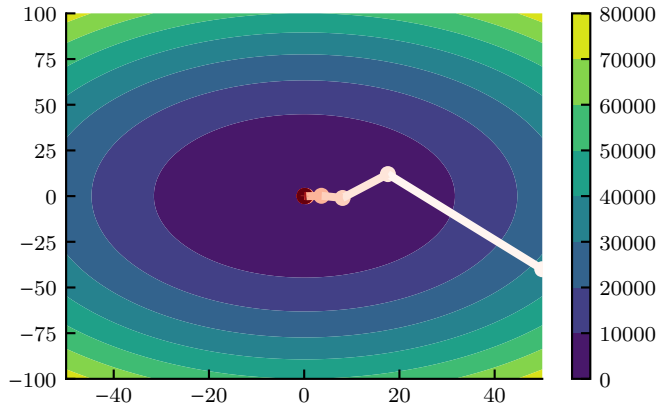
Now let's lambdify the function `f2` so we can plot.

```
f2_lambda = sp.lambdify((x1, x2), f2(x), 'numpy')
```

Now let's plot a contour plot with the gradient descent overlaid.

```
fig, ax = plt.subplots()
# contour plot
X1 = np.linspace(-100,100,100)
X2 = np.linspace(-50,50,100)
X1, X2 = np.meshgrid(X1,X2)
F2 = f2_lambda(X1,X2)
plt.contourf(X2,X1,F2)
plt.colorbar()
# gradient descent plot
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.collections import LineCollection
xX1 = gd.xk[:,0]
xX2 = gd.xk[:,1]
points = np.array([xX1, xX2]).T.reshape(-1, 1, 2)
segments = np.concatenate(
    [points[:-1], points[1:]], axis=1
)
lc = LineCollection(
    segments,
    cmap=plt.get_cmap('Reds')
)
lc.set_array(np.linspace(0,1,len(xX1))) # color segs
lc.set_linewidth(3)
ax.autoscale(False) # avoid the scatter changing lims
ax.add_collection(lc)
ax.scatter(
    xX1,xX2,
    zorder=1,
    marker="o",
    color=plt.cm.Reds(np.linspace(0,1,len(xX1))),
    edgecolor='none'
)
plt.show()
```



Figure 9.2. Gradient descent on  $f_2$ .

## 9.2 Constrained Linear Optimization

Consider a linear objective function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  with variables  $x_i$  in vector  $x$  and coefficients  $c_i$  in vector  $c$ :

$$f(x) = c \cdot x$$

subject to the linear **constraints**—restrictions on  $x_i$ —

$$Ax \leq a, \tag{9.5}$$

$$Bx = b, \text{ and} \tag{9.6}$$

$$l \leq x \leq u \tag{9.7}$$

where  $A$  and  $B$  are constant matrices and  $a, b, l, u$  are  $n$ -vectors. This is one formulation of what is called a **linear programming problem**. Usually we want to **maximize**  $f$  over the constraints. Such problems frequently arise throughout engineering, for instance in manufacturing, transportation, operations, etc. They are called **constrained** because there are constraints on  $x$ ; they are called **linear** because the objective function and the constraints are linear.

We call a pair  $(x, f(x))$  for which  $x$  satisfies equations (9.5) to (9.7) a **feasible solution**. Of course, not every feasible solution is **optimal**: a feasible solution is optimal iff there exists no other feasible solution for which  $f$  is greater (assuming we're maximizing). We call the vector subspace of feasible solutions  $S \subset \mathbb{R}^n$ .



### 9.2.1 Feasible Solutions Form a Polytope

Consider the effect of the constraints. Each of the equalities and inequalities defines a linear **hyperplane** in  $\mathbb{R}^n$  (i.e. a linear subspace of dimension  $n - 1$ ): either as a boundary of  $S$  (inequality) or as a restriction of  $S$  to the hyperplane. When joined, these hyperplanes are the boundary of  $S$  (equalities restrict  $S$  to lower dimension). So we see that each of the boundaries of  $S$  is **flat**, which makes  $S$  a **polytope** (in  $\mathbb{R}^2$ , a polygon). What makes this especially interesting is that polytopes have **vertices** where the hyperplanes intersect. Solutions at the vertices are called **basic feasible solutions**.

### 9.2.2 Only the Vertices Matter

Our objective function  $f$  is linear, so for some constant  $h$ ,  $f(\mathbf{x}) = h$  defines a **level set** that is itself a hyperplane  $H$  in  $\mathbb{R}^n$ . If this hyperplane intersects  $S$  at a point  $\mathbf{x}$ ,  $(\mathbf{x}, f(\mathbf{x}) = h)$  is the corresponding solution. There are three possibilities when  $H$  intersects  $S$ :

1.  $H \cap S$  is a vertex of  $S$ ,
2.  $H \cap S$  is a boundary hyperplane of  $S$ , or
3.  $H \cap S$  slices through the interior of  $S$ .

However, this third option implies that there exists a level set  $G$  corresponding to  $f(\mathbf{x}) = g$  such that  $G$  intersects  $S$  and  $g > h$ , so solutions on  $H \cap S$  are *not optimal*. (We have not proven this, but it may be clear from our progression.) We conclude that either the first or second case must be true for optimal solutions. And notice that in both cases, a (potentially optimal) solution occurs at at least one vertex. The key insight, then, is that **an optimal solution occurs at a vertex of  $S$** .

This means we don't need to search all of  $S$ , or even its boundary: we need only search the vertices. Helpful as this is, it restricts us down to  $\binom{n}{\# \text{ constraints}}$  potentially optimal solutions—usually still too many to search in a naïve way. In section 9.3, this is mitigated by introducing a powerful searching method.

### 9.3 The Simplex Algorithm



The **simplex algorithm** (or “method”) is an iterative technique for finding an optimal solution of the linear programming problem of section 9.2. The details of the algorithm are somewhat involved, but the basic idea is to start at a vertex of the feasible solution space  $S$  and traverse an edge of the polytope that leads to another vertex with a greater value of  $f$ . Then, repeat this process until there is no neighboring vertex with a greater value of  $f$ , at which point the solution is guaranteed to be optimal.

Rather than present the details of the algorithm, we choose to show an example using Python. There have been some improvements on the original algorithm that have been implemented into many standard software packages, including Python’s `scipy` package (Virtanen et al. 2019) module `scipy.optimize`.<sup>1</sup>

#### Example 9.2

Maximize the objective function

$$f(x) = c \cdot x \quad (9.8)$$

for  $x \in \mathbb{R}^2$  and

$$c = \begin{bmatrix} 5 & 2 \end{bmatrix}^T \quad (9.9)$$

subject to constraints

$$0 \leq x_1 \leq 10 \quad (9.10)$$

$$-5 \leq x_2 \leq 15 \quad (9.11)$$

$$4x_1 + x_2 \leq 40 \quad (9.12)$$

$$x_1 + 3x_2 \leq 35 \quad (9.13)$$

$$-8x_1 - x_2 \geq -75. \quad (9.14)$$

First, load some Python packages.

```
from scipy.optimize import linprog
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

**Encoding the Problem** Before we can use `linprog`, we must first encode equations (9.8) and (9.10) into a form `linprog` will recognize. We begin with  $f$ , which we can write as  $c \cdot x$  with the coefficients of  $c$  as follows.

1. Another Python package `pulp` (PuLP) is probably more popular for linear programming; however, we choose `scipy.optimize` because it has applications beyond linear programming.

```
| c = [-5, -2] # negative to find max
```

We've negated each constant because `linprog` *minimizes*  $f$  and we want to *maximize*  $f$ . Now let's encode the inequality constraints. We will write the left-hand side coefficients in the matrix  $A$  and the right-hand-side values in vector  $\mathbf{a}$  such that

$$Ax \leq \mathbf{a}. \quad (9.15)$$

Notice that one of our constraint inequalities is  $\geq$  instead of  $\leq$ . We can flip this by multiplying the inequality by  $-1$ . We use simple lists to encode  $A$  and  $\mathbf{a}$ .

```
| A = [  
|     [4, 1],  
|     [1, 3],  
|     [8, 1]  
| ]  
| a = [40, 35, 75]
```

Now we need to define the lower  $l$  and upper  $u$  bounds of  $x$ . The function `linprog` expects these to be in a single list of lower- and upper-bounds of each  $x_i$ .

```
| lu = [  
|     (0, 10),  
|     (-5, 15),  
| ]
```

We want to keep track of each step `linprog` takes. We can access these by defining a function callback, to be passed to `linprog`.

```
| x = [] # for storing the steps  
| def callback(res): # called at each step  
|     global x  
|     print(f"nit = {res.nit}, x_k = {res.x}")  
|     x.append(res.x.copy()) # store
```

Now we need to call `linprog`. We don't have any equality constraints, so we need only use the keyword arguments `A_ub=A` and `b_ub=a`. For demonstration purposes, we tell it to use the 'simplex' method, which is not as good as its other methods, which use better algorithms based on the simplex.

```

res = linprog(
    c,
    A_ub=A,
    b_ub=a,
    bounds=lu,
    method='simplex',
    callback=callback
)
x = np.array(x)

nit = 0, x_k = [ 0. -5.]
nit = 1, x_k = [10. -5.]
nit = 2, x_k = [8.75  5. ]
nit = 3, x_k = [7.72727273  9.09090909]
nit = 4, x_k = [7.72727273  9.09090909]
nit = 5, x_k = [7.72727273  9.09090909]
nit = 5, x_k = [7.72727273  9.09090909]

```

So the optimal solution  $(x, f(x))$  is as follows.

```

print(f"optimum x: {res.x}")
print(f"optimum f(x): {-res.fun}")

optimum x: [7.72727273  9.09090909]
optimum f(x): 56.81818181818182

```

The last point was repeated

- Once because there was no adjacent vertex with greater  $f(x)$  and
- Twice because the algorithm calls `callback` twice on the last step.

**Plotting** When the solution space is in  $\mathbb{R}^2$ , it is helpful to graphically represent the solution space, constraints, and the progression of the algorithm. We begin by defining the inequality lines from  $A$  and  $a$  over the bounds of  $x_1$ .

```

n = len(c) # number of variables x
m = np.shape(A)[0] # number of inequality constraints
x2 = np.empty([m,2])
for i in range(0,m):
    x2[i,:] = -A[i][0]/A[i][1]*np.array(lu[0]) + a[i]/A[i][1]

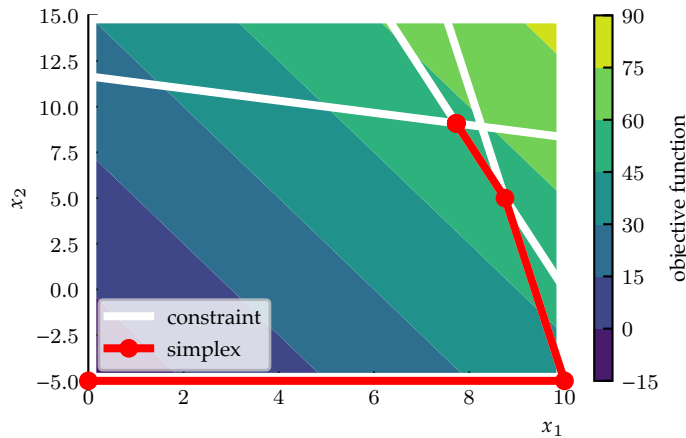
```

Now we plot a contour plot of  $f$  over the bounds of  $x_1$  and  $x_2$  and overlay the inequality constraints and the steps of the algorithm stored in `x`.

```

lu_array = np.array(lu)
fig, ax = plt.subplots()
mpl.rcParams['lines.linewidth'] = 3
# contour plot
X1 = np.linspace(*lu_array[0],100)
X2 = np.linspace(*lu_array[1],100)
X1, X2 = np.meshgrid(X1,X2)
F2 = -c[0]*X1 + -c[1]*X2 # negative because max hack
con = ax.contourf(X1,X2,F2)
cbar = fig.colorbar(con,ax=ax)
cbar.ax.set_ylabel('objective function')
# bounds on x
un = np.array([1,1])
opts = {'c':'w','label':None,'linewidth':6}
plt.plot(lu_array[0],lu_array[1,0]*un,**opts)
plt.plot(lu_array[0],lu_array[1,1]*un,**opts)
plt.plot(lu_array[0,0]*un,lu_array[1],**opts)
plt.plot(lu_array[0,1]*un,lu_array[1],**opts)
# inequality constraints
for i in range(0,m):
    p, = plt.plot(lu[0],x2[i,:],c='w')
p.set_label('constraint')
# steps
plt.plot(
    x[:,0], x[:,1], '-o', c='r',
    clip_on=False, zorder=20, label='simplex'
)
plt.ylim(lu_array[1])
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.show()

```

Figure 9.3. Simplex method on  $f$ .

#### 9.4 The Calculus of Variations



Thus far, we have been concerned with the optimization of functions, which have arguments that each correspond to a single value.

The **calculus of variations** is a branch of mathematics that extends the concept of optimization to functions that have arguments that correspond to entire functions—called **functionals**. This is closely related to the field of **functional analysis**, which is concerned with the study of spaces of functions. The calculus of variations is used to find the function that minimizes or maximizes a given functional. In other words, the calculus of variations attempts to optimize an **objective functional** by finding the function that produces the optimal output value.

The most important objective functional in the calculus of variations is the definite integral of a function  $f(x, y(x), y'(x))$  over an interval  $[a, b]$ , where  $y(x)$  is the function to be optimized:

$$J[y] = \int_a^b f(x, y(x), y'(x)) dx. \quad (9.16)$$

We use the notation  $J[y]$  to denote the functional, which varies with function  $y(x)$ . The goal of the calculus of variations is to find the function  $y(x)$  that minimizes or maximizes the integral  $J[y]$  subject to the boundary conditions  $y(a) = y_a$  and  $y(b) = y_b$ .

The calculus of variations is used in many areas of mathematics and physics, including the study of **geodesics** in **Riemannian geometry**, the path of least time in **optics**, and the **principle of least action** in **classical mechanics**. The last example is particularly important, as it is the basis for the **Lagrangian formulation** and the **Hamiltonian formulation** of classical mechanics.

#### 9.4.1 The Euler-Lagrange Equation

The key result in the calculus of variations is the **Euler-Lagrange equation**, which provides a necessary condition for a function to be an extremum of a functional. We will derive the Euler-Lagrange equation for the functional  $J[y]$  defined above.

Let  $y = \hat{y}(x)$  be a function that minimizes or maximizes the objective functional  $J[y]$  given in equation (9.16). We will consider a perturbation of the optimal solution  $\hat{y}$  constructed by adding a function  $\eta(x)$  scaled by a small parameter  $\epsilon$ ,

$$y(x) = \hat{y}(x) + \epsilon\eta(x),$$

where  $\eta(x)$  is a smooth function that satisfies the boundary conditions  $\eta(a) = \eta(b) = 0$ . The function  $\epsilon\eta(x)$  is called a **variation** of the function  $y(x)$ . There are important details regarding the norm used to quantify the size of the variation (Kot 2014; § 2.3), but we will not delve into them here. We assume here that  $\eta(x)$  is independent of  $\epsilon$ .

Thus we can consider the functional  $J[y]$  to be homologous to a function of the parameter  $\epsilon$ :

$$J(\epsilon) \equiv J[\hat{y} + \epsilon\eta] = \int_a^b f(x, \hat{y} + \epsilon\eta, \hat{y}' + \epsilon\eta') dx.$$

The function  $J(\epsilon)$  is minimized or maximized at  $\epsilon = 0$  (i.e.,  $y = \hat{y}$ ). The **total variation** is

$$\Delta J = J(\epsilon) - J(0) = \int_a^b f(x, \hat{y} + \epsilon\eta, \hat{y}' + \epsilon\eta') - f(x, \hat{y}, \hat{y}') dx.$$

The Taylor expansion of the total variation is

$$\Delta J = \delta J + \frac{1}{2}\delta^2 J + \mathcal{O}(\epsilon^3),$$

where<sup>2</sup>

$$\delta J = \left. \frac{dJ(\epsilon)}{d\epsilon} \right|_{\epsilon=0} = \epsilon \int_a^b f_y(x, \hat{y}, \hat{y}')\eta + f_{y'}(x, \hat{y}, \hat{y}')\eta' dx \quad (9.17)$$

is called the **first variation** and

$$\delta^2 J = \left. \frac{d^2 J(\epsilon)}{d\epsilon^2} \right|_{\epsilon=0} = \epsilon^2 \int_a^b f_{yy}(x, \hat{y}, \hat{y}')\eta^2 + 2f_{yy'}(x, \hat{y}, \hat{y}')\eta\eta' + f_{y'y'}(x, \hat{y}, \hat{y}')\eta'^2 dx$$

is called the **second variation**.

2. The notation  $f_y$  and  $f_{y'}$  denotes the partial derivatives of  $f$  with respect to  $y$  and  $y'$ , respectively.



If the functional  $J[y]$  is minimized at  $\epsilon = 0$ , for sufficiently small  $\epsilon$ , the total variation must satisfy

$$\Delta J \geq 0.$$

In other words,  $\Delta J$  cannot be negative for any  $\epsilon$ . However, the odd, linear first variation  $\delta J$  could be negative for  $\epsilon < 0$ . Because, for sufficiently small  $\epsilon$ , the linear term  $\delta J$  dominates  $\Delta J$ , we conclude that  $\delta J = 0$ . This condition is analogous to the first-derivative test for extrema in single-variable calculus. Moreover, for a minimum, the second variation  $\delta^2 J$  must be nonnegative for any  $\epsilon$ ,

$$\delta^2 J \geq 0.$$

This condition is analogous to the second-derivative test for extrema in single-variable calculus.

Conversely, if the functional  $J[y]$  is maximized at  $\epsilon = 0$ , the total variation must satisfy

$$\delta J = 0 \quad \text{and} \quad \delta^2 J \leq 0.$$

We can summarize the first-variation results in the following lemma.

#### Lemma 9.1: First Variation Condition

If  $y = \hat{y}(x)$  is an extremum of the functional  $J[y]$  defined in equation (9.16), then the first variation  $\delta J$  of the functional must satisfy the condition

$$\delta J = 0$$

for  $y = \hat{y}(x)$  and for any admissible variation  $\epsilon \eta(x)$  of the function  $y$ . Note that this is a necessary condition for an extremum of the functional.

The first variation  $\delta J$  of equation (9.17) is unwieldy. Integration by parts can simplify the expression to (§ 2.3.1)

$$\delta J = \eta \int_a^b M(x) \eta(x) dx,$$

where

$$M(x) = f_y(x, \hat{y}, \hat{y}') - \frac{d}{dx} f_{y'}(x, \hat{y}, \hat{y}'). \quad (9.18)$$

The first variation condition of lemma 9.1 requires that  $\delta J = 0$ . Although it is not obvious, this implies that  $M(x)$  must be zero for all  $x \in [a, b]$ , as is stated in the following lemma.

#### Lemma 9.2: Fundamental Lemma of the Calculus of Variations

If all of the following conditions are met:

- $M(x)$  is a continuous real-valued function on  $[a, b]$

- $\eta(x)$  and  $\eta'(x)$  are continuous functions on  $[a, b]$
- $\eta(a) = \eta(b) = 0$
- $\int_a^b M(x)\eta(x) dx = 0,$

then  $M(x) = 0$  for all  $x \in [a, b]$ .

The truth of this lemma is not immediately obvious, but is well established (Kot 2014; § 2.3.1).

An important consequence of lemma 9.2 is the **Euler-Lagrange equation**, which is the main result of the calculus of variations. From lemma 9.2, we have that  $M(x) = 0$  for all  $x \in [a, b]$ . So equation (9.18) can be set to zero. This is most commonly written as

$$\frac{\partial f}{\partial y} - \frac{d}{dx} \frac{\partial f}{\partial y'} = 0. \quad (9.19)$$

This is the Euler-Lagrange equation for the functional  $J[y]$  defined in equation (9.16). We summarize this result in the following lemma.

### Lemma 9.3: Euler-Lagrange Equation

If  $y = \hat{y}(x)$  is an extremum of the functional  $J[y]$  defined in equation (9.16), then the function  $\hat{y}(x)$  must satisfy the Euler-Lagrange equation

$$\frac{\partial f}{\partial y} - \frac{d}{dx} \frac{\partial f}{\partial y'} = 0.$$

This is a necessary condition for an extremum of the functional.

In this form, the Euler-Lagrange equation can be applied to functionals of the form of equation (9.16) to find a necessary condition for the function that minimizes or maximizes the functional. Let us consider a simple example to illustrate the application of the Euler-Lagrange equation.

### Example 9.3

Consider the problem of finding the shortest path between two points  $p_1 = (x_1, y_1)$  and  $p_2 = (x_2, y_2)$  in the plane. We know intuitively that the shortest path is a straight line. However, we can use the calculus of variations to prove this fact.

For a sufficiently small segment of the path, the length  $ds$  of the path is straight and given by the Pythagorean theorem:

$$ds = \sqrt{dx^2 + dy^2}, \quad (9.20)$$

where  $dx$  and  $dy$  are the corresponding changes in  $x$  and  $y$  along the segment. The total length of the path is the integral of  $ds$  over the path:

$$J[y] = \int_{p_1}^{p_2} ds. \quad (9.21)$$

We have called the functional  $J[y]$  because it is the functional we would like to minimize (i.e., it is the objective functional). Using the chain rule,

$$dy = \frac{dy}{dx} dx = y'(x) dx,$$

so we can rewrite equation (9.20) such that the integrand is a function of  $x$ ,  $y$ , and  $y' = dy/dx$ :

$$ds = \sqrt{1 + y'(x)^2} dx.$$

So the functional can be written from equation (9.21) as

$$J[y] = \int_{x_1}^{x_2} \sqrt{1 + y'(x)^2} dx. \quad (9.22)$$

We can apply the Euler-Lagrange equation equation (9.19) to this functional to find the function  $y(x)$  that minimizes the path length. Identifying  $f(x, y, y') = \sqrt{1 + y'(x)^2}$ , we have

$$\frac{\partial f}{\partial y} = 0 \quad \text{and} \quad \frac{\partial f}{\partial y'} = \frac{y'}{\sqrt{1 + y'(x)^2}}.$$

So the Euler-Lagrange equation becomes

$$\frac{d}{dx} \frac{y'}{\sqrt{1 + y'(x)^2}} = 0.$$

This implies that for some real constant  $C_1$ ,

$$\frac{y'}{\sqrt{1 + y'(x)^2}} = C_1,$$

which can be solved for  $y'(x)$  to show that  $y'(x)$  is a real constant  $C_2$ . Thus the function  $y(x)$  is a straight line,

$$y(x) = C_2 x + C_3,$$

for real constants  $C_2$  and  $C_3$ . This is the equation of a straight line, as we expected.

Note that this result is just a necessary condition for the shortest path. More is required to show that this is also a sufficient condition.

For more complex spaces, such as the surface of a toroid, the shortest path is not a straight line, but a **geodesic**. The same method can be used to find the geodesic on a surface.

### 9.4.2 A Multivariate, Parametric Euler-Lagrange Equations

For many problems, there are multiple functions that must be optimized simultaneously in a functional. Consider the optimization of a functional  $J[q_1, q_2, \dots, q_n]$  that depends on  $n$  functions  $q_1(t), q_2(t), \dots, q_n(t)$ , where  $t$  is a parameter. The functional  $J$  is defined as

$$J[q_1, q_2, \dots, q_n] = \int_{t_1}^{t_2} L(t, q_1, q_2, \dots, q_n, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_n) dt,$$

where  $\dot{q}_i = dq_i/dt$  and  $L$  is a function of  $t, q_i$ , and  $\dot{q}_i$ . The goal is to find the functions  $q_i(t)$  that minimize or maximize the functional  $J$ .

The Euler-Lagrange equation for this multivariate, parametric functional is derived in a similar manner to the single-variable case.

#### Lemma 9.4: Euler-Lagrange Equations for Multivariate, Parametric Functionals

If  $q_i = \hat{q}_i(t)$  are extrema of the functional  $J[q_1, q_2, \dots, q_n]$  defined above, then the functions  $\hat{q}_i(t)$  must satisfy the Euler-Lagrange equations

$$\frac{\partial L}{\partial q_i} - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} = 0$$

for  $i = 1, 2, \dots, n$ . These are necessary conditions for extrema of the functional.

### 9.4.3 Application to Lagrangian Mechanics

The Euler-Lagrange equation is a fundamental tool in the study of **Lagrangian mechanics**, which is a reformulation of Newtonian classical mechanics that uses the concept of action to describe the motion of particles. We begin with a few definitions.

#### Definition 9.5: Generalized Coordinates

The generalized coordinates  $q_i$  of a mechanical system are a set of coordinates that describe the position and orientation of the system. The number of generalized coordinates is equal to the number of degrees of freedom of the system. Generalized coordinates are not necessarily Cartesian, independent, or inertial.

Generalized coordinates are used to define the kinetic and potential energy of a mechanical system. They enter into the following definitions. We can think of the set of generalized coordinates  $q_i$  as comprising a space called the **configuration space**. The time evolution of the system is described by the path  $\mathbf{q}(t)$  in configuration space. The velocity of the system through configuration space is given by the vector  $\dot{\mathbf{q}}(t)$ .

**Definition 9.6: Lagrangian**

The Lagrangian  $L$  of a mechanical system is a function of time  $t$ , a position vector  $\mathbf{q}$ , and the velocity vector  $\dot{\mathbf{q}}$  of the system, and equals the difference between the kinetic energy  $T$  and the potential energy  $U$  of the system:

$$L(t, \mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - U(\mathbf{q}).$$

**Definition 9.7: Action**

The action  $S$  of a system moving through its configuration space on a path  $\mathbf{q}(t)$  from time  $t_1$  to time  $t_2$  is defined as

$$S[\mathbf{q}] = \int_{t_1}^{t_2} L(t, \mathbf{q}, \dot{\mathbf{q}}) dt,$$

for Lagrangian  $L$ .

We can now state the principle of least action.

**Definition 9.8: Principle of Least Action**

For a mechanical system with only energy conserving forces, the path  $\mathbf{q}(t)$  the system takes through its configuration space is such that the action  $S[\mathbf{q}]$  is minimized.

The principle of least action is related to Hamilton's principle.

**Definition 9.9: Hamilton's Principle**

For a mechanical system with only energy conserving forces, the path  $\mathbf{q}(t)$  the system takes through its configuration space is such that first variation of the action  $S[\mathbf{q}]$  is zero:

$$\delta S = 0.$$

A consequence of Hamilton's principle is that the Euler-Lagrange equations for the Lagrangian  $L$  are satisfied by the path  $\mathbf{q}(t)$  that the particle takes.

**Lemma 9.10: Euler-Lagrange Equations in Lagrangian Mechanics**

For a mechanical system with only energy conserving forces, the path  $\mathbf{q}(t)$  the system takes through its configuration space must satisfy the Euler-Lagrange equations

$$\frac{\partial L}{\partial q_i} - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} = 0$$

for  $i = 1, 2, \dots, n$ .

We are now ready to apply the Euler-Lagrange equations to a mechanical system.

#### Example 9.4


Consider a flexible pendulum of unstretched length  $\ell$  with mass  $m$  affixed to its end, free to move in a vertical plane (i.e., the pendulum is subject to gravity). The pendulum is sufficiently stiff transverse to its length that it can be treated as straight. Its length is given by  $\ell + \lambda(t)$ , where  $\lambda(t)$  is the displacement of the pendulum from its unstretched length. The pendulum is also described by the angle  $\theta(t)$  that the pendulum makes with the downward vertical.

Derive the equations of motion for the flexible pendulum. Numerically integrate the equations of motion to find the motion of the pendulum for slightly different initial conditions. Does the pendulum exhibit chaotic motion?

asdf

## 9.5 Problems



**Problem 9.1**  **CHORTLE** Consider the function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ , defined as

$$f(\mathbf{x}) = \cos(x_1 - e^{x_2} + 2) \sin(x_1^2/4 - x_2^2/3 + 4) \quad (9.23)$$

Use the method of Barzilai and Borwein (1988) starting at  $\mathbf{x}_0 = (1, 1)$  to find a minimum of the function.

**Problem 9.2**  **CUMMERBUND** Consider the functions (a)  $f_1 : \mathbb{R}^2 \rightarrow \mathbb{R}$  and (b)  $f_2 : \mathbb{R}^2 \rightarrow \mathbb{R}$  defined as

$$f_1(\mathbf{x}) = 4(x_1 - 16)^2 + (x_2 + 64)^2 + x_1 \sin^2 x_1 \quad (9.24)$$

$$f_2(\mathbf{x}) = \frac{1}{2} \mathbf{x} \cdot A \mathbf{x} - \mathbf{b} \cdot \mathbf{x} \quad (9.25)$$

where

$$A = \begin{bmatrix} 5 & 0 \\ 0 & 15 \end{bmatrix} \quad \text{and} \quad (9.26a)$$


$$\mathbf{b} = \begin{bmatrix} -2 & 1 \end{bmatrix}^\top. \quad (9.26b)$$

Use the method of Barzilai and Borwein (1988) starting at some  $\mathbf{x}_0$  to find a minimum of each function.

**Problem 9.3**  Consider the function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  defined as

$$f(\mathbf{x}) = \sin x_1 + \cos x_2 + \sqrt{(x_1 - 2)^2 + (x_2 + 1)^2}. \quad (9.27)$$

Use the gradient descent method of Barzilai and Borwein (1988) with a step size of  $T = 10^{-8}$  starting at (a)  $\mathbf{x}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top$  and (b)  $\mathbf{x}'_0 = \begin{bmatrix} 2 & 0 \end{bmatrix}^\top$  to find minima of  $f$ . (c) Explain why the two minima are different.

**Problem 9.4**  **MELTY** Maximize the objective function

$$f(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x} \quad (9.28a)$$

for  $\mathbf{x} \in \mathbb{R}^3$  and

$$\mathbf{c} = \begin{bmatrix} 3 & -8 & 1 \end{bmatrix}^\top \quad (9.28b)$$

subject to constraints

$$0 \leq x_1 \leq 20 \quad (9.29a)$$


$$-5 \leq x_2 \leq 0 \quad (9.29b)$$

$$5 \leq x_3 \leq 17 \quad (9.29c)$$

$$x_1 + 4x_2 \leq 50 \quad (9.29d)$$

$$2x_1 + x_3 \leq 43 \quad (9.29e)$$

$$-4x_1 + x_2 - 5x_3 \geq -99. \quad (9.29f)$$

**Problem 9.5**  **LATENESS** Using gradient decent find the minimum of the function,

$$f(x) = x_1^2 + x_2^2 - \frac{x_1}{10} + \cos(2x_1),$$

starting at the location  $x = [-0.5, 0.75]^T$ , and with a constant value  $\alpha = 0.01$ .

- a. What is the location of the minimum you found?
- b. Is this location the global minimum?



# 10 Nonlinear Analysis



The ubiquity of near-linear systems and the tools we have for analyses thereof can sometimes give the impression that nonlinear systems are exotic or even downright flamboyant. However, a great many systems<sup>1</sup> important for a mechanical engineer are frequently hopelessly nonlinear. Here are some examples of such systems.

- A robot arm.
- Viscous fluid flow (usually modelled by the Navier-Stokes equations).
- Nonequilibrium thermodynamics.
- Anything that “fills up” or “saturates.”
- Nonlinear optics.
- Einstein’s field equations (gravitation in general relativity).
- Heat radiation and nonlinear heat conduction.
- Fracture mechanics.
- The 3-body problem.

Lest we think this is merely an inconvenience, we should keep in mind that it is actually the nonlinearity that makes many phenomena useful. For instance, the laser depends on the nonlinearity of its optics. Similarly, transistors and the digital circuits made thereby (including the microprocessor) wouldn’t function if their physics were linear.

In this chapter, we will see some ways to formulate, characterize, and simulate nonlinear systems. Purely analytic techniques are few for nonlinear systems. Most are beyond the scope of this text, but we describe a few, mostly in (**lec:nonlinear-system-characteristics**). Simulation via numerical integration of nonlinear dynamical equations is the most accessible technique, so it is introduced.

We skip a discussion of linearization; of course, if this is an option, it is preferable. Instead, we focus on the nonlinearizable.

1. As is customary, we frequently say “system” when we mean “mathematical system model.” Recall that multiple models may be used for any given physical system, depending on what one wants to know.

For a good introduction to nonlinear dynamics, see (Strogatz and Dichter 2016). A more engineer-oriented introduction is (Kolk and Lerman 1993).

## 10.1 Nonlinear State-Space Models



A state-space model has the general form

$$\frac{dx}{dt} = f(x, u, t) \quad (10.1)$$

$$y = g(x, u, t) \quad (10.2)$$

where  $f$  and  $g$  are vector-valued functions that depend on the system. **Nonlinear state-space models** are those for which  $f$  is a nonlinear functional of either  $x$  or  $u$ . For instance, a state variable  $x_1$  might appear as  $x_1^2$  or two state variables might combine as  $x_1 x_2$  or an input  $u_1$  might enter the equations as  $\log u_1$ .

### 10.1.1 Autonomous and Nonautonomous Systems

An **autonomous system** is one for which  $f(x)$ , with neither time nor input appearing explicitly. A **nonautonomous system** is one for which either  $t$  or  $u$  do appear explicitly in  $f$ . It turns out that we can always write nonautonomous systems as autonomous by substituting in  $u(t)$  and introducing an extra state variable for  $t$  (Strogatz and Dichter 2016).

Therefore, without loss of generality, we will focus on ways of analyzing autonomous systems.

### 10.1.2 Equilibrium

An **equilibrium state** (also called a stationary point)  $\bar{x}$  is one for which  $dx/dt = 0$ . In most cases, this occurs only when the input  $u$  is a constant  $\bar{u}$  and, for time-varying systems, at a given time  $\bar{t}$ . For autonomous systems, equilibrium occurs when the following holds:

$$f(\bar{x}) = 0.$$

This is a system of nonlinear algebraic equations, which can be challenging to solve for  $\bar{x}$ . However, frequently, several solutions—that is, equilibrium states—do exist.

## 10.2 Nonlinear System Characteristics



Characterizing nonlinear systems can be challenging without the tools developed for linear system characterization. However, there are ways of characterizing nonlinear systems, and we'll here explore a few.

### 10.2.1 Those In-Common with Linear Systems

As with linear systems, the **system order** is either the number of state-variables required to describe the system or, equivalently, the highest-order derivative in a single scalar differential equation describing the system.

Similarly, nonlinear systems can have state variables that depend on time alone or those that also depend on space (or some other independent variable). The former lead to ordinary differential equations (ODEs) and the latter to partial differential equations (PDEs).

Equilibrium was already considered in section 10.1.2.

### 10.2.2 Stability

In terms of system performance, perhaps no other criterion is as important as stability.

#### Definition 10.1

If  $x$  is perturbed from an equilibrium state  $\bar{x}$ , the response  $x(t)$  can:

1. asymptotically return to  $\bar{x}$  (asymptotically stable),
2. diverge from  $\bar{x}$  (unstable), or
3. remain perturbed or oscillate about  $\bar{x}$  with a constant amplitude (marginally stable).

Notice that this definition is actually local: stability in the neighborhood of one equilibrium may not be the same as in the neighborhood of another.

Other than nonlinear systems' lack of linear systems' eigenvalues, poles, and roots of the characteristic equation from which to compute it, the primary difference between the stability of linear and nonlinear systems is that nonlinear system stability is often difficult to establish globally. Using a linear system's eigenvalues, it is straightforward to establish stable, unstable, and marginally stable subspaces of state-space (via transforming to an eigenvector basis). For nonlinear systems, no such method exists. However, we are not without tools to explore nonlinear system stability. One mathematical tool to consider is Lyapunov stability theory, which is beyond the scope of this course, but has good treatments in (Brogan 1991; Ch. 10) and (Choukchou-Braham et al. 2013; App. A).

### 10.2.3 Qualities of Equilibria

Equilibria (i.e. stationary points) come in a variety of qualities. It is instructive to consider the first-order differential equation in state variable  $x$  with real constant  $r$ :

$$x' = rx - x^3.$$

If we plot  $x'$  versus  $x$  for different values of  $r$ , we obtain the plots of figure 10.1.

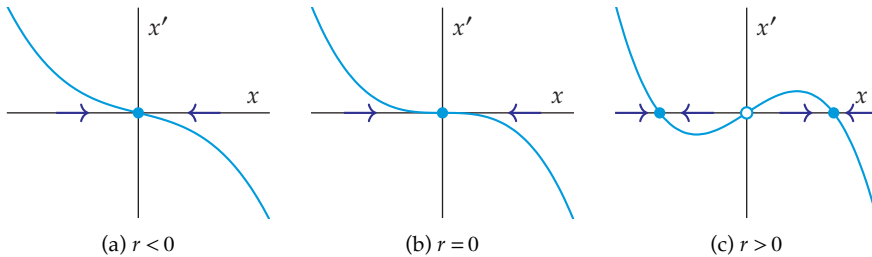


Figure 10.1. Plots of  $x'$  versus  $x$  for section 10.2.3.

By definition, equilibria occur when  $x' = 0$ , so the  $x$ -axis crossings of figure 10.1 are equilibria. The blue arrows on the  $x$ -axis show the direction (sign) of state change  $x'$ , quantified by the plots. For both (a) and (b), only one equilibrium exists:  $x = 0$ . Note that the blue arrows in both plots point *toward* the equilibrium. In such cases—that is, when a neighborhood exists around an equilibrium for which state changes point toward the equilibrium—the equilibrium is called an **attractor** or **sink**. Note that attractors are stable.

Now consider (c) of figure 10.1. When  $r > 0$ , three equilibria emerge. This change of the number of equilibria with the changing of a parameter is called a **bifurcation**. A plot of bifurcations versus the parameter is called a **bifurcation diagram**. The  $x = 0$  equilibrium now has arrows that point away from it. Such an equilibrium is called a **repeller** or **source** and is unstable. The other two equilibria here are (stable) attractors. Consider a very small initial condition  $x(0) = \epsilon$ . If  $\epsilon > 0$ , the repeller pushes away  $x$  and the positive attractor pulls  $x$  to itself. Conversely, if  $\epsilon < 0$ , the repeller again pushes away  $x$  and the negative attractor pulls  $x$  to itself.

Another type of equilibrium is called the saddle: one which acts as an attractor along some lines and as a repeller along others. We will see this type in the following example.

**Example 10.1**

Consider the dynamical equation

$$x' = x^2 + r$$

with  $r$  a real constant. Sketch  $x'$  vs  $x$  for negative, zero, and positive  $r$ . Identify and classify each of the equilibria.

TODO

**10.3 Simulating Nonlinear Systems****Example 10.2**

Simulate a nonlinear unicycle in Python.



First, load some Python packages.

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
```

The state equation can be encoded via the following function  $f$ .

```
def f(t, x, u, c):
    dxdt = [
        x[3]*np.cos(x[2]),
        x[3]*np.sin(x[2]),
        x[4],
        1/c[0] * u(t)[0],
        1/c[1] * u(t)[1]
    ]
    return dxdt
```

The input function  $u$  must also be defined.

```
def u(t):
    return [
        15*(1+np.cos(t)),
        25*np.sin(3*t)
    ]
```

Define time spans, initial values, and constants

```
tspan = np.linspace(0, 50, 300)
xinit = [0,0,0,0,0]
mass = 10
inertia = 10
c = [mass,inertia]
```

Solve differential equation:

```
sol = solve_ivp(
    lambda t, x: f(t, x, u, c),
    [tspan[0], tspan[-1]],
    xinit,
    t_eval=tspan
)
```

Let's first plot the trajectory and instantaneous velocity.

```
xp = sol.y[3]*np.cos(sol.y[2])
yp = sol.y[3]*np.sin(sol.y[2])
fig, ax = plt.subplots()
plt.plot(sol.y[0], sol.y[1])
plt.quiver(sol.y[0], sol.y[1], xp, yp)
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

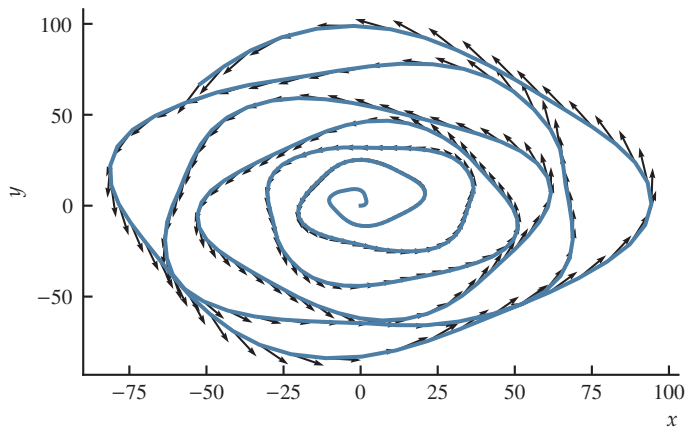


Figure 10.2. Trajectory and instantaneous velocity.

---

## 10.4 Problems







# A Distribution Tables

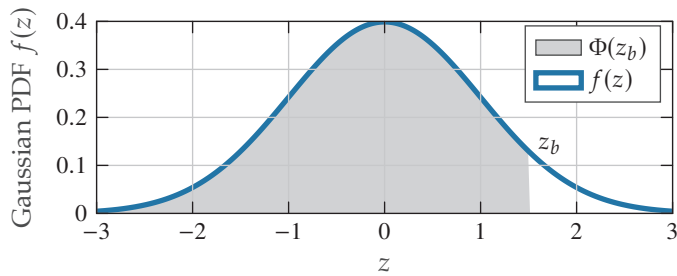


This appendix includes the Gaussian distribution table and Student's t-distribution table.

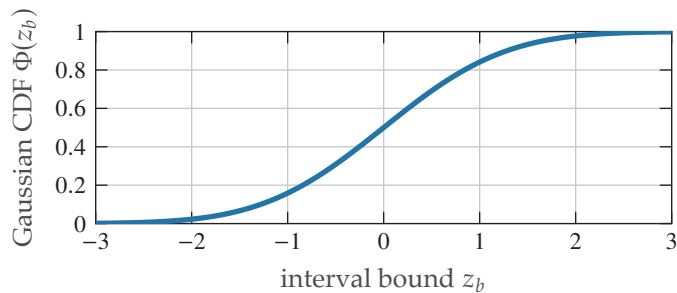
## A.1 Gaussian Distribution Table



Below are plots of the Gaussian probability density function  $f$  and cumulative distribution function  $\Phi$ . Below them is Table A.1 of CDF values.



(a) PDF



(b) CDF

Figure A.1. The Gaussian PDF and CDF for z-scores.

Table A.1. z-score table  $\Phi(z_b) = P(z \in (-\infty, z_b])$ .

$z_b$	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9
-3.4	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0002
-3.3	0.0005	0.0005	0.0005	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0003
-3.2	0.0007	0.0007	0.0006	0.0006	0.0006	0.0006	0.0006	0.0005	0.0005	0.0005
-3.1	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
-3.0	0.0013	0.0013	0.0013	0.0012	0.0012	0.0011	0.0011	0.0011	0.0010	0.0010
-2.9	0.0019	0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014	0.0014
-2.8	0.0026	0.0025	0.0024	0.0023	0.0023	0.0022	0.0021	0.0021	0.0020	0.0019
-2.7	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0026
-2.6	0.0047	0.0045	0.0044	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036
-2.5	0.0062	0.0060	0.0059	0.0057	0.0055	0.0054	0.0052	0.0051	0.0049	0.0048
-2.4	0.0082	0.0080	0.0078	0.0075	0.0073	0.0071	0.0069	0.0068	0.0066	0.0064
-2.3	0.0107	0.0104	0.0102	0.0099	0.0096	0.0094	0.0091	0.0089	0.0087	0.0084
-2.2	0.0139	0.0136	0.0132	0.0129	0.0125	0.0122	0.0119	0.0116	0.0113	0.0110
-2.1	0.0179	0.0174	0.0170	0.0166	0.0162	0.0158	0.0154	0.0150	0.0146	0.0143
-2.0	0.0228	0.0222	0.0217	0.0212	0.0207	0.0202	0.0197	0.0192	0.0188	0.0183
-1.9	0.0287	0.0281	0.0274	0.0268	0.0262	0.0256	0.0250	0.0244	0.0239	0.0233
-1.8	0.0359	0.0351	0.0344	0.0336	0.0329	0.0322	0.0314	0.0307	0.0301	0.0294
-1.7	0.0446	0.0436	0.0427	0.0418	0.0409	0.0401	0.0392	0.0384	0.0375	0.0367
-1.6	0.0548	0.0537	0.0526	0.0516	0.0505	0.0495	0.0485	0.0475	0.0465	0.0455
-1.5	0.0668	0.0655	0.0643	0.0630	0.0618	0.0606	0.0594	0.0582	0.0571	0.0559
-1.4	0.0808	0.0793	0.0778	0.0764	0.0749	0.0735	0.0721	0.0708	0.0694	0.0681
-1.3	0.0968	0.0951	0.0934	0.0918	0.0901	0.0885	0.0869	0.0853	0.0838	0.0823
-1.2	0.1151	0.1131	0.1112	0.1093	0.1075	0.1056	0.1038	0.1020	0.1003	0.0985
-1.1	0.1357	0.1335	0.1314	0.1292	0.1271	0.1251	0.1230	0.1210	0.1190	0.1170
-1.0	0.1587	0.1562	0.1539	0.1515	0.1492	0.1469	0.1446	0.1423	0.1401	0.1379
-0.9	0.1841	0.1814	0.1788	0.1762	0.1736	0.1711	0.1685	0.1660	0.1635	0.1611
-0.8	0.2119	0.2090	0.2061	0.2033	0.2005	0.1977	0.1949	0.1922	0.1894	0.1867
-0.7	0.2420	0.2389	0.2358	0.2327	0.2296	0.2266	0.2236	0.2206	0.2177	0.2148
-0.6	0.2743	0.2709	0.2676	0.2643	0.2611	0.2578	0.2546	0.2514	0.2483	0.2451
-0.5	0.3085	0.3050	0.3015	0.2981	0.2946	0.2912	0.2877	0.2843	0.2810	0.2776
-0.4	0.3446	0.3409	0.3372	0.3336	0.3300	0.3264	0.3228	0.3192	0.3156	0.3121
-0.3	0.3821	0.3783	0.3745	0.3707	0.3669	0.3632	0.3594	0.3557	0.3520	0.3483
-0.2	0.4207	0.4168	0.4129	0.4090	0.4052	0.4013	0.3974	0.3936	0.3897	0.3859
-0.1	0.4602	0.4562	0.4522	0.4483	0.4443	0.4404	0.4364	0.4325	0.4286	0.4247
-0.0	0.5000	0.4960	0.4920	0.4880	0.4840	0.4801	0.4761	0.4721	0.4681	0.4641
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706



## A.2 Student's T-Distribution Table

Below is the two-tail inverse student's t-distribution table.



Table A.2: Two-tail inverse student's t-distribution table.

$\nu$	percent probability										
	60.0	66.7	75.0	80.0	87.5	90.0	95.0	97.5	99.0	99.5	99.9
1	0.325	0.577	1.000	1.376	2.414	3.078	6.314	12.706	31.821	63.657	318.31
2	0.289	0.500	0.816	1.061	1.604	1.886	2.920	4.303	6.965	9.925	22.327
3	0.277	0.476	0.765	0.978	1.423	1.638	2.353	3.182	4.541	5.841	10.215
4	0.271	0.464	0.741	0.941	1.344	1.533	2.132	2.776	3.747	4.604	7.173
5	0.267	0.457	0.727	0.920	1.301	1.476	2.015	2.571	3.365	4.032	5.893
6	0.265	0.453	0.718	0.906	1.273	1.440	1.943	2.447	3.143	3.707	5.208
7	0.263	0.449	0.711	0.896	1.254	1.415	1.895	2.365	2.998	3.499	4.785
8	0.262	0.447	0.706	0.889	1.240	1.397	1.860	2.306	2.896	3.355	4.501
9	0.261	0.445	0.703	0.883	1.230	1.383	1.833	2.262	2.821	3.250	4.297
10	0.260	0.444	0.700	0.879	1.221	1.372	1.812	2.228	2.764	3.169	4.144
11	0.260	0.443	0.697	0.876	1.214	1.363	1.796	2.201	2.718	3.106	4.025
12	0.259	0.442	0.695	0.873	1.209	1.356	1.782	2.179	2.681	3.055	3.930
13	0.259	0.441	0.694	0.870	1.204	1.350	1.771	2.160	2.650	3.012	3.852
14	0.258	0.440	0.692	0.868	1.200	1.345	1.761	2.145	2.624	2.977	3.787
15	0.258	0.439	0.691	0.866	1.197	1.341	1.753	2.131	2.602	2.947	3.733
16	0.258	0.439	0.690	0.865	1.194	1.337	1.746	2.120	2.583	2.921	3.686
17	0.257	0.438	0.689	0.863	1.191	1.333	1.740	2.110	2.567	2.898	3.646
18	0.257	0.438	0.688	0.862	1.189	1.330	1.734	2.101	2.552	2.878	3.610
19	0.257	0.438	0.688	0.861	1.187	1.328	1.729	2.093	2.539	2.861	3.579
20	0.257	0.437	0.687	0.860	1.185	1.325	1.725	2.086	2.528	2.845	3.552
21	0.257	0.437	0.686	0.859	1.183	1.323	1.721	2.080	2.518	2.831	3.527
22	0.256	0.437	0.686	0.858	1.182	1.321	1.717	2.074	2.508	2.819	3.505
23	0.256	0.436	0.685	0.858	1.180	1.319	1.714	2.069	2.500	2.807	3.485
24	0.256	0.436	0.685	0.857	1.179	1.318	1.711	2.064	2.492	2.797	3.467
25	0.256	0.436	0.684	0.856	1.178	1.316	1.708	2.060	2.485	2.787	3.450
26	0.256	0.436	0.684	0.856	1.177	1.315	1.706	2.056	2.479	2.779	3.435
27	0.256	0.435	0.684	0.855	1.176	1.314	1.703	2.052	2.473	2.771	3.421
28	0.256	0.435	0.683	0.855	1.175	1.313	1.701	2.048	2.467	2.763	3.408
29	0.256	0.435	0.683	0.854	1.174	1.311	1.699	2.045	2.462	2.756	3.396
30	0.256	0.435	0.683	0.854	1.173	1.310	1.697	2.042	2.457	2.750	3.385
35	0.255	0.434	0.682	0.852	1.170	1.306	1.690	2.030	2.438	2.724	3.340
40	0.255	0.434	0.681	0.851	1.167	1.303	1.684	2.021	2.423	2.704	3.307
45	0.255	0.434	0.680	0.850	1.165	1.301	1.679	2.014	2.412	2.690	3.281
50	0.255	0.433	0.679	0.849	1.164	1.299	1.676	2.009	2.403	2.678	3.261
55	0.255	0.433	0.679	0.848	1.163	1.297	1.673	2.004	2.396	2.668	3.245
60	0.254	0.433	0.679	0.848	1.162	1.296	1.671	2.000	2.390	2.660	3.232
$\infty$	0.253	0.431	0.674	0.842	1.150	1.282	1.645	1.960	2.326	2.576	3.090

# B Fourier and Laplace Tables



This appendix contains tables of Fourier and Laplace transforms.

## B.1 Laplace Transforms



table B.1 is a table with functions of time  $f(t)$  on the left and corresponding Laplace transforms  $L(s)$  on the right. Where applicable,  $s = \sigma + j\omega$  is the Laplace transform variable,  $T$  is the time-domain period,  $\omega_0 2\pi/T$  is the corresponding angular frequency,  $j = \sqrt{-1}$ ,  $a \in \mathbb{R}^+$ , and  $b, t_0 \in \mathbb{R}$  are constants.

Table B.1. Laplace transform identities.

function of time $t$	function of Laplace $s$
$a_1 f_1(t) + a_2 f_2(t)$	$a_1 F_1(s) + a_2 F_2(s)$
$f(t - t_0)$	$F(s)e^{-t_0 s}$
$f'(t)$	$sF(s) - f(0)$
$\frac{d^n f(t)}{dt^n}$	$s^n F(s) + s^{(n-1)} f(0) + s^{(n-2)} f'(0) + \dots + f^{(n-1)}(0)$
$\int_0^t f(\tau) d\tau$	$\frac{1}{s} F(s)$
$t f(t)$	$-F'(s)$
$f_1(t) * f_2(t) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t - \tau) d\tau$	$F_1(s) F_2(s)$
$\delta(t)$	1
$u_s(t)$	$1/s$
$u_r(t)$	$1/s^2$
$t^{n-1} / (n-1)!$	$1/s^n$
$e^{-at}$	$\frac{1}{s+a}$

$t e^{-at}$	$\frac{1}{(s+a)^2}$
$\frac{1}{(n-1)!} t^{n-1} e^{-at}$	$\frac{1}{(s+a)^n}$
$\frac{1}{a-b} (e^{at} - e^{bt})$	$\frac{1}{(s-a)(s-b)} \quad (a \neq b)$
$\frac{1}{a-b} (ae^{at} - be^{bt})$	$\frac{s}{(s-a)(s-b)} \quad (a \neq b)$
$\sin \omega t$	$\frac{\omega}{s^2 + \omega^2}$
$\cos \omega t$	$\frac{s}{s^2 + \omega^2}$
$e^{at} \sin \omega t$	$\frac{\omega}{(s-a)^2 + \omega^2}$
$e^{at} \cos \omega t$	$\frac{s-a}{(s-a)^2 + \omega^2}$

## B.2 Fourier Transforms

table B.2 is a table with functions of time  $f(t)$  on the left and corresponding Fourier transforms  $F(\omega)$  on the right. Where applicable,  $T$  is the time-domain period,  $\omega_0 2\pi/T$  is the corresponding angular frequency,  $j = \sqrt{-1}$ ,  $a \in \mathbb{R}^+$ , and  $b, t_0 \in \mathbb{R}$  are constants. Furthermore,  $f_e$  and  $f_o$  are even and odd functions of time, respectively, and it can be shown that any function  $f$  can be written as the sum  $f(t) = f_e(t) + f_o(t)$ . (Hsu 1970; appendix E)



Table B.2. Fourier transform identities.

function of time $t$	function of frequency $\omega$
$a_1 f_1(t) + a_2 f_2(t)$	$a_1 F_1(\omega) + a_2 F_2(\omega)$
$f(at)$	$\frac{1}{ a } F(\omega/a)$
$f(-t)$	$F(-\omega)$
$f(t - t_0)$	$F(\omega) e^{-j\omega t_0}$
$f(t) \cos \omega_0 t$	$\frac{1}{2} F(\omega - \omega_0) + \frac{1}{2} F(\omega + \omega_0)$
$f(t) \sin \omega_0 t$	$\frac{1}{j2} F(\omega - \omega_0) - \frac{1}{j2} F(\omega + \omega_0)$
$f_e(t)$	$\Re F(\omega)$
$f_o(t)$	$j \Im F(\omega)$
$F(t)$	$2\pi f(-\omega)$

$f'(t)$	$j\omega F(\omega)$
$\frac{d^n f(t)}{dt^n}$	$(j\omega)^n F(\omega)$
$\int_{-\infty}^t f(\tau) d\tau$	$\frac{1}{j\omega} F(\omega) + \pi F(0)\delta(\omega)$
$-jt f(t)$	$F'(\omega)$
$(-jt)^n f(t)$	$\frac{d^n F(\omega)}{d\omega^n}$
$f_1(t) * f_2(t) = \int_{-\infty}^{\infty} f_1(\tau) f_2(t - \tau) d\tau$	$F_1(\omega) F_2(\omega)$
$f_1(t) f_2(t)$	$\frac{1}{2\pi} F_1(\omega) * F_2(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F_1(\alpha) F_2(\omega - \alpha) d\alpha$
$e^{-at} u_s(t)$	$\frac{1}{j\omega + a}$
$e^{-a t }$	$\frac{2a}{a^2 + \omega^2}$
$e^{-at^2}$	$\sqrt{\pi/a} e^{-\omega^2/(4a)}$
1 for $ t  < a/2$ , else 0	$\frac{a \sin(a\omega/2)}{a\omega/2}$
$t e^{-at} u_s(t)$	$\frac{1}{(a + j\omega)^2}$
$\frac{t^{n-1}}{(n-1)!} e^{-at} u_s(t)$	$\frac{1}{(a + j\omega)^n}$
$\frac{1}{a^2 + t^2}$	$\frac{\pi}{a} e^{-a \omega }$
$\delta(t)$	1
$\delta(t - t_0)$	$e^{-j\omega t_0}$
$u_s(t)$	$\pi\delta(\omega) + \frac{1}{j\omega}$
$u_s(t - t_0)$	$\pi\delta(\omega) + \frac{1}{j\omega} e^{-j\omega t_0}$
1	$2\pi\delta(\omega)$
$t$	$2\pi j\delta'(\omega)$
$t^n$	$2\pi j^n \frac{d^n \delta(\omega)}{d\omega^n}$
$e^{j\omega_0 t}$	$2\pi\delta(\omega - \omega_0)$
$\cos \omega_0 t$	$\pi\delta(\omega - \omega_0) + \pi\delta(\omega + \omega_0)$
$\sin \omega_0 t$	$-j\pi\delta(\omega - \omega_0) + j\pi\delta(\omega + \omega_0)$
$u_s(t) \cos \omega_0 t$	$\frac{j\omega}{\omega_0^2 - \omega^2} + \frac{\pi}{2} \delta(\omega - \omega_0) + \frac{\pi}{2} \delta(\omega + \omega_0)$
$u_s(t) \sin \omega_0 t$	$\frac{\omega_0}{\omega_0^2 - \omega^2} + \frac{\pi}{2j} \delta(\omega - \omega_0) - \frac{\pi}{2j} \delta(\omega + \omega_0)$

$tu_s(t)$	$j\pi\delta'(\omega)-1/\omega^2$
$1/t$	$\pi j-2\pi ju_s(\omega)$
$1/t^n$	$\frac{(-j\omega)^{n-1}}{(n-1)!}(\pi j-2\pi ju_s(\omega))$
$\text{sgn } t$	$\frac{2}{j\omega}$
$\sum_{n=-\infty}^{\infty}\delta(t-nT)$	$\omega_0\sum_{n=-\infty}^{\infty}\delta(\omega-n\omega_0)$

---



# C Mathematics Reference



This appendix contains a reference for algebra, trigonometry, and other mathematical topics.

## C.1 Quadratic Forms



The solution to equations of the form  $ax^2 + bx + c = 0$  is

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \quad (\text{C.1})$$

### C.1.1 H

is accomplished by re-writing the quadratic formula in the form of the left-hand-side (LHS) of this equality, which describes factorization

$$x^2 + 2xh + h^2 = (x + h)^2. \quad (\text{C.2})$$

## C.2 Trigonometry

### C.2.1 Triangle Identities



With reference to figure C.1, the *law of sines* is

$$\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} \quad (\text{C.3})$$

and the *law of cosines* is

$$c^2 = a^2 + b^2 - 2ab \cos \gamma \quad (\text{C.4a})$$

$$b^2 = a^2 + c^2 - 2ac \cos \beta \quad (\text{C.4b})$$

$$a^2 = c^2 + b^2 - 2cb \cos \alpha \quad (\text{C.4c})$$

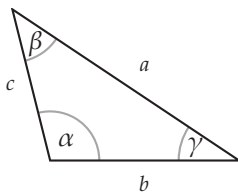


Figure C.1. Triangle for the law of sines and law of cosines.

### C.2.2 Reciprocal Identities

$$\csc u = \frac{1}{\sin u} \quad (\text{C.5a})$$

$$\sec u = \frac{1}{\cos u} \quad (\text{C.5b})$$

$$\cot u = \frac{1}{\tan u} \quad (\text{C.5c})$$

### C.2.3 Pythagorean Identities

$$1 = \sin^2 u + \cos^2 u \quad (\text{C.6a})$$

$$\sec^2 u = 1 + \tan^2 u \quad (\text{C.6b})$$

$$\csc^2 u = 1 + \cot^2 u \quad (\text{C.6c})$$

### C.2.4 Cofunction Identities

$$\sin\left(\frac{\pi}{2} - u\right) = \cos u \quad (\text{C.7a})$$

$$\cos\left(\frac{\pi}{2} - u\right) = \sin u \quad (\text{C.7b})$$

$$\tan\left(\frac{\pi}{2} - u\right) = \cot u \quad (\text{C.7c})$$

$$\csc\left(\frac{\pi}{2} - u\right) = \sec u \quad (\text{C.7d})$$

$$\sec\left(\frac{\pi}{2} - u\right) = \csc u \quad (\text{C.7e})$$

$$\cot\left(\frac{\pi}{2} - u\right) = \tan u \quad (\text{C.7f})$$

**C.2.5 Even-Odd Identities**

$$\sin(-u) = -\sin u \quad (\text{C.8a})$$

$$\cos(-u) = \cos u \quad (\text{C.8b})$$

$$\tan(-u) = -\tan u \quad (\text{C.8c})$$

**C.2.6 Sum-Difference Formulas (AM or Lock-In)**

$$\sin(u \pm v) = \sin u \cos v \pm \cos u \sin v \quad (\text{C.9a})$$

$$\cos(u \pm v) = \cos u \cos v \mp \sin u \sin v \quad (\text{C.9b})$$

$$\tan(u \pm v) = \frac{\tan u \pm \tan v}{1 \mp \tan u \tan v} \quad (\text{C.9c})$$

**C.2.7 Double Angle Formulas**

$$\sin(2u) = 2 \sin u \cos u \quad (\text{C.10a})$$

$$\cos(2u) = \cos^2 u - \sin^2 u \quad (\text{C.10b})$$

$$= 2 \cos^2 u - 1 \quad (\text{C.10c})$$

$$= 1 - 2 \sin^2 u \quad (\text{C.10d})$$

$$\tan(2u) = \frac{2 \tan u}{1 - \tan^2 u} \quad (\text{C.10e})$$

**C.2.8 Power-Reducing or Half-Angle Formulas**

$$\sin^2 u = \frac{1 - \cos(2u)}{2} \quad (\text{C.11a})$$

$$\cos^2 u = \frac{1 + \cos(2u)}{2} \quad (\text{C.11b})$$

$$\tan^2 u = \frac{1 - \cos(2u)}{1 + \cos(2u)} \quad (\text{C.11c})$$

**C.2.9 Sum-To-Product Formulas**

$$\sin u + \sin v = 2 \sin \frac{u+v}{2} \cos \frac{u-v}{2} \quad (\text{C.12a})$$

$$\sin u - \sin v = 2 \cos \frac{u+v}{2} \sin \frac{u-v}{2} \quad (\text{C.12b})$$

$$\cos u + \cos v = 2 \cos \frac{u+v}{2} \cos \frac{u-v}{2} \quad (\text{C.12c})$$

$$\cos u - \cos v = -2 \sin \frac{u+v}{2} \sin \frac{u-v}{2} \quad (\text{C.12d})$$

**C.2.10 Product-To-Sum Formulas**

$$\sin u \sin v = \frac{1}{2} [\cos(u-v) - \cos(u+v)] \quad (\text{C.13a})$$

$$\cos u \cos v = \frac{1}{2} [\cos(u-v) + \cos(u+v)] \quad (\text{C.13b})$$

$$\sin u \cos v = \frac{1}{2} [\sin(u+v) + \sin(u-v)] \quad (\text{C.13c})$$

$$\cos u \sin v = \frac{1}{2} [\sin(u+v) - \sin(u-v)] \quad (\text{C.13d})$$

**C.2.11 Two-To-One Formulas**

$$A \sin u + B \cos u = C \sin(u + \phi) \quad (\text{C.14a})$$

$$= C \cos(u + \psi) \text{ where} \quad (\text{C.14b})$$

$$C = \sqrt{A^2 + B^2} \quad (\text{C.14c})$$

$$\phi = \arctan \frac{B}{A} \quad (\text{C.14d})$$

$$\psi = -\arctan \frac{A}{B} \quad (\text{C.14e})$$

### C.3 Matrix Inverses

This is a guide to inverting  $1 \times 1$ ,  $2 \times 2$ , and  $n \times n$  matrices.

Let  $A$  be the  $1 \times 1$  matrix

$$A = [a].$$

The inverse is simply the reciprocal:

$$A^{-1} = [1/a].$$

Let  $B$  be the  $2 \times 2$  matrix

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}.$$

It can be shown that the inverse follows a simple pattern:

$$\begin{aligned} B^{-1} &= \frac{1}{\det B} \begin{bmatrix} b_{22} & -b_{12} \\ -b_{21} & b_{11} \end{bmatrix} \\ &= \frac{1}{b_{11}b_{22} - b_{12}b_{21}} \begin{bmatrix} b_{22} & -b_{12} \\ -b_{21} & b_{11} \end{bmatrix}. \end{aligned}$$

Let  $C$  be an  $n \times n$  matrix. It can be shown that its inverse is

$$C^{-1} = \frac{1}{\det C} \text{adj } C,$$

where  $\text{adj}$  is the **adjoint** of  $C$ .

### C.4 Euler's Formulas

**Euler's formula** is our bridge back-and forth between trigonometric forms ( $\cos \theta$  and  $\sin \theta$ ) and complex exponential form ( $e^{j\theta}$ ):

$$e^{j\theta} = \cos \theta + j \sin \theta. \quad (\text{C.15})$$

Here are a few useful identities implied by Euler's formula.

$$e^{-j\theta} = \cos \theta - j \sin \theta \quad (\text{C.16a})$$

$$\cos \theta = \Re(e^{j\theta}) \quad (\text{C.16b})$$

$$= \frac{1}{2} (e^{j\theta} + e^{-j\theta}) \quad (\text{C.16c})$$

$$\sin \theta = \Im(e^{j\theta}) \quad (\text{C.16d})$$

$$= \frac{1}{j2} (e^{j\theta} - e^{-j\theta}). \quad (\text{C.16e})$$



## C.5 Laplace Transforms

The definition of the one-side Laplace and inverse Laplace transforms follow.



### Definition C.1: Laplace transforms (one-sided)

Laplace transform  $\mathcal{L}$ :

$$\mathcal{L}(y(t)) = Y(s) = \int_0^{\infty} y(t)e^{-st} dt. \quad (\text{C.17})$$

Inverse Laplace transform  $\mathcal{L}^{-1}$ :

$$\mathcal{L}^{-1}(Y(s)) = y(t) = \frac{1}{2\pi j} \int_{\sigma-j\infty}^{\sigma+j\infty} Y(s)e^{st} ds. \quad (\text{C.18})$$

See table B.1 for a list of properties and common transforms.

## Bibliography

- Ash, Robert B. 2008. *Basic Probability Theory*. Dover Publications, Inc.
- Bagaria, Joan. 2019. "Set Theory." In *The Stanford Encyclopedia of Philosophy*, Fall 2019, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Baghrmian, Maria, and J. Adam Carter. 2019. "Relativism." In *The Stanford Encyclopedia of Philosophy*, Winter 2019, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Barker, Stephen, and Mark Jago. 2012. "Being Positive About Negative Facts." *Philosophy and Phenomenological Research* 85 (1): 117–138. <https://doi.org/10.1111/j.1933-1592.2010.00479.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1933-1592.2010.00479.x>. <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1933-1592.2010.00479.x>.
- Barzilai, Jonathan, and Jonathan M. Borwein. 1988. "Two-Point Step Size Gradient Methods." *IMA Journal of Numerical Analysis* 8, no. 1 (January): 141–148. This includes an innovative line search method., <https://doi.org/10.1093/imanum/8.1.141>. <https://doi.org/10.1093/imanum/8.1.141>.
- Biletzki, Anat, and Anat Matar. 2018. "Ludwig Wittgenstein." In *The Stanford Encyclopedia of Philosophy*, Summer 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University. An introduction to Wittgenstein and his thought.
- Bove, Antonio, F. (Ferruccio) Colombini, and Daniele Del Santo. 2006. *Phase space analysis of partial differential equations* [in eng]. Progress in nonlinear differential equations and their applications ; v. 69. Birkhäuser.
- Brogan, William L. 1991. *Modern Control Theory*. Third. Prentice Hall.
- Bullo, Francesco, and Andrew D. Lewis. 2005. *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*. Edited by J.E. Marsden, L. Sirovich, and M. Golubitsky. Springer.
- Choukchou-Braham, A., B. Cherki, M. Djemäi, and K. Busawon. 2013. *Analysis and Control of Underactuated Mechanical Systems*. SpringerLink : Bücher. Springer International Publishing. <https://link.springer.com/content/pdf/bbm%3A978-3-319-02636-7%2F1.pdf>.
- Ciesielski, K. 1997. *Set Theory for the Working Mathematician*. London Mathematical Society Student Texts. Cambridge University Press. A readable introduction to set theory.

- David, Marian. 2016. "The Correspondence Theory of Truth." In *The Stanford Encyclopedia of Philosophy*, Fall 2016, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University. A detailed overview of the correspondence theory of truth.
- Dolby, David. 2016. "Wittgenstein on Truth." Chap. 27 in *A Companion to Wittgenstein*, 433–442. John Wiley & Sons, Ltd. <https://doi.org/10.1002/9781118884607.ch27>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118884607.ch27>. <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118884607.ch27>.
- Enderton, H.B. 1977. *Elements of Set Theory*. Elsevier Science. A gentle introduction to set theory and mathematical reasoning—a great place to start.
- Feynman, Richard P., Robert B. Leighton, and Matthew Sands. 2010. *The Feynman Lectures on Physics*. New Millennium. Perseus Basic Books. <http://www.feynmanlectures.caltech.edu/index.html>.
- Glanzberg, Michael. 2018. "Truth." In *The Stanford Encyclopedia of Philosophy*, Fall 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Glock, Hans Johann. 2006. "Truth in the Tractatus." *Synthese* 148, no. 2 (January): 345–368. <https://doi.org/10.1007/s11229-004-6226-2>. <https://doi.org/10.1007/s11229-004-6226-2>.
- Gómez-Torrente, Mario. 2019. "Alfred Tarski." In *The Stanford Encyclopedia of Philosophy*, Spring 2019, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Guyer, Paul, and Rolf-Peter Horstmann. 2018. "Idealism." In *The Stanford Encyclopedia of Philosophy*, Winter 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Haberman, R. 2018. *Applied Partial Differential Equations with Fourier Series and Boundary Value Problems (Classic Version)*. Pearson Modern Classics for Advanced Mathematics. Pearson Education Canada.
- Hegel, G.W.F., and A.V. Miller. 1998. *Phenomenology of Spirit*. Motilal Banarsidass.
- Hibbeler, R. 2015. *Engineering Mechanics: Statics and Dynamics, Student Value Edition*. Pearson Education.
- Hodges, Wilfrid. 2018a. "Model Theory." In *The Stanford Encyclopedia of Philosophy*, Fall 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Hodges, Wilfrid. 2018b. "Tarski's Truth Definitions." In *The Stanford Encyclopedia of Philosophy*, Fall 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Hsu, Hwei P. 1970. *Fourier Analysis*. Simon / Schuster. <http://gen.lib.rus.ec/book/index.php?md5=24D6068CC9DEC5E41EC67CC79FD78912>.
- Hylton, Peter, and Gary Kemp. 2019. "Willard Van Orman Quine." In *The Stanford Encyclopedia of Philosophy*, Spring 2019, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Jaynes, E.T., E.T.J. Jaynes, G.L. Bretthorst, and Cambridge University Press. 2003. *Probability Theory: The Logic of Science*. Cambridge University Press. An excellent and comprehensive introduction to probability theory.
- Kant, I., P. Guyer, and A.W. Wood. 1999. *Critique of Pure Reason*. The Cambridge Edition of the Works of Immanuel Kant. Cambridge University Press.



- Kennedy, Juliette. 2018. "Kurt Gödel." In *The Stanford Encyclopedia of Philosophy*, Winter 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Khrentzos, Drew. 2016. "Challenges to Metaphysical Realism." In *The Stanford Encyclopedia of Philosophy*, Winter 2016, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Klein, Peter. 2015. "Skepticism." In *The Stanford Encyclopedia of Philosophy*, Summer 2015, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Kline, M. 1982. *Mathematics: The Loss of Certainty*. A Galaxy book. Oxford University Press. A detailed account of the "illogical" development of mathematics and an exposition of its therefore remarkable utility in describing the world., <https://books.google.com/books?id=RNwnUL33epsC>.
- Kolk, W. Richard, and Robert A. Lerman. 1993. *Nonlinear System Dynamics*. 1st ed. Springer US. <http://gen.lib.rus.ec/book/index.php?md5=589b0829bfe0fda7d4b52f0a09122064>.
- Kot, M. 2014. *A First Course in the Calculus of Variations*. Student Mathematical Library. American Mathematical Society.
- Kreyszig, Erwin. 2011. *Advanced Engineering Mathematics*. 10<sup>th</sup>. John Wiley & Sons, Limited. The authoritative resource for engineering mathematics. It includes detailed accounts of probability, statistics, vector calculus, linear algebra, fourier analysis, ordinary and partial differential equations, and complex analysis. It also includes several other topics with varying degrees of depth. Overall, it is the best place to start when seeking mathematical guidance.
- Lee, John M. 2012. *Introduction to Smooth Manifolds*. Second. Vol. 218. Graduate Texts in Mathematics. Springer.
- Legg, Catherine, and Christopher Hookway. 2019. "Pragmatism." In *The Stanford Encyclopedia of Philosophy*, Spring 2019, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University. An introductory article on the philosophical movement "pragmatism." It includes an important clarification of the pragmatic slogan, "truth is the end of inquiry."
- O'Reilly, O. M. 2019. *Engineering Dynamics: A Primer*. Springer International Publishing.
- Raatikainen, Panu. 2018. "Gödel's Incompleteness Theorems." In *The Stanford Encyclopedia of Philosophy*, Fall 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University. A thorough and contemporary description of Gödel's incompleteness theorems, which have significant implications for the foundations and function of mathematics and mathematical truth.
- Redding, Paul. 2018. "Georg Wilhelm Friedrich Hegel." In *The Stanford Encyclopedia of Philosophy*, Summer 2018, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Schey, H.M. 2005. *Div, Grad, Curl, and All that: An Informal Text on Vector Calculus*. W.W. Norton. <https://books.google.com/books?id=sembQgAACAAJ>.
- Shields, Christopher. 2016. "Aristotle." In *The Stanford Encyclopedia of Philosophy*, Winter 2016, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Skiena, Steven S. 2001. *Calculated Bets: Computers, Gambling, and Mathematical Modeling to Win*. Outlooks. Cambridge University Press. This includes a lucid section on probability versus

- statistics, also available here: <https://www3.cs.stonybrook.edu/~skiena/jaialai/excerpts/no-de12.html>, <https://doi.org/10.1017/CBO9780511547089>.
- Smith, George. 2008. "Isaac Newton." In *The Stanford Encyclopedia of Philosophy*, Fall 2008, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Stoljar, Daniel, and Nic Damnjanovic. 2014. "The Deflationary Theory of Truth." In *The Stanford Encyclopedia of Philosophy*, Fall 2014, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Strauss, W.A. 2007. *Partial Differential Equations: An Introduction*. Wiley. A thorough and yet relatively compact introduction.
- Strogatz, S.H., and M. Dichter. 2016. *Nonlinear Dynamics and Chaos*. Second. Studies in Nonlinearity. Avalon Publishing.
- Textor, Mark. 2016. "States of Affairs." In *The Stanford Encyclopedia of Philosophy*, Winter 2016, edited by Edward N. Zalta. Metaphysics Research Lab, Stanford University.
- Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, et al. 2019. "SciPy 1.0—Fundamental Algorithms for Scientific Computing in Python." *arXiv e-prints* (July): arXiv:1907.10121. arXiv:1907.10121 [cs.MS].
- Wikipedia. 2019a. *Algebra* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 26-October-2019].
- Wikipedia. 2019b. *Carl Friedrich Gauss* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 26-October-2019].
- Wikipedia. 2019c. *Euclid* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 26-October-2019].
- Wikipedia. 2019d. *First-order logic* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 29-October-2019].
- Wikipedia. 2019e. *Leonhard Euler* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 26-October-2019].
- Wikipedia. 2019f. *Linguistic turn* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 23-October-2019]. Hey, we all do it.
- Wikipedia. 2019g. *Probability space* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 31-October-2019].
- Wikipedia. 2019h. *Propositional calculus* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 29-October-2019].
- Wikipedia. 2019i. *Quaternion* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 26-October-2019].
- Wikipedia. 2019j. *Set-builder notation* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 29-October-2019].
- Wikipedia. 2019k. *William Rowan Hamilton* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 26-October-2019].
- Wittgenstein, L., P.M.S. Hacker, and J. Schulte. 2010. *Philosophical Investigations*. Wiley.
- Wittgenstein, Ludwig. 1922. *Tractatus Logico-Philosophicus*. Project Gutenberg. Edited by C.), familyi=C., given=K. Ogden, giveni=. O. International Library of Psychology Philosophy

and Scientific Method. Kegan Paul, Trench, Trubner & Co., Ltd. A brilliant work on what is possible to express in language—and what is not. As Wittgenstein puts it, “What can be said at all can be said clearly; and whereof one cannot speak thereof one must be silent.”

Žižek, Slavoj. 2012. *Less Than Nothing: Hegel and the Shadow of Dialectical Materialism*. Verso. This is one of the most interesting presentations of Hegel and Lacan by one of the most exciting contemporary philosophers.



## **Contributors**

**Associate Professor Rico A. R. Picone**  
Department of Mechanical Engineering  
Saint Martin's University  
Lacey, Washington, USA