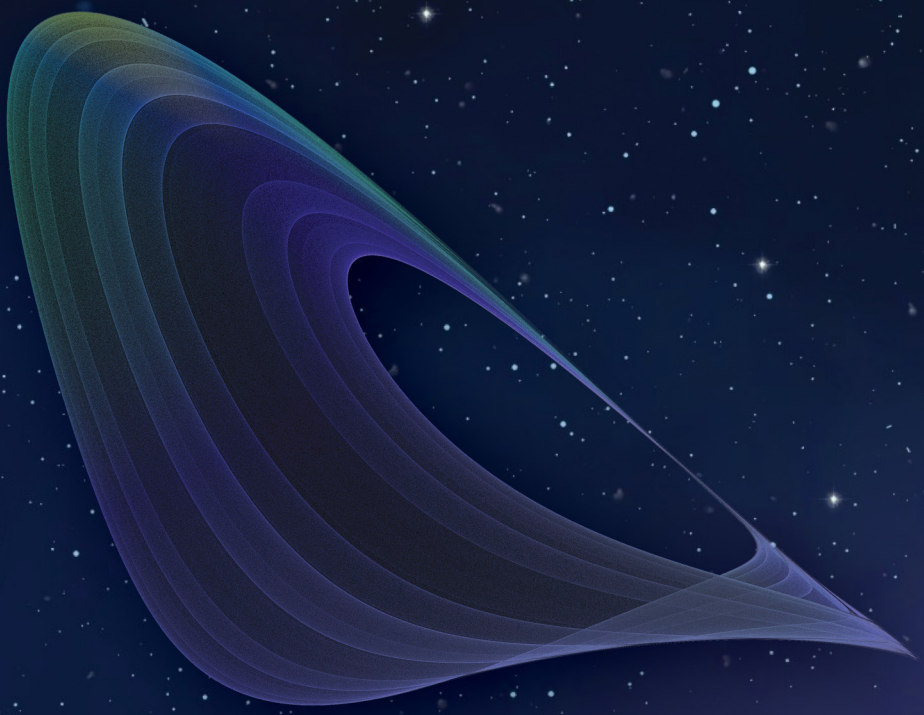
Classical Mechanics

A Computational Approach with Examples Using Mathematica and Python



Christopher W. Kulp
Vasilis Pagonis

CRC Press
Taylor & Francis Group

Classical Mechanics

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Classical Mechanics

A Computational Approach with Examples Using Mathematica and Python

Christopher W. Kulp
Vasilis Pagonis

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

First edition published 2021
by CRC Press
6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742

and by CRC Press
2 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

Library of Congress Cataloging-in-Publication Data
Names: Kulp, Christopher W., author. | Pagonis, Vasilis, author.
Title: Classical mechanics : a computational approach, with examples using mathematica and python / Christopher W. Kulp, Vasilis Pagonis.
Description: Boca Raton : CRC Press, 2020. | Includes bibliographical references and index.
Identifiers: LCCN 2020021514 | ISBN 9781138495289 (paperback) | ISBN 9781138495173 (hardback) | ISBN 9781351024389 (ebook)
Subjects: LCSH: Mechanics. | Mechanics--Data processing. | Python (Computer program language) | Mathematica (Computer file)
Classification: LCC QC127 .K85 2020 | DDC 531.0285/53--dc23
LC record available at https://lccn.loc.gov/2020021514

ISBN: 978-1-138-49517-3 (hbk)
ISBN: 978-1-138-49528-9 (pbk)
ISBN: 978-1-351-02438-9 (ebk)

Typeset in LMRoman
by Nova Techset Private Limited, Bengaluru & Chennai, India

Visit the [eResources]: www.routledge.com/9781138495289

Chris dedicates this book to his wife Gail, mother Linda, and his late father, Chester. Without their support, this book would not have been possible.

Vasilis dedicates this book to his wife, Mary Jo Boylan, and to his students at McDaniel College.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Contents

Preface

WHY DID WE WRITE THIS BOOK?

The use of computers to solve problems has become a fundamental and critical skill in all scientific fields. This book is our attempt to merge instruction in computation and classical mechanics into a single presentation, where computer programming and computer algebra systems are thought of as simply one more tool to use for problem solving.

Alongside instruction in classical mechanics, we provide instruction in using computers to solve physics problems. Where relevant, we discuss how various algorithms work, such as the fourth-order Runge Kutta method, by providing a type of "pseudo code" that outlines how algorithms can be implemented. Our goal was to write a book that provides concrete examples of how to implement computer programming to solve physics problems. In order to provide such examples, we use Python and Mathematica. However, we believe that this book can be used with any programming language. For example, if a student wants to solve a differential equation using the language R with the Euler method described in Chapter 2, then the student could follow our pseudo code and create their own algorithm, by performing an internet search for "Euler method R," or searching "how to numerically solve a differential equation in R."

Why did we choose to include both Python *and* Mathematica? Physicists need many different computational tools to solve problems. Python is a traditional scripted programming language that many students learn in computer science courses. In addition, there are many freely available websites which provide instruction on Python. Python is easy to learn, free, and has become a regularly-used programming language in a variety of fields. In addition, when libraries such as Numpy are used, Python code executes fast, and algorithms written in Python can quickly and efficiently solve the problems presented in this book.

Now you may wonder if Python is such a powerful tool, why also include Mathematica? Mathematica is a powerful computer algebra system. Like Python it is easy to learn, but Mathematica can handle complex algebraic manipulations better than Python. Furthermore, Mathematica often requires fewer lines of coding than Python, because of its rich set of commands. A command like *NDSolve* in Mathematica runs in the background many lines of code, which the user need not see or access. By presenting a variety of tools, we are hoping that students will be able to choose the best tool for solving a particular problem. Sometimes a computer algebra system is the better tool, while other times writing your own script in Python is the better choice. We discuss the differences in computer programming languages more thoroughly in Chapter 1.

As we wrote the programs in Python or Mathematica to solve the physics problems, we tended towards clarity over efficiency. You will certainly find more efficient ways of solving the various example problems in this book. However, we believe that it is much more beneficial for the student to include a few more lines of code for the purpose of clarity, rather than trying to combine multiple lines for a more elegant algorithm. Solving the problems with algorithms that they develop themselves is a beneficial exercise for students and is highly recommended. One learns physics by solving problems. The authors encourage the reader to examine other texts such as

[de Lange and Pierrus(2010), Taylor(2005), Morin(2008), Thornton and Marion(2004)] for additional problems. These texts approach classical mechanics in different ways, and it is beneficial to see multiple presentations of the same topic.

Technology has often disrupted the status quo on how things are done. When pocket calculators were first introduced, people were afraid that it would ruin students' ability to do math. However a 2003 study found that "students' operational skills and problem solving skills improved when calculators were an integral part of instruction" [Ellington(2003)]. With further advances in technology, one can ask whether the use of computer algebra systems (CAS) will damage a student's ability to solve physics problems. A 2008 dissertation [Tokpah(2008)] showed that "students using CAS tend to perform better than students taught using non-CAS instruction" when learning mathematics. However, we believe that the use of computers to assist, for example, in algebraic manipulations, will greatly benefit physics students. Long and involved algebraic manipulation is rarely insightful in terms of understanding the underlying physics. When a student is allowed to offload tedious algebraic manipulations to a computer, this student can then focus on higher-level mathematics, such as analyzing the problem using limits, in order to better develop a physical intuition of what the equations are describing.

This book is for students who are taking a semester of classical mechanics, following a full course in introductory physics. The prerequisites for this book are two semesters of introductory physics and two semesters of calculus. Despite the title of the book, a semester of computer science is not needed (but would be helpful) to start reading this book.

A NOTE TO THE STUDENTS

To the students using this book, we say, "Don't feel overwhelmed!" We understand physics is hard, and now we are asking you to learn physics, mathematics, *and* programming. Just like you have learned math along the way of learning physics, you can learn programming that way, too. We would argue that learning programming, at least for solving physics problems, is easier than learning the math! Programming is challenging at first. The more you do it, the better you will be at it and the more fun your experience will be.

Programming will change the way you think about problems, since writing a program requires you to think procedurally. This will certainly improve your problem-solving skills in your other physics and math classes. Don't hesitate to look online for help. Websites like stackexchange.com will become your best friend. As you continue to solve problems and look for help online, you will notice that some of the programming will stick with you, and you will find yourself searching less online, as you solve more and more problems. When learning a new programming language, it is very useful to first do some basic reading on syntax, then jump right into solving problems, by using websites like Stack Exchange for assistance. As you solve more programming problems, you will be learning an invaluable skill that will serve you in many ways in your development as a scientist! All of the code that appears in this book can be downloaded at www.routledge.com/9781138495289. We encourage you to download and modify the code. It will help you learn how to use both Python and Mathematica. We recommend the Anaconda Python distribution (www.anaconda.com) to run the Python programs in this book.

A NOTE TO INSTRUCTORS

One of our motivations for writing this book was to better prepare our students for the large variety of careers that they pursue after graduating from a physics program. We have found that after graduation, our students are increasingly taking on careers where computation is a critical element. Our students often take at least one computer science course.

However, they were not always making the connection of how to apply the programming skills they learned in computer science for the purpose of solving problems in physics. Part of the reason is because traditional physics courses are still focused on closed-form solutions and, when computation is used, it is generally focused on using computer algebra systems to perform complicated integrals, with the occasional numerical solution of a differential equation thrown in for good measure.

We believe that it is time for computation to be more closely integrated into the physics curriculum. Doing so clearly demonstrates to students how to use computation to solve problems, a skill that many of them will find critical whether they go to graduate school, or enter into careers in government or industry after graduation. We should remember that the majority of physics students (65% at the time of this writing [Supiano(2018)]) do not go to graduate programs in physics or astronomy. Of the graduates who work in the private sector, 77% work either in engineering, computer or information systems, or non-science fields that regularly solve technical problems [Supiano(2018)]. Teaching students to use computation to solve physics problems will provide them a transferable skill that they can apply in their future careers, even if they never need to find a Lagrangian after taking their classical mechanics course.

Of course including computation in a classical mechanics course (or any physics course) comes at a cost, but we believe the payoff is well worth the cost. Every minute spent on the instruction of programming and computation is one less minute that can be spent on the instruction of topics like conservation laws, Hamiltonians, etc. In addition, by using this textbook the student may be in a position where he/she needs to learn computer programming, the use of a computer algebra system, *and* classical mechanics. This can be a challenging position to be in as a student. However, today's online resources can allow students to more quickly pick up computing skills.

A quick online search will result in sites that provide online courses in Python programming. However, even simple web searches such as, "How do I integrate an equation in Mathematica?" can provide the student enough instruction for solving problems in both Python and Mathematica. We find that the best method of learning how to use computational tools is by actually solving problems with them. Physics students often learn mathematics along with the physics topics, in order to solve physics problems. The same can be true for programming.

At this point, you might be wondering about the trade-off in content? Both of the authors teach one-semester–long classical mechanics courses and are aware of sacrifices that need to be made when introducing new material in a well-established course like classical mechanics. We did our best to include in this book all of the major topics traditionally covered in a classical mechanics course. We believe we were successful in that endeavor. This book can be used like any other classical mechanics book, such as the classic *Classical Dynamics of Particles and Systems* by Thornton and Marion, and the chapters are in fact structured like many other texts in the field.

An instructor can skip much of the discussion of computation and use this book like any other. However, the inclusion of the computation allows the students to explore problems that are difficult, or inaccessible without computation. It may be the case that in a one-semester course, the instructor may need to be more selective in the physics topics covered. We believe the exchange is worthwhile. The payoff is the ability to explore problems that are not solvable in closed-form. Many real-world problems that students will encounter are not solvable in closed form. Furthermore, the aforementioned transferable skill of using computation to solve problems is an additional payoff that will serve students well for their entire career.

ACKNOWLEDGMENTS

Finally, we want to thank several people who have helped make this book happen.We thank Kirsten Barr, Rebecca Davies, and Shashi Kumar at CRC Press for all of their help in the preparation of this book. We also extend our heart felt thanks to Mary Jo Boylan and Maryam Esmat for their hard work and time they have put into this book. Mary Jo typed the author's hand written solutions for the instructor's manual. She put in countless hours preparing the solution manual to this book. We would have never made our deadline without her! We very much appreciate her efforts. Maryam Esmat helped us proofread our book and provided comments from a student's perspective. Her detailed comments on each chapter were invaluable to the development of this book. We believe the book is much better because of Maryam's efforts. We thank her for all of her time reading and rereading each chapter!

CHAPTER 1

The Foundations of Motion and Computation

So, you have decided to—or are required to—learn the subject of classical mechanics. But, what is *classical mechanics*? Is it fixing old cars? No, but a knowledge of classical mechanics will help you understand how your car works! To understand the term, *classical mechanics* let us first understand the term *classical physics*. Classical physics are the fields of physics that don't involve either quantum theory or the theory of relativity. *Mechanics* is the branch of physics that deals with the actions of forces on an object that involve motion. So, classical mechanics involves the study of forces on objects that are well-described without using quantum theory and whose motion is nonrelativistic (i.e., cases where relativity is not needed to correctly model the motion). In other words, classical mechanics is the physics of your day-to-day world! An understanding of classical mechanics will help you understand how to build roller coasters, merry-go-rounds, and airplanes. Our knowledge of classical mechanics also allows us to make predictions on the motion of objects like comets and punted footballs. The world you interact with on a daily basis is generally the world of classical mechanics.

At first it might seem that classical mechanics is a dusty old subject that you need to learn in order to get to the "interesting stuff" like quantum mechanics. While the field of classical mechanics is one of the older subjects in physics, it is certainly not dusty! In fact, there is a lot of intriguing current research done in classical systems. For example, in Chapter 13 we will explore the field of nonlinear systems, which occur in in many of the natural and social sciences, not only in physics, and they display interesting types of behaviors including *chaos*.

1.1 THE WORLD OF PHYSICS

It is helpful to break up the field of physics into different branches. It should be noted that such divisions are largely arbitrary but useful human constructs that help us understand the world around us. As we sometimes tell students: Mother Nature doesn't care if a particular system is a thermodynamics problem or a classical mechanics problem; humans make those distinctions in order to better understand how to describe and model the system. In fact, most real-world problems involve multiple fields of physics. So with that in mind, we can loosely break up physics into the following branches:

- *Classical Mechanics* deals with how forces cause motion in classical systems.

- *Thermodynamics* deals with relationships between all forms of energy; often the focus is on heat and its relationship with other forms of energy.
- *Electromagnetism* deals with the interaction of electrically-charged particles using the concepts of electric and magnetic fields.
- *Statistical Mechanics* deals with understanding how macroscopic properties, such as temperature and pressure, emerge from a large number of particles that make up the system.
- *Relativity* deals with the dependence of physical phenomenon on the relative motion between the observer and the observed. Physicists often consider three cases of relativity: Galilean relativity (which falls under the category of classical physics), Einstein's theory of special relativity, and Einstein's theory of general relativity. *Special relativity* focuses on the dependencies between inertial frames of reference while *general relativity*, a generalization of special relativity, takes into account noninertial frames.
- *Quantum Mechanics* deals with the interactions between subatomic particles and between subatomic particles and radiation.

Classical mechanics is one of the first topics learned by a physics student because it deals with the more intuitive concepts of force and motion. It provides tools for not just describing motion but also predicting motion. One of the central ideas of classical mechanics is that if one knows the position and velocity of a particle at time, t, then one can find the particle's position and velocity at any point in the past or future (except for the case of chaotic systems, which we will deal with later). The ability to make predictions is critical in science, hence, classical mechanics provides a powerful set of tools, which are widely applicable to many different types of systems.

As mentioned earlier, almost all real-world problems involve more than one branch of physics, and classical mechanics is often an important element to those problems. For example, one might want to know the motion of a charged particle in a magnetic field. The first step would be to find the force on the particle from the magnetic field using formulas from electromagnetism, then the position and velocity as a function of time can be found using Newton's second law (a formula from classical mechanics). Although this example is simple, it illustrates the point that problems rarely involve only one branch of physics. Even if one is working with a system in which quantum theory must be included, *semiclassical* approaches, where part of the problem is treated classically, are sometimes quite useful.

It is common for classical mechanics to take a central role in many physics problems because of the question it addresses.

The "Fundamental Question" of Classical Mechanics

Given the forces acting on an object, what is the resulting motion of that object?

Hopefully, you understand how classical mechanics fits in with the other branches and its central importance to the larger field of physics. In the rest of this chapter, we will lay down the important foundations for classical mechanics, starting with the next section where we will explore the basic assumptions of classical mechanics.

1.2 THE BASICS OF CLASSICAL MECHANICS

When thinking about any discipline of science, first ask what are the basic foundations of the field? Sometimes these foundations are assumptions. For classical mechanics we can start with space and time:

- *Space* serves as a background in which physical processes occur. Unlike in the theory of general relativity, space has no effect on the behavior of physical systems. Furthermore, length measurements are the same for every observer, unlike in the theory of special relativity.
- *Time* progresses at the same rate for every observer (unlike in the theories of special and general relativity).

Classical mechanics addresses the problem of predicting an object's motion given the forces acting on the object. For now, we will consider the object to be a point particle. The advantage of working with point particles is that they have no size and no internal dynamics. In addition, point particles do not rotate nor do they deform, further simplifying their dynamics. It turns out that treating objects as point particles can be a very good approximation for describing translational motion. In later chapters, we will study the physics of extended bodies and rotational dynamics, where we will no longer restrict ourselves to working with point particles.

1.2.1 The Basic Descriptors of Motion

In order to describe the translational motion of a particle, we need three quantities that we will call the basic descriptors of motion: *position*, *velocity*, and *acceleration*. For rotational motion, we will need three additional quantities: *angular position, angular velocity,* and *angular acceleration,* and we will return to those later in the book.

1.2.1.1 Position and Displacement

The position of a particle is the location of the particle with respect to an origin and is measured in meters. The meter is defined as the distance light travels in 1/299,792,458 seconds. Note that the second was defined in 1967 at the 13th meeting of the International Committee on Weights and Measures. At that meeting the following definition was adopted: "The second is the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom."

The position of the particle is typically described by using a vector whose components consist of the particle's distance from the origin along three perpendicular axes. For example, in Cartesian coordinates the location of a particle can be described using:

$$\mathbf{r} = x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}} \tag{1.2.1}$$

where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ are the Cartesian unit vectors along the x, y, and z axes, respectively. The variables x, y, and z in (1.2.1) give the distance between the particle and the origin along each axis and are called the *components* of the vector. Also note that in (1.2.1): $\mathbf{r} = \mathbf{r}(t)$, $x = x(t)$, $y = y(t)$, and $z = z(t)$. In this book, vectors are denoted by bold font. In Chapter 3, we will discuss vector quantities in more detail.

The vector in (1.2.1) tells us that in order to get to the location of the particle, $\mathbf{r}$, one needs to move a distance x along the x-axis (as denoted by $\hat{\mathbf{i}}$), turn and move a distance y along the y-axis (as denoted by $\hat{\mathbf{j}}$), and then turn and move a distance z along the z-axis

(as denoted by $\hat{\mathbf{k}}$). The notation used in this book is that a hat (the ˆ symbol) represents a *unit vector*, a vector of length one. Hence, we can think of $\mathbf{r}$ as the sum of three vectors: $x\hat{\mathbf{i}}$, $y\hat{\mathbf{j}}$, and $z\hat{\mathbf{k}}$, each representing a *displacement* from the origin along one of the axes.

The displacement of the particle, $\Delta\mathbf{r} = \mathbf{r} - \mathbf{r}_0$, is the change of the particle's position from the position, $\mathbf{r}_0$, to the position, $\mathbf{r}$. The displacement is found by subtracting the two vectors $\mathbf{r}_0$ and $\mathbf{r}$, component by component,

$$\Delta\mathbf{r} = \mathbf{r} - \mathbf{r}_0 = (x - x_0)\hat{\mathbf{i}} + (y - y_0)\hat{\mathbf{j}} + (z - z_0)\hat{\mathbf{k}} \tag{1.2.2}$$

You may recall that a vector quantity contains information about both magnitude (amount) and direction. In this case, the vector $\Delta\mathbf{r}$ tells us how far (magnitude) and in which direction the particle traveled.

You may also see the unit vectors in the form $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$, sometimes used to clarify that one is working in the Cartesian coordinate system. In addition, sometimes it will be easier to use more generic notation for vectors where r_1, r_2, and r_3 are used instead of the components x, y, and z and the unit vectors $\hat{\mathbf{e}}_1$, $\hat{\mathbf{e}}_2$, and $\hat{\mathbf{e}}_3$ are used instead of $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$. This generic notation allows for a more compact method of writing vectors:

$$\mathbf{r} = r_1\hat{\mathbf{e}}_1 + r_2\hat{\mathbf{e}}_2 + r_3\hat{\mathbf{e}}_3 = \sum_{i=1}^{3} r_i\hat{\mathbf{e}}_i \tag{1.2.3}$$

There are other coordinate systems which we will explore later in this book. Each coordinate system will have its own components and unit vectors, however, the basic idea of position and displacement will be the same.

1.2.1.2 Velocity

The velocity $\mathbf{v}$ of a particle is the particle's displacement (change of position) per unit time and is measured in meters per second (m/s). The instantaneous velocity is found by computing the time derivative of the position vector:

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} = \lim_{\Delta t \to 0} \frac{\Delta\mathbf{r}}{\Delta t} \tag{1.2.4}$$

where $\Delta\mathbf{r} = \mathbf{r}(t + \Delta t) - \mathbf{r}(t)$. All of the vectors we will come across here are differentiable, and the limits will exist. Furthermore, the derivative in (1.2.4) will behave like derivatives you have encountered before. Therefore, we can use the standard derivative rules:

$$\frac{d}{dt}(\mathbf{r}_1 + \mathbf{r}_2) = \frac{d\mathbf{r}_1}{dt} + \frac{d\mathbf{r}_2}{dt} \tag{1.2.5}$$

$$\frac{d}{dt}(c\mathbf{r}) = \mathbf{r}\frac{dc}{dt} + c\frac{d\mathbf{r}}{dt} \tag{1.2.6}$$

where c is a scalar function of time. The rules in (1.2.5) and (1.2.6) allow us to distribute the derivative to each vector component. In addition, if the unit vectors are constant, like they are in Cartesian coordinates (but not in others!), then we can compute the velocity by:

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} \tag{1.2.7}$$

$$= \frac{d}{dt}\left(x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}}\right) \tag{1.2.8}$$

$$= \frac{dx}{dt}\hat{\mathbf{i}} + \frac{dy}{dt}\hat{\mathbf{j}} + \frac{dz}{dt}\hat{\mathbf{k}} \tag{1.2.9}$$

or: $\mathbf{v} = v_x\hat{\mathbf{i}} + v_y\hat{\mathbf{j}} + v_z\hat{\mathbf{k}}$ where $v_x = dx/dt$, and so on. Note that there are only three terms (and not six) in (1.2.9) because the Cartesian unit vectors are constant. Note that the *speed* of a particle is the magnitude of its velocity vector. We will discuss how to calculate vector magnitudes in Chapter 3.

Finally, we can further simplify the notation by using dots to denote differentiation with respect to time (i.e., $\dot{x} = dx/dt$). Hence:

$$\mathbf{v} = \dot{\mathbf{r}} = \dot{x}\hat{\mathbf{i}} + \dot{y}\hat{\mathbf{j}} + \dot{z}\hat{\mathbf{k}} \tag{1.2.10}$$

We will find the dot notation to be very useful in the chapters to come.

1.2.1.3 Acceleration

Acceleration, $\mathbf{a}$, is the change of velocity per unit time and is measured in meters per second squared ($\mathrm{m/s^2}$). The acceleration of an object is computed similarly to velocity:

$$\mathbf{a} = \frac{d\mathbf{v}}{dt} \tag{1.2.11}$$

$$= \frac{d}{dt}\left(v_x\hat{\mathbf{i}} + v_y\hat{\mathbf{j}} + v_z\hat{\mathbf{k}}\right) \tag{1.2.12}$$

$$= \frac{dv_x}{dt}\hat{\mathbf{i}} + \frac{dv_y}{dt}\hat{\mathbf{j}} + \frac{dv_z}{dt}\hat{\mathbf{k}} \tag{1.2.13}$$

$$= \dot{v}_x\hat{\mathbf{i}} + \dot{v}_y\hat{\mathbf{j}} + \dot{v}_z\hat{\mathbf{k}} \tag{1.2.14}$$

which, like velocity can be rewritten as $\mathbf{a} = a_x\hat{\mathbf{i}} + a_y\hat{\mathbf{j}} + a_z\hat{\mathbf{k}}$. The acceleration is also the second derivative of the position vector and using the dot notation we can write:

$$\mathbf{a} = \frac{d^2\mathbf{r}}{dt^2} \tag{1.2.15}$$

$$= \ddot{x}\hat{\mathbf{i}} + \ddot{y}\hat{\mathbf{j}} + \ddot{z}\hat{\mathbf{k}} \tag{1.2.16}$$

As we will see in future chapters, the acceleration is very important in classical mechanics. Much of what is discussed in this book is about how to develop an equation for the acceleration, which can then be integrated to get the motion of the system. All measurements of position, velocity, and acceleration, are relative to the frame of reference (choice of origin and axes) from which the measurements are made. We will return to the important idea of reference frames in Section 1.4.

1.2.2 Mass and Force

The basic descriptors of motion simply provide a means of describing how an object is moving. If we want to understand why an object moves as it does, then we need to understand the concepts of mass and force.

1.2.2.1 Mass

The mass of an object is a measure of the object's *inertia*, how strongly the object resists acceleration. For example, a loaded shopping cart is more difficult to accelerate than an empty one. Recall from your introductory physics course that mass is a scalar quantity because in order to describe mass, only a magnitude is needed.

To compare the mass of objects, one could use a beam balance. In order to speak quantitatively about mass, it is helpful to have a standard mass to which all other masses could be compared. Other physical quantities, such as the meter and second, are based on fundamental constants. As mentioned earlier, the meter is based on the speed of light, and the second is based on the ground state hyperfine splitting frequency of Cesium-133 atoms, respectively. The definition of the kilogram has recently changed in order to be based on a fundamental constant.

Before 2019, the unit of mass, the kilogram, was defined to be the mass of a platinum-iridium cylinder stored at the International Bureau of Weights and Measures outside of Paris, France. Hence, the mass of an object could be found by placing the international standard on one side of the beam balance and the object on the other. If the object has a mass of one kilogram, then the balance will remain level, otherwise the balance will tilt towards the more massive object.

The problem with using the old definition of the kilogram is that the mass of the standard changed over time. Although the international standard and its copies (many countries have several copies) were carefully stored, they were occasionally exposed to air and, therefore, absorbed atmospheric contaminates. The problem here is that the copies used by various countries gained mass at different rates than the international standard. Although these differences might be in the micrograms, they were important when measuring sensitive processes such as radioactive decay.

As of May 2019, the kilogram was redefined by setting the Planck constant to be $6.62607015 \times 10^{-34}$ kg $\mathrm{m^2 s^{-1}}$.

Regardless of how the kilogram is defined, you might be wondering why we can use a beam balance to compare masses. We can use a beam balance because weight is proportional to mass according to the *weak equivalence principle*, which says that *gravitational mass* is equal to *inertial mass*. Gravitational mass is the mass that determines gravitational forces between objects; whereas the inertial mass determines the acceleration of an object experiencing a given force. At the time of this writing, experiments show that the two masses are equivalent to about one part in 10^{12}, and future experiments are planned for even more accurate testing.

1.2.2.2 Force

Like mass, the concept of force is one with which we are intuitively familiar. A force is essentially a push or a pull in a particular direction. You push forward on a shopping cart to move it in the direction you wish to go. However, if someone steps in front of your cart as it is moving, you would pull back on the cart to make it stop. From our everyday experience, we know that forces cause motion and that multiple forces can be acting on an object. For example, if you are in the gym lifting weights, then you are exerting a force to raise a dumbbell. If you let go of the dumbbell, it will fall, demonstrating that the force of gravity is also acting on the dumbbell. Hence, we see that in order to understand the motion of an object, we need to know about all of the forces acting on the object. When accounting for all of the forces, we need to know both the magnitude (or amount) and direction of each force. Similar to displacement, velocity, and acceleration, force is a vector quantity.

The unit of force is the newton (abbreviated N) and 1 N is the total amount of force needed to provide an acceleration of 1 m/s^2 to a 1 kg mass. We also know from everyday experiences that there is a direct linear relationship between force and acceleration. For example, a 4 N net force will cause a 1 kg object to accelerate 4 m/s^2. Of course, the acceleration is caused by the vector sum of the forces. When you hold your cell phone, you are exerting a force upwards that matches the downward force of gravity, hence the two forces are equal in magnitude but opposite in direction, and therefore, their sum is zero. Hence, the cell phone does not accelerate.

We have now laid out all of the tools needed to describe and explain an object's motion. Next, we will discuss the core topic of classical mechanics, Newton's Laws of Motion, which will explain the role of force and mass in determining the motion of an object.

1.3 NEWTON'S LAWS OF MOTION

Isaac Newton (1642–1727) developed both calculus and the foundations for classical mechanics. Newton's book, *Philosophiae Naturalis Principia Mathematica (Mathematical Principles of Natural Philosophy)* was published in 1687 and is considered to be one of the most important works in the history of modern science. In the *Principia*, Newton stated his three laws of motion. Newton's laws of motion are vital tools that allow us not only to explain why objects move the way that they do, but they also provide us with a means of predicting an object's motion. In classical mechanics, the importance of Newton's laws cannot be overstated; they are worth committing to memory. As you solve problems in this book, ask yourself how Newton's laws are involved in the setup and solution to each problem. In this section, we will go through each of the three laws in detail.

1.3.1 Newton's First Law

Newton's first law of motion is often remembered by students using the phrase, "An object in motion remains in motion, and an object at rest stays at rest unless acted upon by an unbalanced force." The problem with this phrasing is, what is meant by motion? Does motion mean position, velocity, acceleration, or something else? The proper phrasing of physical laws is critical for developing a good understanding of what the law says. The phrasing of the law should use proper physics terminology, not to be confusing, but rather to be clear!

Newton's First Law

A particle's velocity remains constant if the net force acting on the particle is equal to zero.

Let us look carefully at what the first law says. The first law says that a particle's velocity remains constant if the vector sum of the forces (net force) is zero. The term "net force" is important because there can be forces acting on the particle, but if all of those forces sum to zero, then the particle's velocity doesn't change. The first law is sometimes referred to as the law of inertia because it says that an object will continue moving with a constant velocity forever, if there are no net forces acting on it. Inertia is, simply put, an object's resistance to acceleration. Hence, the particle's inertia will ensure that the particle will continue moving in a straight line with a constant velocity or remain at rest (zero velocity), until it experiences a force that will change the particle's speed, direction, or both.

We know that acceleration is the change of velocity and hence, the first law states that a nonzero net force is needed to cause an acceleration.

Finally, the first law provides a definition for the term *equilibrium*. A particle is in equilibrium if the acceleration of the particle is equal to zero. Hence, one condition for equilibrium is that the net force acting on a particle must be zero. We will later see that when we study rotational dynamics, we will also need the net torque to be equal to zero as an additional condition for equilibrium.

Before moving on to Newton's second law, it is useful to compare our statement of the first law to the more colloquial *"An object in motion remains in motion, and an object at rest stays at rest unless acted upon by an unbalanced force."* While there is nothing technically wrong with this statement, notice that it is longer than our statement of the law on the previous page. The colloquial version breaks down motion and rest as two different behaviors. However, our use of the word *velocity* takes both states (motion and rest) into account because being at rest simply means that the particle's velocity is equal to zero. As mentioned previously, the term *motion* is not one of our descriptors and therefore it is not clear what is remaining constant. Additional terms can be added to explain motion, but at the cost of conciseness. Finally, the term *unbalanced force* is also not as clear as the term *net force.* The term *net force* means vector sum of forces, a clearly defined term. While the colloquial statement of the First Law may use words familiar to most people, a physicist would prefer a statement similar to the one in the box above, due to its use of precise language.

1.3.2 Newton's second law

Newton's second law of motion is used to find a particle's *equations of motion*, which are equations that give the particle's position, velocity, or acceleration at any point in time. We will be using Newton's second law as the central tool for mathematically describing the motion of a particle throughout this book.

Newton's Second Law

A particle's time rate of change of linear momentum, $\mathbf{p}$, is equal to the net force, $\mathbf{F}$, applied to the particle:

$$\mathbf{F} = \dot{\mathbf{p}} \tag{1.3.1}$$

where the linear momentum of a particle with a mass m and velocity $\mathbf{v}$ is defined to be:

Linear Momentum of a Single Particle

$$\mathbf{p} = m\mathbf{v} \tag{1.3.2}$$

Equation (1.3.1) might not be the way you are used to seeing Newton's second law. If we consider a system with constant mass m such as a single particle experiencing a nonzero net force, then we have:

$$\mathbf{F} = \frac{d\mathbf{p}}{dt} = \frac{d}{dt}(m\mathbf{v}) \tag{1.3.3}$$

$$\mathbf{F} = m\mathbf{a} \tag{1.3.4}$$

and we recover the more familiar form of the second law. However, when studying the motion of an object whose mass m is changing with time, such as in the case of a rocket,

the form of the second law presented in (1.3.1) will be the one needed in order to derive the object's equations of motion.

Although we will use the form of Newton's second law $\mathbf{F} = m\mathbf{a}$ most often for calculational purposes in this book, we find it helpful to think of the second law in the form:

$$\mathbf{a} = \frac{\mathbf{F}}{m} \tag{1.3.5}$$

We find this form to be more useful when understanding the concepts behind Newton's second law. Equation (1.3.5) tells us:

- *Acceleration is in the same direction as the net force.* There is no minus sign in front of $\mathbf{F}$ to denote an acceleration in the opposite direction, nor is there any mathematical transformation done to the vector $\mathbf{F}$ to rotate it.
- *The magnitude of the acceleration is directly proportional to the magnitude of the net force.* In other words, net forces with a large magnitude produce larger magnitude accelerations than net forces with small magnitudes. Mathematically, this can be seen because $\mathbf{F}$ appears in the numerator of the fraction in the right-hand side of (1.3.5) and by thinking of the equation as: $|\mathbf{a}| = |\mathbf{F}|/m$.
- *Mass "resists" acceleration.* The mass m appears in the denominator of the fraction in the right-hand side of (1.3.5). Large denominators result in smaller overall fractions when compared to a small denominator with the same numerator (i.e., $1/4 < 1/2$). In other words, with the same given force, objects with a larger mass experience a smaller acceleration, and smaller mass objects experience a larger acceleration.

All three of the above bullet points are contained in the one simple equation (1.3.5)! This is one of the reasons why physicists prefer math as the language for describing the universe. A lot can be said in one simple equation. As a physicist, you should learn how to "read equations" like we did above.

Notice the consistency between the first and second laws. If $\mathbf{F} = 0$, i.e., a zero net force is acting on the particle, then $\mathbf{a} = 0$, and the particle's velocity is not changing, just as stated in the first law. Likewise, we could have restated Newton's first law as: "A particle's momentum remains constant if no external net force acts on the particle."

As we mentioned, the second law will be used to produce differential equations which describe the motion of a particle. As a simple example, we will consider a particle moving in one dimension (along a line) under the influence of a constant force, where $\mathbf{a} = \ddot{x}\hat{\mathbf{i}}$ and $\mathbf{F} = F\hat{\mathbf{i}}$. Recall that the dots above the x denote a second derivative with respect to time. We can use (1.3.5) to write:

$$\ddot{x} = F/m = a \tag{1.3.6}$$

Equation (1.3.6) says that the solution $x(t)$ is a function such that its second derivative is equal to the constant, a. Recall from simple algebra that an equation like $x + 3 = 10$ says x is the number that when added to 3 gives 10. In that case, the solution, x, is a number ($x = 7$). The second-order differential equation (1.3.6) is similar to our algebra problem (but much harder!), except instead of finding a number, you are asked to find a function. Differential equations are the mathematical language used by physicists to describe the motion of a particle. Be prepared to solve them both in closed form (analytically) and numerically. Just to whet your appetite, if a is constant, the solution to (1.3.6) is:

$$x(t) = x_0 + v_0 t + \frac{1}{2} a t^2 \tag{1.3.7}$$

where $x_0 = x(0)$ and $v_0 = v(0)$, are the initial position and velocity, respectively. You can check this for yourself by taking two derivatives of (1.3.7) and showing that it satisfies (1.3.6) in the case of a constant a. In Chapter 2, we will go through the steps to derive (1.3.7) and other solutions to differential equations. If you aren't certain how we obtained (1.3.7) from (1.3.6), don't worry, you should understand it by the end of Chapter 2.

Finally, we can return to the weak equivalence principle. Suppose a particle is near the surface of the Earth and experiences only the force of gravity (weight W):

$$\begin{aligned} F &= W \\ m_{\text{inertial}}\, a &= m_{\text{gravitational}}\, g \end{aligned} \tag{1.3.8}$$

where we have dropped the vector notation, assumed "downward" is the positive direction, and used g for the acceleration due to gravity (9.8 m/s^2). The mass m_{inertial} in this equation is the m from Newton's second law, i.e., it is the mass that "resists" acceleration, and we call this the inertial mass. The mass $m_{\text{gravitational}}$ is the mass that is affected by gravity and is called the gravitational mass. The weak principle of equivalence says that $m_{\text{inertial}} = m_{\text{gravitational}}$, and therefore the masses cancel, and $a = g$ for a freely falling body near the surface of the Earth. If the weak equivalence principle were not true, then the mass of an object would affect its acceleration due to gravity. So far, physicists have not been able to detect any mass dependence (outside experimental error) on an object's acceleration due to gravity, even in the most sensitive of experiments.

1.3.3 Newton's third law

Newton's third law is a statement that discusses the nature of interactions between two particles. In order to state Newton's third law, we need to consider two objects interacting by exerting a force on each other. Let us define $\mathbf{F}_{12}$ to be the force on object 1 exerted by object 2, and $\mathbf{F}_{21}$ is the force on object 2 exerted by object 1. Newton's third law is then stated:

Newton's third law

If object 2 exerts a force $\mathbf{F}_{12}$ on object 1, then object 1 exerts a force $\mathbf{F}_{21}$ on object 2 such that:

$$\mathbf{F}_{21} = -\mathbf{F}_{12} \tag{1.3.9}$$

Notice that the minus sign and lack of a scalar multiple in (1.3.9) says that the force exerted by object 2 on object 1 is equal in magnitude (lack of scalar multiple which, if present, would change the magnitude), but opposite in direction (denoted by the the minus sign) to the force object 1 applies on 2. Many of the interaction forces we will study in this book will be *central forces*. Central forces are forces that act along the line that joins the centers of the two interacting objects, such as gravity and electrostatic forces, and obey Newton's third law. Velocity-dependent forces are not central, and the third law may not apply. An example is the force between two moving electric charges; the Lorentz force is

velocity-dependent, and the magnetic force vectors between the two charged particles do not lie along the same line, hence the resulting net forces do not obey Newton's third law.

Let us take (1.3.9) a little further:

$$\mathbf{F}_{21} = -\mathbf{F}_{12} \tag{1.3.10}$$

$$m_1\mathbf{a}_1 = m_2(-\mathbf{a}_2) \tag{1.3.11}$$

Not surprisingly, we see that the accelerations of each object are in opposite directions. If we continue manipulating (1.3.11), we find:

$$\frac{m_1}{m_2} = -\frac{a_2}{a_1} \tag{1.3.12}$$

where we have taken the magnitude of the acceleration vectors. Notice that ratio of the accelerations is inverse to the ratio of the masses. In other words, if $m_1 > m_2$ then $a_1 < a_2$, in order for the right-hand side of (1.3.12) to be a fraction greater than one. As an example, consider the classic scenario of a mosquito hitting the windshield of a moving automobile. Let the mosquito be object 2 and the car be object 1. Clearly, the automobile has more mass than the mosquito, $m_1 > m_2$, and according to (1.3.12), the acceleration of the mosquito is greater than that of the automobile, $a_2 > a_1$. It is a bad day for the mosquito. Another way to interpret (1.3.12) is that an object cannot accelerate without another object accelerating in the opposite direction.

Next, we need to address one other important fundamental concept for classical mechanics: reference frames. Once we have a working understanding of reference frames, we will have finished laying out the foundations of classical mechanics, and we will be ready to do some physics!

1.4 REFERENCE FRAMES

All of the basic descriptors of motion are measured with respect to a reference frame. A reference frame is a choice of origin, spatial and temporal, as well as a set of axes with respect to which all measurements are made. For example, if you are holding your cell phone in a car moving at a constant speed of 60 MPH while your friend is driving, then you observe your phone to be at rest. However, a bystander on the side of the road will observe your phone to be moving at 60 MPH. Who is correct? They both are, because the measurement of velocity depends on the reference frame. In this case, your frame is moving with the car while the bystander's is at rest on the side of the road (ignoring Earth's motion). By choosing the right reference frame, you may be able to simplify a particular physics problem, in this case a moving phone versus one at rest. For another example, in introductory physics you no doubt worked on problems involving inclined planes. It is well known that choosing one axis to be parallel to the incline greatly simplifies the free-body diagram for the problem. Other changes of reference frame may simply involve changing the time for which $t = 0$.

Another important point to mention about the above example is that the two reference frames are moving relative to one another with a constant velocity. This will make the accelerations measured in each frame the same. Consider the following example, shown in Figure 1.1, with two reference frames S and S', where S' is moving with respect to S with a velocity $u = \dot{d}$, and $d(t)$ is the distance between the two reference frames at a time, t. Note that in this case, primed variables denote the reference frame they belong to not differentiation. In each frame, an observer is measuring the location of the black dot. The

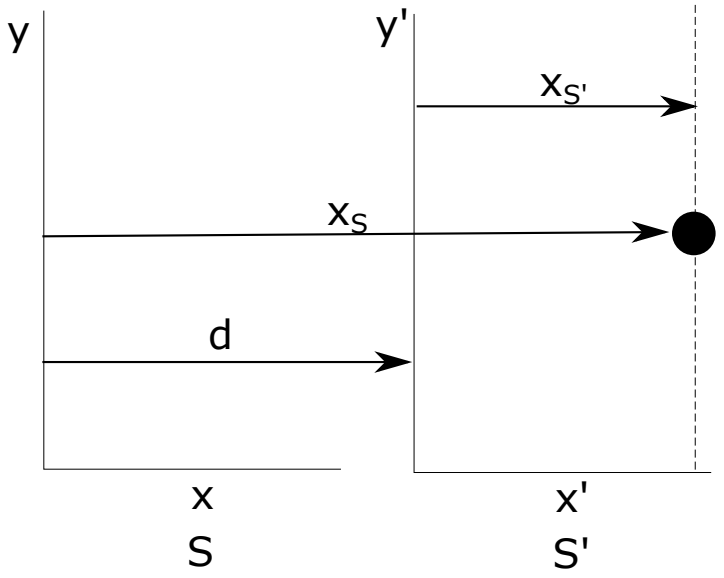


Figure 1.1: The frame S' is moving at a constant velocity with respect to the fixed frame, S. The particle (black dot) is measured to have a position x_S in the reference frame S and $x_{S'}$ in the reference frame S'. The distance between the references frames is d.

observer in S measures the black dot to be located at x_S. Likewise, the observer in S' measures the black dot to be at $x_{S'}$.

Now suppose the observers in each frame want to communicate to each other the motion of the black dot. To keep things simple, we will consider motion only along the x-direction. The coordinate transformation between the two reference frames:

$$x_S = x_{S'} + d \tag{1.4.1}$$

allows the two frames to consistently describe the motion in each frame. First, we will consider the case where S' is moving at a constant speed relative to S, in other words, $u = \dot{d}$ is constant. Differentiating (1.4.1) with respect to time will give us the transformations of the velocity between coordinate systems:

$$\dot{x}_S = \dot{x}_{S'} + u \tag{1.4.2}$$
$$v_S = v_{S'} + u \tag{1.4.3}$$

Hence, if the observers in each frame wanted to determine if they are consistently measuring the velocity of the particle, then they can insert their measured velocities into (1.4.3) in order to check the other's result. Next, we compute the acceleration transformation:

$$\ddot{x}_S = \ddot{x}_{S'} + \dot{u} \tag{1.4.4}$$
$$a_S = a_{S'} \tag{1.4.5}$$

where we used the fact that u (the relative velocity between frames) is constant. Note that (1.4.5) says that the measured accelerations in the two frames are the same, thus (1.4.5)

also tells us that the forces measured on the particle in each reference frame are the same and therefore Newton's laws hold in their typical form in each frame. In other words, if we measured the acceleration of the particle in each reference frame, we would find that the measured acceleration can be accounted for by considering all of the forces acting on the particle. The reference frame, S' is called an *inertial reference frame* because it is moving with a constant velocity. Newton's second law holds in inertial reference frames.

Let us now contrast that with the case of S' accelerating with respect to S. In this case, (1.4.3) doesn't change, and (1.4.5) becomes:

$$a_S = a_{S'} + \dot{u} \tag{1.4.6}$$

and $\dot{u} \neq 0$. Now the two measured accelerations are not the same, and therefore the measured forces in each frame are different. In particular, the acceleration of a particle in a noninertial reference frame cannot be accounted for by summing all of the forces acting on the particle. For example, on certain carnival rides that involve rotation, you will experience an apparent outward force which is not caused by any forces acting on you from the ride itself. This is an example of a *noninertial reference frame* where Newton's laws no longer hold in their standard form because there is an acceleration measured in S that is not apparent in S'. As we will see in a later chapter, we can modify Newton's laws in order to address the case of noninertial frames. Such modifications involve treating the acceleration $\dot{u}$ as coming from an *inertial force*, a force that is not created by physical interactions, but rather is due to an accelerating frame. While the surface of the Earth typically approximates an inertial frame, the Earth does rotate and revolve around the Sun; hence, the velocity of a reference frame "glued" to the Earth's surface is not constant. There are cases where the noninertial nature of the "glued" frame needs to be accounted for. Such cases include long-distance motion such as missile trajectories and the motion of wind and water currents.

1.5 COMPUTATION IN PHYSICS

An important part of this book is the inclusion of computation in solving problems. Computation is defined by Merriam-Webster [MW()] as "the act or action of computing." The term "compute" is further defined by Merriam-Webster [MW()] as "to determine by especially mathematical means" and "to determine or calculate by means of a computer." It is the latter definition that is addressed in this book. So when we say "computation" in this book, we mean using a computer to solve physics and mathematics problems, either analytically or numerically. Physicists use computers in a variety of ways to solve problems. In addition to providing instruction in classical mechanics, this book will also provide you instruction on how to use computers to solve physics and mathematics problems.

Extensive experience in coding is not necessary in order to begin reading this book. In fact, we will mainly rely on commands that are already a part of software packages. That said, you will pick up the coding that you need along the way. In this section, we will provide a motivation for how computing is used in physics and why it is important. Also in this section, we will discuss different types of programming languages, and some direction on how to go about learning to code. Coding has become a fundamental skill for a physicist. We believe that all students should have coding experience before graduating with an undergraduate degree in physics. Hopefully, by the time you have finished this book, you'll have a thorough understanding of both classical mechanics and how to use computers to solve physics problems. Think of it as a two-for-one deal!

1.5.1 The Use of Computation in Physics

To understand how computation is used in physics, we will demonstrate the solution to some problems analytically (also known as "by hand") and using computation. At this time, it is not important to understand all of the physics in the problems, nor is it necessary to understand the computational methods used to solve the problems; we will cover those topics in detail later in this text. The important thing to take away from this section is an appreciation of how computing is used and why computing is important.

Let us start with a simple physics problem that can be "solved by hand."

Example 1.1: Velocity as a function of time

Consider a particle that is moving along a line with a constant acceleration, a. If at time $t = 0$ the particle's velocity is v_0, find the formula for the particle's velocity as a function of time.

Solution:

We know from (1.2.11) that acceleration is the first derivative of velocity with respect to time:

$$\frac{dv}{dt} = a \tag{1.5.1}$$

Next, we separate variables and integrate both sides of the equation. Think of separation of variables as multiplying by dt in order to get all of the $v's$ on one side and the $t's$ on the other. After separation of variables (1.5.1) becomes:

$$\int_{v_0}^{v(t)} dv' = \int_0^t a dt' \tag{1.5.2}$$

where we matched the lower and upper limits on each side. Notice that the lower-limit on the right-hand side is $t = 0$, and the lower limit on the left-hand side is the value of v at $t = 0$, similar for the upper limits. In addition, we included a prime on our variables of integration to distinguish them from the limits of integration. Finally we integrate, noting that a is constant:

$$v(t) = v_0 + at \tag{1.5.3}$$

The result is the velocity as a function of time, as requested by the problem.

Example 1.1 is an example of a problem that is "done by hand," meaning that we were able to perform the necessary mathematical manipulations to solve the differential equation (1.5.1) without the aid of a computer. Equation (1.5.3) is called a "closed form" solution to the differential equation (1.5.1) because it gives a specific solution consisting of functions and mathematical operations. What is considered as "closed-form" is somewhat arbitrary because, for example, a solution in the form of an infinite sum may not be considered in "closed-form."

As you will see in this book, solving physics problems very often involves finding the solution to differential equations. Some of those equations have no closed-form solution, while others are very difficult to find. In those cases, computation can be extremely helpful.

For the purposes of this book, there are two forms of computation:

- *Symbolic Computation*: involves using a computer to help find closed-form solutions to differential equations, integrals, eigenvalues, and much more. You enter the equation

or integral you want solved, and the computer program returns a closed-form solution. You may have seen tables of derivatives and integrals; symbolic computation is a much more sophisticated version of those.

- *Numerical Computation*: (also sometimes referred to as "numerics") provides a list of numerical values representing a solution. For example, a numerical solution to (1.5.1) can be thought of as a table of values with two columns, t and $v(t)$, where each row contains a specific time, t, and the value of v at time, t. Numerical solutions are often best represented as a graph of ordered pairs (and sometimes triplets, depending on the dimension of the problem being solved).

As an example of each type of computation, we will use the programming languages, Mathematica and Python, to solve Example 1.1 both symbolically and numerically. Again, the point of the next two examples is to provide an illustration of each type of computation. We follow each example with an explanation of what is going on in the code. Please note that all of the code that appears in this book can be downloaded at www.routledge.com/9781138495289

Example 1.2: An example of symbolic computation

Using symbolic computation, solve the differential equation from Example 1.

Solution:

The code for using Mathematica to solve (1.5.1) is shown below.

```
solution = DSolve[{v'[t] == a, v[0] == v0}, v, t];

v[t]/.solution[[1]]//TraditionalForm

OUTPUT: at + v0
```

The code for using the interpreted-language Python to solve 1.5.1 is shown below.

```
from sympy import *
init_printing()

v = Function('v')
t = Symbol('t', real = True, positive = True)
a = Symbol('a', real = True)

general_soln = dsolve(Derivative(v(t),t) - a, v(t))

print(general_soln.rhs)

OUTPUT
C1 + a*t
```

The first line in the Mathematica code for Example 1.2 uses Mathematica's *DSolve* command to produce a solution to the differential equation. Notice that the differential equation and its initial condition appear as the first argument of the command (in between the curly brackets). Notice also the "==" as opposed to "=" in the equation. Often in programming languages, the single equal sign is used for variable assignment, whereas the double-equal sign is used for equivalence. The first line of code stores the solution generated by *DSolve* in

the variable *solution* (notice the single equal sign). The second argument v tells Mathematica that v is the function for which we are solving, and the third argument identifies t as the independent variable. The second line in the code outputs the solution. The command *TraditionalForm* displays the output in a more readable format; without using *TraditionalForm*, Mathematica would display the output differently. Notice that Mathematica's solution is the same as the one we derived in Example 1.1.

The Python code for Example 1.2 is much more involved than the Mathematica code. The first line, "from sympy import *" causes Python to import the SymPy library. Python, by itself cannot perform symbolic manipulations. We needed to import the SymPy library in order to expand Python's capabilities. Libraries are written by experienced programmers, and they include functions which can be used by any code that imports the library. In this case, the SymPy library includes functions like *dsolve* which solve differential equations in closed form. Note that this is not the same as the Mathematica command, *DSolve.* Commands from one language typically cannot be used in another. In order to make a distinction between the two languages, we will preserve the capitalization used in each language. For example, if we write *DSolve,* then we are discussing Mathematica's command for solving differential equation because in Mathematica the D and S are capitalized in the command. The letters D and S are not capitalized in Python and, therefore, when we discuss solving differential equations in Python, we will write the command *dsolve,* as it appears in the language of Python.

A word of warning: libraries, while useful, may not be free of bugs. It is best practice to test any library functions you are using in your code to make sure they behave as expected. Python libraries like SciPy, NumPy, and SymPy are well-tested and well-documented. Documentation for these and other Python libraries can be found online.

Mathematica automatically recognizes a, vo, and t as symbols to be manipulated. In addition, the Mathematica syntax, $v[t]$, allows Mathematica to automatically recognize v as a function. However, Python by itself would think of a, vo, t as variables for assignment and would not recognize $v(t)$ as a function. Therefore, all of the symbol identifications need to be specified in Python. The *Symbol* and *Function* commands from the SymPy library do exactly that. The general solution of the differential equation is given by the SymPy command *dsolve*, whose arguments are the differential equation and the function for which the differential equation is being solved. Notice that in Python, the differential equation is written such that the right-hand side is equal to zero, and only the left-hand side of the differential equation is entered. At the time of this writing, Python's *dsolve* command cannot solve the differential equation with the initial conditions at the same time (except for power series solutions), unlike Mathematica's *DSolve.* With more code, one could perform the substitutions, however, they are not included here in order to prevent (further) confusion. The output of the Python program is included below the word, OUTPUT. The word, OUTPUT, and the line below it are not part of the Python code; we included it only to show the output of the program. In this book, we will often show the output of a Python program in this manner. Note the the *C1* in the output represents a constant of integration.

In Example 1.2, we see that Mathematica requires a much shorter code to solve this problem than Python because Mathematica is designed, in part, for symbolic manipulations. Computer algebra systems (CAS) like Mathematica and Maple are very convenient and easy to use when performing symbolic computations. The disadvantage is that they are both proprietary and more expensive than other options, such as the open-source free CAS SageMath (https://www.sagemath.org/).

Next, we will look at an example of a numeric computation and compare the two languages in that context.

Example 1.3: An example of numeric computation

Using $a = 9.8$ m/s^2 and an initial velocity of $v(0) = v_0 = 1$ m/s, find and graph a numerical solution to the differential equation from Example 1.1.

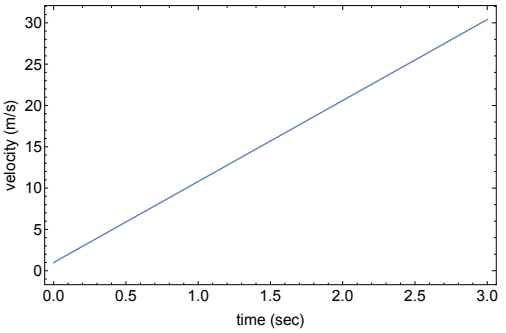
Solution:

The code for using Mathematica to solve (1.5.1) is shown below.

```
a = 9.8;

solution = NDSolve[{v'[t] == a, v[0]==1.0}, v, {t, 0, 3}];

Plot[v[t]/.solution, {t, 0, 3}, Frame → True, Axes → False,
FrameLabel → {"time (sec)", "velocity (m/s)"}, BaseStyle → {FontSize → 18},
ImageSize → Large]
```



The Python code used to solve (1.5.1) is shown below. For brevity, we did not include the resulting graph which is the same as the one above.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

vo = 1


def velderiv(v,t):
    a = 9.8
    dvdt = a
    return dvdt

times = np.linspace(0,3,30)
velocity = odeint(velderiv,vo,times)

plt.plot(times,velocity)
plt.xlabel('time (sec)')
plt.ylabel('velocity (m/s)')
plt.show()
```

In order to produce a numerical solution for Example 1.3, both languages needed to have numerical values for all variables. In this case, we needed to specify a and v_0. In the Mathematica program, we used the first line of the code to specify the value of a, but we defined the initial value of v in the Mathematica command *NDSolve*, which produces a numerical solution to a differential equation. Notice that the arguments for *NDSolve* are similar to those of *DSolve*, i.e., the order of the arguments are: equation to be solved and initial values, the function for which we are solving (v), and identification of the independent variable, t. This time, however, we needed to specify the range of t values for which v is being solved. Recall the idea of a numerical solution being like a table of values containing columns t and $v(t)$, so we need to tell the computer when to start finding the solution ($t = 0$) and when to stop ($t = 3$). The final line of the code produces a graph of the solution. Notice, we get the expected line with y-intercept of 1.0 and slope of 9.8. Everything after $\{t, 0, 3\}$ (which dictates which values of t should be used to make the plot) in the final line of the code are formatting commands which only affect the visual appearance of the graph. We show the formatting commands so that you may make similar plots.

In the Python example, we needed to import three libraries, NumPy, SciPy, and Matplotlib in order to perform the task of numerical integration. From the SciPy library we imported only one function, *odeint* which will be used to numerically solve the differential equation (1.5.1). Later in this book, we will talk about how to numerically solve differential equations without using the libraries, but for now, we wanted to use functions included in the aforementioned libraries in order to simplify the code. The NumPy and SciPy library contain functions and algorithms critical for scientific computing. The functions contained in NumPy and SciPy will greatly increase the speed of code written in Python, and we strongly encourage you to use them whenever possible. The Matplotlib library is a plotting library that will allow you to make graphs in Python. After the libraries are imported, we defined the initial condition variable *vo* and the differential equation. In Section 13.2, we comment more on libraries, the risks of using other people's code, and the *odeint* command.

To solve the differential equation numerically, we created a new function in Python that we called *velderiv*, which will contain the differential equation we are solving. User-defined functions, like *velderiv*, are convenient in programming languages when a particular calculation needs to be repeated many times. In Python, function definition is done using the command *def* and is demonstrated in the above code. The arguments of the function are included in parentheses following the function name. In this case, the arguments are v and t. The next few lines contain the actual calculation of the function. Notice we included a local variable *dvdt*, which stays within the function and is equal to the first derivative of v. Equation (1.5.1) states that the first derivative of v is equal to a. The last line of the function begins with the command *return,* which tells Python what value to return when the function is called. In this case, we return the value of the variable *dvdt*. Hence, the function *velderiv* returns the first derivative of v.

For the purposes of solving a differential equation, we write the function of the form,

$$\frac{dx}{dt} = f(x,t) \tag{1.5.4}$$

where the derivative is on the left-hand side and everything else on the right. As described in the paragraph above, the function *velderiv* contains the right-hand side of the differential equation (the side without derivatives). We will expand this procedure to second-order differential equations in Chapter 2. After the differential equation is defined as a function, we need to tell Python for which values of t we will be computing v. In this case, *times* is an array which includes the values of t, which will be used to compute v. Arrays are lists of values. In this case the array *times* contains the list $[t_0, t_1, \ldots, t_{30}] = [0, 0.1, \ldots, 3.0]$, and

hence we will be solving for $v(0)$, $v(0.1)$, $\ldots, v(3.0)$. Next, it is time to solve the differential equation. The command, *odeint*, used to solve the differential equation comes from the scipy.integrate subpackage of the SciPy library. The *odeint* command requires the differential equation (as a user-defined function, *velderiv*), the initial value (v_0), and the list of times, t as arguments. The result is an array, we called *velocity*, and it contains values $[v_0, v_1, \ldots]$, where $v_i = v(t_i)$. The last four lines of the program set up the graph of v versus t, which is not shown.

Again, the Python code is longer than the Mathematica code. Does that make Mathematica better than Python? That answer depends on what you are trying to do. If you are interested in quick development and implementation, then Mathematica might be a very useful tool for you. However, as we mentioned before, Mathematica is proprietary and more expensive than Python. The proprietary nature of Mathematica means that you don't have access to the source code, and this can be a problem if you want to know exactly what the commands are doing. Python is open source, so if you want to know what is "going on under the hood," you can find out. The open source nature of Python is valuable when doing research, and you want to identify if unexpected results are due to bugs in code or new science. That said, Mathematica is a language of its own, and you can write your own programs in it. Finally, Python is free, and your budget may dictate your choice. We will elaborate more on language selection in the next section. When working on their own research problems, the authors of this book will often use both programs, choosing the most appropriate tool for the particular job at hand.

It should be pointed out that sometimes, a closed-form solution to a differential equation doesn't exist or is difficult to find. In those cases, we rely on numerical solutions to give us the information we need. You may think that numerical solutions are of limited value, but as we will see throughout this book, there is a lot that can be found using numerics. In Chapter 13, we will study nonlinear oscillators, and we will learn how to obtain a lot of information from numerical solutions.

It is easy to walk away from an undergraduate physics education thinking that all physics problems have closed-form solutions. This conclusion arises from the types of problems that undergraduate students solve as part of their education. The truth of the matter is that most problems physicists work on outside of the classroom require numerical solutions. The differential equations governing real-world systems are often complex and not solvable in closed-form. In those cases, numerics may be the only option to gaining any kind of insight into the problem. A physicist with strong computational skills will be well-prepared to tackle a wide variety of problems, not just in classical mechanics but in any field of physics or engineering.

Furthermore, physicists find themselves working on a variety of problems outside the traditional subfields of physics. Physicists often end up working on problems in finance, economics, biology, climate science, and materials science, just to name a few. Today's physicists are involved in modeling economies, disease propagation, and social networks. These types of problems allow physicists to apply their skills, including strong computational and modeling skills, to interesting problems. It is our hope that this book will not only prepare you well in physics but also set you on the path of developing strong computational skills, so that you may tackle the exciting problems both inside and outside of physics, wherever your career may take you.

1.5.2 Different Computational Tools

A carpenter has a toolbox filled with a variety of tools, each tool designed for a specific task. The same is true for computation: there are different types of programming languages, each ideal for a variety of tasks. A computer programming language (here after, we will just call them "languages") is a special language programmers use to give a computer instructions on how to perform a specific task. In this book, those tasks will focus on performing the mathematics needed to solve physics problems. There are many different computer languages used by physicists to solve problems; each language has its own set of advantages and disadvantages. Just like a carpenter can use a screwdriver as a hammer, computer languages can be used in applications they might not have been originally intended for, to varying degrees of success. It is important for a physicist to be comfortable with a few different programming languages so that he or she will be prepared for a variety of problems.

In this book, we will divide languages up into three types: computer algebra systems (CAS), interpreted languages, and compiled languages. This division is not intended to be complete and cover all languages, but it will establish a way of thinking for how we will approach languages in this book. Keep in mind that many languages will actually fit in multiple categories, for example any language can be compiled or interpreted, but we will assign languages by type in the way they are most often used.

- ***Computer Algebra System (CAS):*** CAS is software that can manipulate mathematical expressions, similar to the ways you have learned in your mathematics courses. CAS can solve a variety of mathematical problems in closed form, as we saw in Example 1.2. Besides symbolic manipulations, CAS often has numerical components and its own programming language. For example, the CAS Mathematica is also an interpreted language (see below). Examples of CAS are: Mathematica, Maple, and SageMath.

- ***Interpreted Language:*** An interpreted language is one that can be executed directly from the source code using software called an *interpreter*. Many people feel that interpreted languages are easier to learn, have a shorter development time (i.e., it takes less time to write code), and are easier to debug (find and fix errors in code). Interpreted languages are generally easier to run on multiple operating systems, i.e., the same Python (an interpreted language) script can be run on a Windows and a Linux computer, provided both computers have Python installed on them. Examples of interpreted languages include: Python, Mathematica, R, *MATLAB*, and JavaScript.

- ***Compiled Language:*** A compiled language is one whose source code needs to be compiled by software called a *compiler,* which transforms the source code into machine code and creates an executable. The executable is then run in order to perform the instructions contained in the program. If changes are made to the program, it needs to be recompiled, unlike an interpreted language. Likewise, the compiler creates instructions for the specific type of computer being used. Hence the executable created by compiling a program on a Windows computer will not work on a Mac. Compiled languages typically have a shorter run-time compared to interpreted languages, although for simple computational problems, the difference can be minimal. Examples of compiled languages are: FORTRAN, C, and C++.

The above categories are not intended to be strict divisions. We already mentioned that theoretically, any language can be either interpreted or compiled. In addition, although we will tend to think of Mathematica as a CAS in this book, Mathematica is also an interpreted language which can be used to solve any numerical or symbolic computational

problem. Python, an interpreted language, has a library called SymPy, which, as we saw above, allows Python to become a CAS.

The type of language that you use depends on the nature of the problem you are trying to solve. If you need help simplifying the mathematical form of an equation, a CAS is probably your best choice. While SymPy may be capable of mathematical manipulations, you may find Mathematica easier to use for that problem, since Mathematica was specifically designed for, among other things, symbolic manipulations. Interpreted languages are helpful for problems that are not computationally intensive, such as numerically solving certain differential equations. The speed of compiled languages makes them ideal for computationally intensive programs, such as climate models or modeling the motion of many stars in a galaxy.

In this book, we are not going to be tied to one specific language. The focus of this book is on classical mechanics and computation, not on a particular programming language. The problems we will solve in this book are not computationally intensive, i.e., they won't demand a lot of computer resources or time. Hence, we will not be using a compiled language in this book. For the sake of consistency, we chose to demonstrate the computations using two languages, Mathematica and Python. The exercises and examples done in this book can be done in any language you prefer, and, when appropriate, the code will be written in such a way that it will be easy to reproduce in your language of choice. However, we felt it would be best to avoid using too many different languages in the examples done in the text.

Our choice of Mathematica and Python as example languages is not random. Both languages are used extensively in physics, and both languages are mature. The code that we will write will focus on core commands that have been around for a long time and are not likely to become outdated anytime soon. Python is free to use and available online for any platform. You'll likely want to download a distribution like Anaconda (`https://www.anaconda.com`) which comes with an integrated developer environment (IDE). Mathematica is not free, however, there are open source alternatives, such as SageMath which is available online (`https://www.sagemath.org/`). SageMath was developed as an open-source alternative to Mathematica. While there will be syntax differences between Mathematica and SageMath, the symbolic calculations done in this text in Mathematica can, for the most part, be done in SageMath.

You do not need to have experience with any programming languages to understand the material presented in this book. Such previous experience will undoubtedly help in understanding the code contained in the book. However, we will introduce the necessary programming concepts as they are needed, and we assume no programming experience. Mathematica has extensive and very useful help files that we recommend going through. If you'd like to get a head start on Python, we recommend Code Academy (`https://www.codecademy.com/`) and the most recent edition of [Kinder and Nelson(2015)]. A quick way of getting your feet wet and learning some basic programming skills is through using your favorite search engine. Searches such as: "How to solve a differential equation in Mathematica" or "How to integrate an equation in Python" often lead to many useful pages, which will at least give you the syntax for how to solve a specific problem. While this is no substitution for an introductory computer science course or a course in numerical analysis, it will certainly get you started.

1.5.3 Some Warnings

We will end this section with some warnings. First, the availability of computational tools to solve problems does not mean that you do not need to know how to do math by hand! There are several reasons for this. The first reason is that computer algorithms can and do give wrong answers. Computational algorithms have significantly improved over the decades, and wrong answers from them are not as common as in the early days of computing. However, you need to be able to identify a wrong answer when you see one. A proficiency with mathematics is still needed for identifying wrong answers. Furthermore, it is not unusual for one to perform additional algebraic manipulation to the output of a CAS in order for the result to be in a more convenient or insightful form. We will see some examples in this book where the output of a CAS will be technically correct, but not as useful for describing the physics of the situation as compared to a result that we would have obtained "by hand."

Second, a computer will provide a solution with no context. The techniques and critical thinking that you learn by doing mathematics will train you in how to understand and interpret computer-generated solutions, so that you can put the solution in the proper context. For example, does your solution have appropriate limits? Does the solution conform to known laws of physics? A person skilled in mathematics and physics is still required to interpret computer-generated solutions (at least for now...). In particular, when one becomes skilled at mathematics, symbolic algorithms should be considered as a tool to help one focus on the physics without getting encumbered by lots of mathematics. In a sense, you should think of symbolic algorithms as being similar to a calculator. Only after learning arithmetic "by hand," should one begin to use a calculator. In the same sense, mastery of algebra, trigonometry, and calculus, should be a prerequisite for extensive use of symbolic algorithms.

Third, who programs computers to solve math problems? The people who know how to solve the math problems. Hence, we need people who can do math, in order to ensure that we will have future generations of algorithms to assist us.

The final warning comes from the nature of the algorithms presented in this text. The authors do not claim that the codes contained within are the most efficient means of solving the problems in the text. We tended to produce codes that focused on pedagogy over efficiency. We encourage you to rewrite the programs contained within, to see if you can improve them and make them more efficient. By playing around with the codes, you'll get a better understanding of the algorithms and programming in general. Have fun!

1.6 CLASSICAL MECHANICS IN THE MODERN WORLD

Students sometimes come to a classical mechanics text wondering about the relevance of the material to modern-day physics. They have questions about why they need to learn classical mechanics when "all of the interesting stuff" is in the quantum realm or deals with relativity. While some of the most exciting popular science books deal with the realms of quantum mechanics and cosmology, there is plenty of active research going on in classical systems.

One of the most exciting fields of research in classical systems is in nonlinear and complex systems. Nonlinear systems is a field of science that attempts to understand systems whose governing equations are nonlinear, and often not solvable in closed-form. These types of systems include large amplitude oscillators, weather, predator-prey models, and certain chemical reactions. Notice that two of these three examples are not from physics. That is because nonlinear systems is a multidisciplinary field and one in which physicists have made many contributions. Complex systems, simply put, are large systems of many inter-

acting agents whose collective behavior cannot be explained by the dynamics of the individual agents alone. Examples of complex systems include power grids, global climate, and economies. Again, these examples aren't strictly "physics problems" but rather problems to which physicists have contributed greatly. Progress in the fields of nonlinear and complex systems has required people trained in physics, notably classical mechanics and statistical mechanics.

Another important field involving classical mechanics is fluid mechanics and turbulence. There is a famous story of Werner Heisenberg, that on his deathbed he said, "When I meet God, I am going to ask him two questions: Why relativity? And why turbulence? I really believe he will have an answer for the first." Variations of this story are told about other physicists as well. The point is that turbulence continues to be a difficult problem in classical mechanics. Improvements in our understanding of turbulence will greatly benefit many technologies that involve fluid flow, such as pipelines and flight.

The paragraphs above contain just a few examples of the relevance of classical mechanics to the modern world. We have left out many other examples such as mechanical engineering, oceanography, and space flight. The point is that ignoring classical mechanics or treating it simply as a step to get to something else, is a mistake. A fundamental understanding of classical systems is critical to becoming a complete physicist. In many ways, classical mechanics serves as the foundation of the field of physics itself. A physicist needs a solid foundation of classical mechanics in order to understand other sub fields of physics.

One of the many benefits of learning classical mechanics is that it gives you the ability to identify what factors influence a system (physical or otherwise) and how to express those factors mathematically. Much of this ability centers on the idea of learning how to specify a system, identify the forces acting on the system, and then writing down the relevant equations of motion that govern the system. It is a powerful and useful way of thinking, that will be a critical component of your training as a physicist and is a valuable skill regardless of how you end up using your physics training.

1.7 CHAPTER SUMMARY

Motion is described using the *basic descriptors* of motion, displacement, velocity, and acceleration. In Cartesian coordinates the position, velocity, and acceleration of a particle are:

$$\mathbf{r} = x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}} \tag{1.7.1}$$

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} = \frac{dx}{dt}\hat{\mathbf{i}} + \frac{dy}{dt}\hat{\mathbf{j}} + \frac{dz}{dt}\hat{\mathbf{k}} = v_x\hat{\mathbf{i}} + v_y\hat{\mathbf{j}} + v_z\hat{\mathbf{k}} = \dot{x}\hat{\mathbf{i}} + \dot{y}\hat{\mathbf{j}} + \dot{z}\hat{\mathbf{k}} \tag{1.7.2}$$

$$\mathbf{a} = \frac{d\mathbf{v}}{dt} = \frac{dv_x}{dt}\hat{\mathbf{i}} + \frac{dv_y}{dt}\hat{\mathbf{j}} + \frac{dv_z}{dt}\hat{\mathbf{k}} = a_x\hat{\mathbf{i}} + a_y\hat{\mathbf{j}} + a_z\hat{\mathbf{k}} = \ddot{x}\hat{\mathbf{i}} + \ddot{y}\hat{\mathbf{j}} + \ddot{z}\hat{\mathbf{k}} \tag{1.7.3}$$

where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$ are the unit vectors.

When describing the motion of an object, two additional quantities become important, the mass of the object and the net force acting on the object. The inertial mass of an object m_{inertial} is a measure of the object's inertia, where as the gravitational mass of an object $m_{\text{gravitational}}$ is the mass that determines the gravitational force acting on it. However,the weak principle of equivalence says that $m_{\text{inertial}} = m_{\text{gravitational}}$.

A change in an object's motion is caused by a nonzero net external force acting on the object. Newton's laws of motion describe how forces change the motion of an object.

Newton's First Law: A particle's velocity remains constant if the net force acting on the object is zero.

Newton's second law: A particle's time rate of change of momentum $d\boldsymbol{p}/dt$, is equal to the net force $\mathbf{F}$ applied to the particle :

$$\mathbf{F} = d\boldsymbol{p}/dt = \dot{\mathbf{p}} \tag{1.7.4}$$

Newton's third law: If object 2 exerts a force $\mathbf{F}_{12}$ on object 1, then object 1 exerts a force $\mathbf{F}_{21}$ on object 2 such that:

$$\mathbf{F}_{21} = -\mathbf{F}_{12} \tag{1.7.5}$$

All measurements are made with respect to a reference frame. A reference frame is a choice of origin (spatial x and temporal t), and a set of axes with respect to which all measurements are made.

Two reference frames S, S' are called inertial reference frames when Newton's laws hold, and the forces measured in S' are the same forces as those measured in S.

Two reference frames S, S' are called noninertial reference frames when Newton's laws no longer hold in their standard form, because there is an acceleration measured in S that is not apparent in S'.

Computation is an important tool for physicists. In this book, we'll focus on two types of computation, symbolic and numerical. Symbolic computation involves using a computer to help find closed-form solutions to differential equations, integrals, eigenvalues, etc. Numerical computation or "numerics," provides a list of numerical values representing a solution to differential equations, integrals, eigenvalues, etc.

Computer programming languages can be loosely categorized as a computer algebra systems, interpreted languages, and compiled languages. A computer algebra system (CAS) is a software that can manipulate mathematical expressions, and can solve a variety of mathematical problems in closed-form. An interpreted language is one that can be executed directly from the source code, using software called an interpreter. A compiled language is one whose source code needs to be compiled by software called a compiler, which transforms the source code into machine code.

1.8 END-OF-CHAPTER PROBLEMS

Section 1.2: The Basics of Classical Mechanics

1. A 0.50 kg particle's position can be described using the vector function: $\mathbf{r}_0 = 3t\hat{\mathbf{i}} - 2t^2\hat{\mathbf{j}} + 7t^{-2}\hat{\mathbf{k}}$.
   (a) What are the units of each of the coefficients in each component? Assume that time is measured in seconds and position is measured in meters.
   (b) Compute the velocity of the particle at $t = 3$ seconds.
   (c) Compute the acceleration of the particle at $t = 1$ second.
   (d) Find the force on the particle at $t = 1$ second.
2. Consider the position vector $\mathbf{r} = 3.0t\hat{\boldsymbol{\rho}}$, where $\hat{\boldsymbol{\rho}} = \cos\phi\hat{\mathbf{i}} + \sin\phi\hat{\mathbf{j}}$ and $\phi = \phi(t)$. Compute the velocity of the particle.
3. A particle's position can be given by $\mathbf{r}(t) = R\cos(\omega t)\hat{\mathbf{i}} + R\sin(\omega t)\hat{\mathbf{j}}$, where R and ω are constants. Describe the path of this particle. What is the particle's acceleration as a function of time?

Section 1.3: Newton's Laws of Motion

4. Using Newton's third law, explain why a gun recoils when it is fired. Explain why the gun's recoil velocity is smaller than that of the bullet.

5. Consider a universe where the weak equivalence principle was not true. How would Newton's universal law of gravitation be changed?

6. Using Newton's laws, explain what would happen to the Earth's trajectory if the Sun were to suddenly disappear.

7. A ball is accelerating along the y-axis, in which direction is the net force acting on the ball?

Section 1.4 Reference Frames

8. Consider a reference frame S, which is at rest and another reference frame S' whose origin is at a location $\mathbf{r}_0 = ct^2\hat{\mathbf{i}}$ relative to the origin of S, where c is a positive constant. A particle of mass m has a position in S', which described by the vector function, $\mathbf{r}' = bt^3\hat{\mathbf{i}}$, where b is a positive constant.

   (a) Compute the acceleration of the particle as measured in each frame.

   (b) Compute the force on the particle measured in each frame. Why are the two forces different?

9. Consider a reference frame S, which is at rest and another reference frame S', whose origin is at a location $\mathbf{r}_0 = at\hat{\mathbf{i}} + bt\hat{\mathbf{j}}$ relative to the origin in S, where a and b are positive constants. Compute the velocity of a particle as measured in S, whose position in S' is described by $\mathbf{r}'(t) = -ct^2\hat{\mathbf{i}}$, where c is a positive constant.

10. Consider a reference frame S, which is at rest, and another reference frame S', whose origin is at a location $\mathbf{r}_0 = 3t^2\hat{\mathbf{i}} - 5t^3\hat{\mathbf{j}}$ relative to the origin of frame S. A particle's position in S' is described by the vector function, $\mathbf{r}'(t) = -t^2\hat{\mathbf{i}} + 3t^2\hat{\mathbf{j}}$. Find the force measured in each frame if the mass of the particle is m.

11. In the theory of special relativity, measurements between two inertial frames are related by the factor $\gamma = 1/\sqrt{1 - v^2/c^2}$, where c is the speed of light, and v is the speed of the moving frame S' relative to the rest frame S. For example, time passes at different rates depending on the speed of an observer, by the rule: $\Delta t = \gamma \Delta t'$, where Δt is the interval of time measured in the rest frame S, and $\Delta t'$ is the duration of the same interval of time as measured in a moving frame S'. If a rocket leaves the Earth and travels for 100 years, as measured by people on Earth, at a speed $0.75c$, how much time has passed for the travelers on the rocket?

12. In the theory of special relativity, we need to alter the transformation equations between two frames. Let S be a rest frame, and S' a frame moving at a speed v relative to S. Suppose that S' moves in the x direction, in other words, S' moves parallel to the x-axis of the frame S. The theory of special relativity then states that measurements made in S and S' are related by the formulas:

$$\begin{aligned} x' &= \gamma(x - vt) \\ y' &= y \\ z' &= z \\ t' &= \gamma\left(t - \frac{vx}{c^2}\right) \end{aligned}$$

where $\gamma = \sqrt{1 - v^2/c^2}$. Now consider that two events in S occur at two locations $x = 0$ and $x = a$ at time $t = 0$. Find the times of the two events as measured in S'. Notice that events that are simultaneous in S are not simultaneous in S'. Which event was seen first in S'?

Section 1.5 Computation in Physics

The problems in this section are different from the rest of the book. They are intended to prepare you to better understand the computational aspects of this book. The problems may be done in any language specified by your instructor. You may want to consider doing some of the problems in Python and/or Mathematica so that you have some familiarity with the code that appears throughout this text. As long as it is approved by your instructor, online searches can be of significant help in solving these problems. Finally, these problems are not exhaustive for the different types of problems you'll encounter in this text. We will introduce new algorithms and commands as they are needed.

13. Assign the values 5 and 6 to the variables, a and b. Then compute:
    (a) $a + b$
    (b) $b - a$
    (c) ab
    (d) b/a
    (e) a^b

14. Compute the following:
    (a) $\sin(0.25\pi)$
    (b) $\cos(0.25\pi)$
    (c) e^3
    (d) $\sqrt{7}$

15. Define an array (sometimes called a list) called t which contains the numbers 0 through 2π in steps of:
    (a) 0.1
    (b) 0.01
    (c) 0.25

16. Define the function $f(x) = e^{-mx}cos(2\pi kx)$ and compute $f(x)$ for three different combinations of values of x, m, and k.

17. Plot the function from Problem 16 with $k = 1$ and $m = 0.5$ for values x starting at $x = 0$ and ending at $x = 10$. Note that you may need to create an array for x, and that in that case use steps of 0.01. Label the axes and choose a color other than the default for your curve. Save the resulting graph to a .jpg or .png file.

18. Plot $f(x) = x^2$ and $g(x) = x$ on the same graph for the range $x = 0$ to $x = 10$. Each function should have its own color in the graph.

19. A conditional statement is an important tool for helping computers deal with contingencies. They take the form "if-then-else." Conditional statements allow for computers to do different things, depending on whether or not a condition is true. In other words, if a condition is true, then do one thing, if it is not true (else) then do something different. For example, if a variable x is less than 5 assign the value 1 to the variable a; otherwise, assign the value 0 to a. Write a code in which you assign a value to the variable x and that prints "x is less than 5" if $x < 5$, and prints "x is greater than or equal to 5" if $x \geq 5$.

20. Using the conditional *if* statement, described in Problem 19, define the piecewise function

$$f(x) = \begin{cases} 0 & x < 5 \\ x^2 & x \geq 5 \end{cases}$$

and evaluate $f(x)$ for $x = 3$ and $x = 7$ and plot $f(x)$ for the range $x \in [0, 10]$.

21. Computers are very good at repeating tasks over and over again, in a procedure called a *loop*. The basic types of loops are *for* loops, *do* loops, and *while* loops. Using any kind of loop, compute 6! (factorial) and display the result.

22. Using the methods learned from the problems above, compute the first 20 values of the Fibonacci sequence. If you are unfamiliar with the Fibonacci sequence, do an online search. You will need to use loops, and you may want to save your results into an array. Look up how to *append* values to an array and how to recall specific values for an array in the language of your choice.

23. Use symbolic computation to solve the following equations for x. Note that you will get complex roots for some of the solutions.

    (a) $7x + 5 = 0$
    (b) $x^2 - 5x + 2 = 0$
    (c) $x^3 + 7x - 5 = 3$

24. Use symbolic computation to solve for (x, y):

$$\begin{aligned} 2x - 5y &= 7 \\ x + y &= 2 \end{aligned}$$

25. Use symbolic computation to solve the following differential equation for the unknown function $f(x)$:

$$\frac{df}{dx} = -x^2 + 3$$

with the initial condition $f(x = 0) = 3$. Plot the solution.

26. Numerically solve the differential equation:

$$\frac{df}{dx} = 5\sin(x) - 4e^{-x}$$

for the initial condition $f(0) = 0$. Plot the solution for $x = 0$ to 3 with a step size of 0.1.

27. Numerically solve the coupled differential equations:

$$\frac{dx}{dt} = 0.6x - 1.2xy,$$
$$\frac{dy}{dt} = xy - y,$$

using the initial conditions $x(0) = 40$ and $y(0) = 7$. Plot x as a function of time, y as a function of time, and y as a function of x.

28. Use symbolic computation to perform the integral:

$$\int x^2 cos(x) dx.$$

29. Numerically evaluate the definite integral:

$$\int_0^{2\pi} x^2 e^{2x} dx.$$

30. Numerically evaluate the definite double integral:

$$\int_0^1 \int_0^2 xydxdy.$$

If you are using Python, you will want to consider the *nquad* command from the SciPy integrate library.

31. Numerically evaluate the integral:

$$\int_0^2 \int_0^{1-y} xydxdy.$$

As with Problem 30, you will want to consider using the *nquad* command if you are using Python.

References

Merriam-Webster online dictionary. https://www.merriam-webster.com/. Accessed: 2018-02-18.

Massachutsetts institute of technology open course ware. https://ocw.mit.edu/index.htm, 2018. Accessed: 2018-05-31.

Wolfram language and system documentation center. https://reference.wolfram.com, 2018. Accessed: 2018-05-31.

HDI Abarbanel . Analysis of Observed Chaotic Data. Springer, New York, 1996.

VI Arnold . Mathematical Methods of Classical Mechanics. Springer, New York, 2nd edition, 1997.

OL de Lange and J Pierrus . Solved Problems in Classical Mechanics. Oxford, New York, 2010.

AJ Ellington . A meta-analysis of the effects of calculators on students achievement and attitude levels in precollege mathematics classes. Journal for Research in Mathematics Education, 34(5):433463, 2003.

RH Enns . Its a Nonlinear World. Springer, New York, 2011.

J Fajans and L Fridland . Autoresonant (nonstationary) excitation of pendulums, plutinos, plasmas, and other nonlinear oscillators. American Journal of Physics, 69:10961102, 2001.

H Goldstein , CP Poole , and JL Safko . Classical Mechanics. Pearson, London, 3rd edition, 2001.

A Hadhazy . Fact or fiction: The days (and nights) are getting longer. Scientific American, 6 2001.

WR Hamilton . On a general method in dynamics; by which the study of the motions of all free systems of attracting or repelling points is reduced to the search and differentiation of one central relation, or characteristic function. Philosophical Transactions of the Royal Society, Part 2, 247308, 1834.

WR Hamilton . Second essay on a general method in dynamics. Philosophical Transactions of the Royal Society, Part 1, 95144, 1835.

R Hilborn . Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers. Oxford University Press, New York, 2nd edition, 2001.446

H Kantz and T Schreiber . Nonlinear Time Series Analysis. Cambridge University Press, Cambridge, UK, 2nd edition, 2004.

JM Kinder and P Nelson . A Students Guide to Python for Physical Modeling. Princeton University Press, Princeton, NJ, 2015.

E Lorenz . Deterministic nonperiodic flow. Journal of Atmospheric Sciences, 20: 130141, 1963.

D Morin . Introduction to Classical Mechanics with Problems and Solutions. Cambridge, Cambridge, 2008.

E Noether . Invariante variationsprobleme. Nachrichten von der Gesellschaft der Wissenschaften zu Gttingen, Mathematisch-Physikalische Klasse, 1918:235257, 1918.

R Piessens , E. de Doncker-Kapenga , C.W. berhuber , and D.K. Kahaner . QUADPACK: A Subroutine Package for Automatic Integration. Springer, New York, 1983.

WH Press , SA Teukolsky , WT Vetterling , and BP Flannery . Numerical Recipes, 3rd Edition. Cambridge University Press, Cambridge, UK, 2007.

JC Sprott . Dynamical models of love. Nonlinear Dynamics, Psychology, and Life Sciences, 8(8):303313, 2004.

S Strogatz . Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering. Westview Press, Cambridge, UK, 2 edition, 2014.

B. Supiano . So what are you going to do with that degree? physics majors get that question, too. The Chronicle of Higher Education, 9 2018.

JR Taylor . Classical Mechanics. University Science Books, Sausalito, CA, 2005.

ST Thornton and JB Marion . Classical Dynamics of Particles and Systems. Thomson-Brooks/Cole, Belmont, CA, 5 edition, 2004.

CL Tokpah . The Effects of Computer Algebra Systems on Students Achievement in Mathemaics. dissertation, Kent State University, 2008.

U Wilensky and W Rand . An Introduction to Agent-Based Modeling. MIT Press, Cambridge, MA, 2015.

J Worthington . A Study of the Planar Circular Restricted Three Body Problem and the Vanishing Twist. PhD thesis, University of Sydney, 10 2012.