CHAPTER 2

Single-Particle Motion in One Dimension

In this chapter we will examine one-dimensional motion, i.e. motion along a line. It is sometimes the case that a particle's motion need only to be described along one direction. Furthermore, a careful study of one-dimensional motion will be a useful foundation for understanding more general motion in higher dimensions. In this chapter, we will will give several examples of solving Newton's Second Law, $F = ma$ in one dimension. We will consider several types of forces: both constant and those which depend on time $F(t)$, velocity $F(v)$ and position $F(x)$. In addition, we will discuss and demonstrate two different uses of computers: how to use computer algebra systems (CAS) to obtain the analytical solutions of Newton's Second Law, and how to obtain numerical solutions of ordinary differential equations (ODE) using software packages and by using the Euler Method.

2.1 EQUATIONS OF MOTION

To begin our study of one-dimesional motion, we first need to make some assumptions about the object whose motion we are examining. One fundamental assumption in this chapter is that the object being studied is a point particle. In order to mathematically describe the motion of a particle under the influence of a force, we need to find the particle's *equations of motion*. The equations of motion of a particle are the equations which describe its position, velocity, and acceleration as a functions of time. Equations of motion can be in the form of algebraic equations, or in the form of differential equations.

As we will see, the equations of motion of a particle can be found by solving Newton's Second Law as a differential equation. In this chapter we will focus on one-dimensional motion, where the force vector and the particle's displacement are along the same line (but not necessarily in the same direction-the direction could be horizontal or vertical). Because all vectors in a given problem lay along the same line, we drop the vector notation in all the equations. A negative sign between two quantities will denote vectors that lay in opposite directions along the same line.

Newton's Second Law in one dimension is:

$$F = ma \tag{2.1.1}$$

Recall that acceleration is the first derivative of velocity v with respect to time and the second derivative of displacement x with respect to time. To solve for the equations of motion, we will think of (2.1.1) as a differential equation, by re-writing (2.1.1) in the following ways:

$$F = m\frac{d^2x}{dt^2} \tag{2.1.2}$$

$$F = m\frac{dv}{dt} \tag{2.1.3}$$

$$F = mv\frac{dv}{dx} \tag{2.1.4}$$

Notice that each of the above equations is a differential equation which can be solved once the net force F acting on the particle is specified. Each of the above equations yields a different equation of motion: (2.1.2) can be solved for $x(t)$, (2.1.3) for $v(t)$, (2.1.4) for $v(x)$. Equation (2.1.4) comes from the chain rule:

$$m\frac{dv}{dt} = m\frac{dv}{dx}\frac{dx}{dt} = mv\frac{dv}{dx} \tag{2.1.5}$$

Throughout the rest of this chapter, we will solve (2.1.1) for several different cases where forces are constant ($F = F_0$), time-dependent ($F = F(t)$), velocity-dependent ($F = F(v)$), and position-dependent ($F = F(x)$). However, before solving (2.1.1), we will make a few comments about differential equations in general.

2.2 ORDINARY DIFFERENTIAL EQUATIONS

Simply put, an ordinary differential equation (ODE) is an equation which contains the derivative of a function. For example,

$$\frac{dx}{dt} = 7 \tag{2.2.1}$$

is a differential equation which says that $x(t)$, the solution to the differential equation, is a function whose first derivative is equal to 7. Of course we know that $x(t) = 7t$ is a solution that works. However, there are an infinite number of other solutions as well, since we can add a constant to $x(t)$ and still have a solution to our ODE. Hence, the so-called general solution is $x(t) = 7t + c$, where c is a constant. This differential equation is simple enough to solve. However, most ODEs are not that simple, and many cannot be solved at all. Before discussing how to solve an ODE, let's first point out a few things about our example.

1. Equation (2.2.1) is called a *first-order ODE*, because the highest derivative in the equation is a first derivative. In general, an n^{th}-order ODE is an ODE whose highest derivative is an n^{th} derivative.

2. The ordinary derivative implies that x is a function of only one variable t, which is the variable of differentiation.

3. The number of arbitrary constants in the general solution of (2.2.1) is equal to the order of the ODE.

To solve (2.2.1), we needed to separate the variables of the equation. Colloquially speaking, this means getting all the terms with x on one side of the equation and all the terms that are either constant or depend on t on the other. This process is called *separation of variables* and is performed by treating the derivative as a fraction, and multiplying both sides of (2.2.1) by dt,

$$dx = 7dt \tag{2.2.2}$$

$$\int dx = \int 7dt \tag{2.2.3}$$

$$x = 7t + c \tag{2.2.4}$$

To solve (2.2.1), we carried out the integral after separating out variables. Note that both integrals would produce constants of integration, but since both are constants, we can combine them into one arbitrary constant. You can double check the solution by computing the derivative of $7t + c$, to check that it satisfies (2.2.1).

What happens in the case of second order ODEs? Second order ODEs are more common in physics. There are many techniques to solve them, but we will demonstrate only one. Consider the ODE,

$$\frac{d^2x}{dt^2} = 7 \tag{2.2.5}$$

Separation of variables does not make sense here, because we typically do not integrate terms like d^2x. However, we can define a new variable v, such that, $v = dx/dt$. Then (2.2.5) becomes,

$$\frac{dv}{dt} = 7 \tag{2.2.6}$$

which we know gives the answer $v(t) = 7t + c_1$, where c_1 is the constant of integration. However we want $x(t)$, so we use $v = dx/dt$:

$$\frac{dx}{dt} = 7t + c_1 \tag{2.2.7}$$

$$\int dx = \int (7t + c_1)\, dt \tag{2.2.8}$$

$$x(t) = 3.5t^2 + c_1 t + c_2 \tag{2.2.9}$$

where c_2 is the constant of integration obtained by performing the above integral. Hence, we pick up an additional constant of integration in our solution, giving two arbitrary constants for the solution of the second order ODE (2.2.5). Loosely speaking, we see that the number of arbitrary constants in the solution of an n^{th}-order ODE is equal to n, because we need to do n integrations in order to solve the equation, and each integration produces an arbitrary constant.

Next, we return to (2.2.1). Suppose (2.2.1) was an equation we wanted to use in order to find the position of a particle as a function of time. The infinite number of solutions is not helpful. Which solution describes the actual path taken by the particle? In order to specify the *particular solution* for an ODE, we need to include *initial conditions*, the value of our function at a particular time (normally at $t = 0$). Suppose we know that at $t = 0$, the particle is at a position, $x(0) = 3$. Then we can solve for the arbitrary constant by inserting the initial condition into our general solution,

$$x(0) = (7)\,(0) + c = 3 \tag{2.2.10}$$

which gives $c = 3$. Our particular solution is then, $x(t) = 7t + 3$. A different initial condition will give a different particular solution. Now suppose we wanted to find a particular solution

to (2.2.5), in that case one initial condition will not be enough because it will leave one arbitrary constant. Hence, we will need to specify both $x(t)$ and dx/dt at a particular time (usually $t = 0$). Suppose that $x(0) = 3$ and $v(0) = 1$, where $v = dx/dt$. Then we have:

$$x(0) = 3.5\,(0)^2 + c_1\,(0) + c_2 = 3 \tag{2.2.11}$$

$$v(0) = 7\,(0) + c_1 = 1 \tag{2.2.12}$$

where (2.2.12) is the derivative of the general solution evaluated at $t = 0$. The result here is that $c_1 = 1$ and $c_2 = 3$, and the particular solution is $x(t) = 3.5t^2 + t + 3$. In summary, in order to solve for the particular solution of an n^{th}-order ODE, we need n initial conditions. In addition, we can also specify x using knowledge of the value of x at two different times, as opposed to knowing initial values of x and its first derivative. In classical mechanics, it is most common to know the initial conditions of the position and velocity. However, in other fields, such as electromagnetism and thermodynamics, it is often more common to know the value of a function, say the temperature, at two different locations. In this case, we have what is known as a *boundary value problem*, and the conditions that provide the constants of integration are known as *boundary conditions*.

There are many other properties of ODEs such as linear superposition, that we will explore in this book as we need them. For now, we know enough about ODEs to get started. Let's get back to the physics.

2.3 CONSTANT FORCES

Consider the case where a constant net force, $F = F_0$, acts on a particle constrained to move along a line. In this case, Newton's Second Law (2.1.1) gives:

$$\frac{dv}{dt} = \frac{F_0}{m} = a \tag{2.3.1}$$

where a is a constant because F_0 is constant. You may recall (2.3.1) from Example 1.1. Equation (2.3.1) is an example of a first order differential equation. We can solve this through the process of separation of variables and then integrating the resulting equation. The first step in separating variables is to multiply both sides of the equation by dt:

$$dv = a dt \tag{2.3.2}$$

which can be integrated to yield,

$$v(t) = \int a\,dt = at + c_1 \tag{2.3.3}$$

where c_1 is the constant of integration. The constant, c_1, can be found using initial conditions of $v(t_0) = v_0$, where v_0 is the initial velocity at the initial time t_0. When $t_0 = 0$, (2.3.3) gives $v(0) = c_1$ or $c_1 = v_0$. Therefore, the solution to the differential equation, (2.3.1) is,

$$v(t) = v_0 + at. \tag{2.3.4}$$

Before moving forward, we should mention that there is an alternate method for finding (2.3.3), which can be found by explicitly inserting the initial and final conditions when integrating (2.3.2):

$$\int_{v_0}^{v(t)} dv' = \int_{t_0}^{t} a dt' \tag{2.3.5}$$

$$v(t) - v_0 = a(t - t_0) \tag{2.3.6}$$

where we have introduced primes to the variables of integration in order to distinguish them from the limits of integration. Notice that the lower limit in the left-hand side of (2.3.5) corresponds to the value of $v(t)$ when $t = t_0$, the lower limit of the right-hand side of (2.3.5), and with similar considerations for the upper limits. It is very important that the limits match on both sides of the equation.

Next, we can get an equation for $x(t)$ by writing $v = dx/dt$:

$$\frac{dx}{dt} = at + v_0, \tag{2.3.7}$$

and separating variables we obtain:

$$\int_{x_0}^{x(t)} dx' = \int_{t_0}^{t} (at' + v_0) dt' \tag{2.3.8}$$

Notice how all of the time-dependent and constant terms are on the same side of the equation. If we had subtracted v_0 from both sides of the equation before integrating, we would have $dx - v_0$ which does not make sense because $\int v_0$ is meaningless without a variable of integration. In this context, separation of variables always involves multiplication and division, not addition and subtraction. Choosing $t_0 = 0$ and performing the integral results in:

$$x(t) = \frac{1}{2} a t^2 + v_0 t + x_0 \tag{2.3.9}$$

Together, equations (2.3.4) and (2.3.9) are the only equations you need to know, in order to solve for the position and velocity of a particle moving in one-dimension and experiencing a constant net force.

Finally, if we want $v(x)$ we can solve (2.1.4):

$$v \frac{dv}{dx} = \frac{F_0}{m} \tag{2.3.10}$$

$$v dv = a dx \tag{2.3.11}$$

$$\int_{v_0}^{v} v' dv' = a \int_{x_0}^{x} dx' \tag{2.3.12}$$

$$v^2 - v_0^2 = a(x - x_0) \tag{2.3.13}$$

where $a = F_0/m$ was used in (2.3.11). We could have also obtained this result by eliminating t between equations (2.3.3) and (2.3.9). The box below summarizes all of the constant force equations, sometimes called the *Kinematic Equations*. These are equations that you should memorize.

Kinematic Equations (a=constant)

$$v(t) = v_o + at \tag{2.3.14}$$

$$x(t) = x_o + v_o t + \frac{1}{2}at^2 \tag{2.3.15}$$

$$v(x)^2 = v_o^2 + 2a\,(x - x_o) \tag{2.3.16}$$

In the case of a freely falling particle near the surface of the Earth, we use $a = -g = -9.8\,\mathrm{m/s^2}$ (assuming "down" is in the negative direction), where g is the acceleration due to gravity. The Kinematic Equations become:

$$v(t) = v_0 - gt \tag{2.3.17}$$

$$y(t) = y_0 + v_0 t - \frac{1}{2}gt^2 \tag{2.3.18}$$

$$v(y)^2 = v_0^2 - 2g(y - y_0) \tag{2.3.19}$$

We used y as the position variable, which is common when describing vertical motion. Notice that these are not different equations from the Kinematic Equations, they are simply the Kinematic Equations with a specific value of a. We now look at some well-known examples of situations in which the acceleration of the system is constant.

Example 2.1: The Atwood machine

The Atwood machine consists of two masses m_1, m_2 hanging over a pulley as shown in Figure 2.1. Assume that $m_2 > m_1$. Find the acceleration a of the masses and the tension of the string.

Solution:

We can derive an equation for the acceleration by using force analysis. If we consider a massless, inextensible string and an ideal massless pulley, the only forces we have to consider are: tension (T) and weight of the two masses ($W_1 = m_1 g$ and $W_2 = m_2 g$). The tension acting on each mass will be the same, because the tension is constant throughout the string. Because the string is inextensible, the magnitude of the acceleration of each mass will also be the same although they'll be in opposite directions.

To find the acceleration we need to consider the forces affecting each individual mass. First, we need to define a frame of reference. In this case, we will choose $+y$ is downwards so that the heavier mass, m_2, moves in a positive direction. Newton's Second Law can be used to derive a system of equations for the acceleration a.

Forces affecting m_1 : $m_1 g - T = -m_1 a$
Forces affecting m_2 : $m_2 g - T = m_2 a$

The upward motion of m_1 has been made explicit by including a minus sign on the right-hand side of the equation describing the forces affecting m_1. Adding the two previous equations we obtain

$$m_1 g + m_2 g = -m_1 a + m_2 a \tag{2.3.20}$$

and after solving for the acceleration:

$$a = g\frac{m_2 - m_1}{m_1 + m_2} \tag{2.3.21}$$

Once a has been found, we can find the position $x(t)$ and velocity $v(t)$ of the masses using (2.3.15) and (2.3.14). Note that for the position and velocity of m_1, we would need to insert $-a$ in the Kinematic Equations.

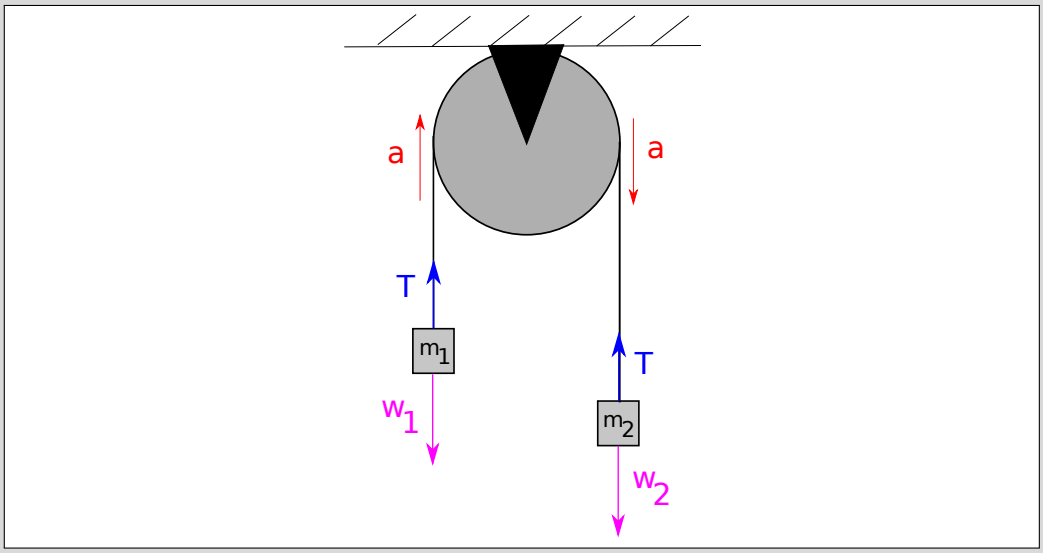


Figure 2.1: The free body diagrams of the two hanging masses of the Atwood machine.

To find the tension in the string, substitute (2.3.21) into, $-m_1 a = m_1 g - T$, which results in:

$$T = \left(\frac{2m_1 m_2}{m_1 + m_2}\right) g \tag{2.3.22}$$

Of course we could have also substituted (2.3.21) into the equation describing the forces acting on m_2 and we have obtained the same result for T.

Example 2.2: Motion on an inclined plane

The mass m shown in Figure 2.2 is placed on an inclined plane with angle θ. The coefficient of friction between the mass and the plane is μ.If the mass is released from rest, find the position $x(t)$, velocity $v(t)$, and acceleration a.

Solution:

We will use the standard reference frame for inclined planes that you learned in your introductory physics course; hence we define $+x$ as downward, along the plane. The force of friction is written as $f = \mu N$ where N is the normal force between the mass and the inclined plane.

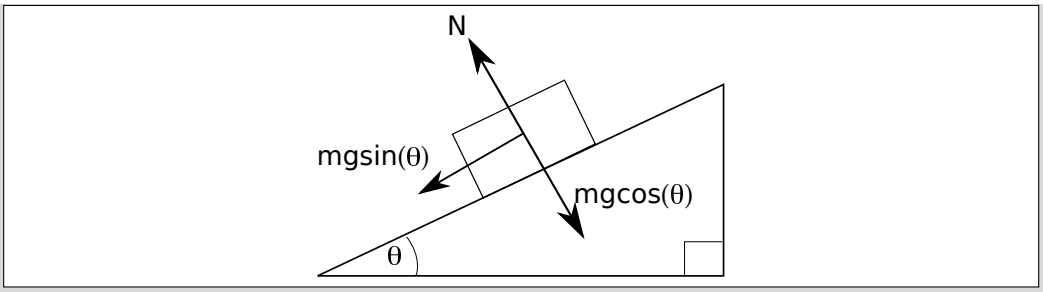


Figure 2.2: The forces on an incline plane.

The normal force here represents the force applied by the plane against the object (and vice versa). The magnitude of the normal force N can be calculated from the free body diagram, as shown in Figure 2.2, to be $N = mg\cos\theta$, where g is the acceleration due to gravity and θ is the angle of the inclined surface measured from the horizontal. The component of the weight $W = mg$ which acts along the direction of the plane is again found from the force diagram to be $F = mg\sin\theta$.

The total force along the x-axis (downward direction along the plane) is:

$$F_{total} = ma = mg\sin\theta - f = mg\sin\theta - \mu mg\cos\theta \tag{2.3.23}$$

and by dividing with the mass m we find the constant acceleration:

$$a = g\sin\theta - \mu g\cos\theta \tag{2.3.24}$$

This acceleration can then be used in (2.3.15) and (2.3.14) to get $x(t)$ and $v(t)$, respectively.

2.4 TIME-DEPENDENT FORCES

Now consider a case where a particle constrained to move along a line is experiencing a net force $F(t)$ which is dependent on time. Time dependent forces arise in a variety of applications, one common occurrence is when a sinusoidal external force acts on the system. In the cases where the net force $F = F(t)$, Newton's Second Law 2.1.1 gives:

$$\frac{dv}{dt} = \frac{F(t)}{m} \tag{2.4.1}$$

and by separating variables and integrating from $t' = t_o$ to t we find :

$$v(t) - v(t=0) = \frac{1}{m}\int_{t'=t_0}^{t} F(t')\,dt' \tag{2.4.2}$$

By using the initial conditions $v(t=0) = v_0$:

$$v(t) = v_0 + \frac{1}{m}\int_{t'=0}^{t} F(t')\,dt' \tag{2.4.3}$$

Once we know $v(t)$, we can find $x(t)$ by integrating $v = dx/dt$ and using the initial conditions $x(t_0) = x_o$ to obtain:

$$x(t) - x_0 = \int_{t'=0}^{t} v(t')\, dt' \tag{2.4.4}$$

Next, we will demonstrate an example of how to find the equations of motion for a particle experiencing a time-dependent force. We will demonstrate the solution both by-hand and using the CAS Mathematica. Refer to Chapter 1 for comments on how to use Mathematica to solve differential equations in closed form.

Example 2.3: A decreasing force, F(t)

The force acting on a mass m is decreasing with time according to $F(t) = F_0/(t^2+1)$ where t is the elapsed time. Find the position $x(t)$ and the velocity $v(t)$ with the initial conditions $x(0) = x_0 = 0$ and $v(0) = v_0 = 0$.

Solution:

This is the case of a time dependent force $F = F(t)$, so we can use (2.4.3) with $v_0 = 0$:

$$v(t) = v_0 + \frac{1}{m}\int_{t'=0}^{t} F(t')\, dt' = \frac{1}{m}\int_{t'=0}^{t} \frac{F_0}{t^2+1}\, dt = \frac{F_0}{m}\tan^{-1}(t) \tag{2.4.5}$$

Does our v(t) make sense? To check, we ask what happens when $t \to \infty$? In the limit of $t \to \infty$, $F \to 0$ and we would expect that v becomes constant. In our particular case, as $t \to \infty$, $\tan^{-1}(t) \to \pi/2$, and therefore, our solution $v(t)$ follows our expectation of the velocity approaching a constant. Next, we find $x(t)$ using (2.4.4) with $x_0 = 0$:

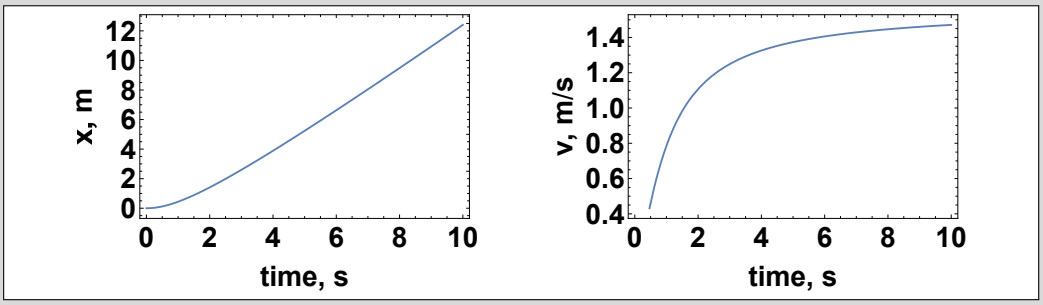
$$x(t) - x_0 = \int_{t'=0}^{t} v(t')\, dt' \tag{2.4.6}$$

$$= \frac{F_0}{m}\int_{t'=0}^{t} \tan^{-1}\left(t'\right)\, dt' \tag{2.4.7}$$

$$= \frac{F_0}{m}\left(t\, \tan^{-1}(t) - \frac{1}{2} ln\left(t^2+1\right)\right) \tag{2.4.8}$$

In this case, as $t \to \infty$, we know that $v \to$ *constant* and therefore, we would expect $x(t)$ to continue to grow, which is exactly what our solution does! We can also obtain these results and plot them by using the Mathematica commands *DSolve* and *Plot*, as shown below. In Mathematica, a semi-colon at the end of a line means suppress the output. Therefore, only lines without a semi-colon produce an output to the screen. The output of the first line starting with *ODEsolution* is suppressed, while the output of the line starting with *position*, is not. The output appears immediately below the line of code, and matches the formulas above.

```
ODEsolution = DSolve[{m * x''[t] == F0/t^2+1, x[0] == 0, x'[0] == 0}, x[t], t];

position = x[t]/.ODEsolution[[1]]

OUTPUT: F0(2tArcTan[t]-Log[1+t^2])/2m

velocity = D[position, t]

OUTPUT: F0ArcTan[t]/m

F0 = 1; m = 1;

GraphicsGrid[{{Plot[position, {t, 0, 10}, Frame → True, Axes → False, FrameLabel →
{"time, s", "x, m"}, BaseStyle → {FontSize → 24}], Plot[velocity, {t, 0, 10}, Frame →
True, Axes → False, FrameLabel → {"time, s", "v, m/s"}, BaseStyle → {FontSize →
24}]}}]
```



Notice that sometimes curly brackets (*i.e.* { }) are used in Mathematica, while other times square brackets (*i.e.* [ ]) are used. In Mathematica, square brackets are used to denote the argument of a function or command. For example, we would write $x(t)$ as a hand-written solution to a differential equation. The notation $x(t)$ uses parentheses to denote the fact that the variable t is the argument of the function x. In Mathematica, we would write the function as $x[t]$. Notice, we used the notation $x[t]$ in the command *DSolve* in order to tell Mathematica that x is a function of t. Also notice that the arguments of commands are also placed in square brackets. The command D has the arguments *position* and t which are inside square brackets. The D command takes a derivative of the first argument (in this case, the equation stored in the variable *position*) with respect to the argument in the second position (in this case, t). Curly brackets in Mathematica are generally used to denote a range. Notice in the *Plot* command the second argument is $\{t, 0, 10\}$ which is telling Mathematica to plot the function stored in the variable *position* as a function of time (the t in the first argument of the curly brackets) starting at $t = 0$ (the second argument in the curly brackets) and ending at $t = 10$ (the third argument in the curly brackets). Different programming languages use square brackets, curly brackets, and parentheses in different ways. Mathematica uses parentheses mainly to group elements together (as you would when hand writing mathematics), however Python uses parentheses for many of the same things for which Mathematica uses square and curly brackets. Once one knows several programming languages, it is common to get the usage of parentheses, curly brackets, and square brackets mixed up!

2.5 AIR RESISTANCE AND VELOCITY-DEPENDENT FORCES

You may recall from your introductory physics course that when studying the motion of a particle in free fall, air resistance was often ignored. As you may know, air resistance depends on the velocity of the object moving through the air. Hence, air resistance is a velocity-dependent force. Here we will learn how to address air resistance when describing the motion of a particle. It should be noted however, that although we focus on air resistance as an example of a velocity-dependent force, most of our comments will be applicable to any velocity-dependent force.

When speaking about an object moving through the air, two forces are often described, drag and lift. The *drag* force is the component of the air resistance which is in the opposite direction of the object's velocity (direction of motion). *Lift* forces are the component of the air resistance which is perpendicular to the object's motion. In this section, we will consider cases where the lift is negligible. We will also only consider non-rotating bodies. A rotating body moving through air will experience the *Magnus effect* where the spinning motion of the object drags air faster on one side than the other. The resulting pressure difference on either side of the object causes the object to curve away from its flight path. This is what allows a baseball pitcher to throw a curve ball. There are many interesting videos online which demonstrate the Magnus effect.

In general, the air resistance force studied in this chapter takes the mathematical form,

$$\mathbf{f} = -f(v)\hat{\mathbf{v}} \tag{2.5.1}$$

where the unit vector $\hat{\mathbf{v}} = \mathbf{v}/v$ is along the direction of the object's velocity (or motion) and the minus sign is included to explicitly show that the direction of the air resistance force is opposite of the direction of the velocity. The magnitude of the air resistance $f(v)$ is a function of velocity which generally takes the form,

$$f(v) = bv + cv^2 \tag{2.5.2}$$

and therefore has both a linear and a quadratic component in v. For spheres in air the values of the constants b and c are [Taylor(2005)]:

$$\begin{aligned} b &= 1.6 \times 10^{-4} D \quad \text{N·s/m} \\ c &= 0.25 D^2 \quad \text{N·s}^2/\text{m}^2 \end{aligned} \tag{2.5.3}$$

where D is the diameter of the sphere in meters. Fortunately, one usually doesn't need to include both the linear and quadratic terms for air resistance. In order to determine which term, linear or quadratic, is the most important in a given situation, one can compute the ratio:

$$\gamma = \frac{cv^2}{bv} = \frac{0.25 D^2 v^2}{1.6 \times 10^{-4} D v} = 1.6 \times 10^3 D v \tag{2.5.4}$$

Note that v should be in meters per second, and D in meters. If $\gamma >> 1$, then the quadratic term dominates and the linear term can be neglected. If $\gamma << 1$, then the linear term dominates and the quadratic term can be neglected. However, if $\gamma \approx 1$, then both the linear and quadratic terms need to be included. So for a 0.22 m soccer ball, the quadratic term dominates for speeds approximately greater than 0.003 m/s (or 3 mm/s), while for lower speeds the linear term dominates. Both linear and quadratic terms would need to be included in the air resistance formula if the soccer ball is traveling near 3 mm/s.

To obtain equations for position and velocity as functions of time, we will consider a generic form for $F(v)$. Newton's Second Law (2.1.1) gives:

$$\frac{dv}{dt} = \frac{F(v)}{m} \tag{2.5.5}$$

Rearranging the above equation gives:

$$dt = m\frac{dv}{F(v)} \tag{2.5.6}$$

and by integrating from $t' = t_0$ to t , and using the initial conditions $v(t_0) = v_0$ we find :

$$t - t_0 = m\int_{v'=v_0}^{v} \frac{dv'}{F(v')} \tag{2.5.7}$$

By solving (2.5.7), we can find $v(t)$, the velocity as a function of time. Once we know $v(t)$, we can again use (2.4.4) to find $x(t)$ with the initial condition $x(t_0) = x_0$.

We can follow a similar procedure for finding $v(x)$. Newton's Second Law gives:

$$mv\frac{dv}{dx} = F(v) \tag{2.5.8}$$

After rearranging, we obtain:

$$x - x_0 = m\int_{v'=v_0}^{v} \frac{v'dv'}{F(v')} \tag{2.5.9}$$

Next, we will, show some examples involving velocity-dependent forces.

Example 2.4: Air resistance varying linearly with speed

The force of air resistance on a mass m depends linearly on the velocity v according to $F(v) = -bv$. Find $x(t)$ and $v(t)$ with the initial conditions $v(0) = v_0$ and $x(0) = x_0$.

Solution:

We use (2.5.7) to find $x(t)$ with $t_0 = 0$:

$$t = m\int_{v'=v_0}^{v} \frac{dv'}{F(v')} = -\frac{m}{b}\int_{v'=v_0}^{v} \frac{dv'}{v'} = -\frac{m}{b}\ln\left(\frac{v'}{v_0}\right) \tag{2.5.10}$$

Solving for $v(t)$ we obtain:

$$v(t) = v_0 e^{-\frac{bt}{m}} \tag{2.5.11}$$

We also find x(t) from (2.4.4):

$$x(t) - x_0 = \int_{t'=t_0}^{t} v(t')\,dt' = \int_{t'=t_0}^{t} v_0 e^{-\frac{bt'}{m}}\,dt' \tag{2.5.12}$$

which gives:

$$x(t) = x_0 + \frac{mv_0}{b}\left(1 - e^{-\frac{bt}{m}}\right) \tag{2.5.13}$$

Here is the solution using Mathematica:

ODEsolution = DSolve[{$m * x''[t] == -b * x'[t], x[0] ==$ xo, $x'[0] ==$ vo}, $x[t], t$];

position = Collect[$x[t]$/.ODEsolution[[1]], {vo, m}]

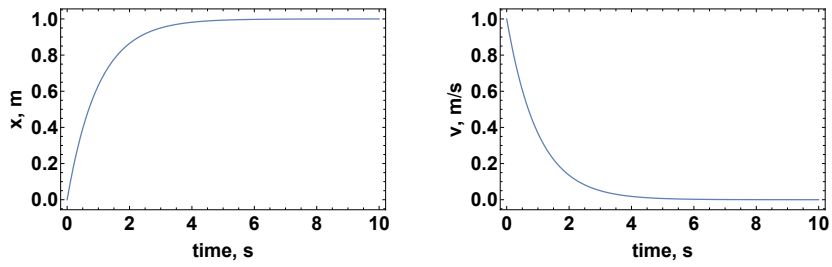
OUTPUT: $\frac{m\text{vo}\left(1-e^{-\frac{bt}{m}}\right)}{b} + \text{xo}$

velocity = D[position, t]

OUTPUT: $e^{-\frac{bt}{m}}\text{vo}$

vo = 1; xo = 0; m = 1; b = 1;

GraphicsGrid[{{Plot[position, {t, 0, 10}, Frame → True, Axes → False, FrameLabel → {"time, s", "x, m"}, BaseStyle → {FontSize → 24}, PlotRange → All],
Plot[velocity, {t, 0, 10}, Frame → True, Axes → False, FrameLabel → {"time, s", "v, m/s"}, BaseStyle → {FontSize → 24}, PlotRange → All]}}]



The *Collect* command gathers together terms that involve the same powers of the objects in the curly brackets (second argument of *Collect*). It is a useful command for simplifying algebraic terms and for identifying coefficients.

Example 2.5: Air resistance in the presence of gravity

Consider the case of a falling object near the surface of the Earth. Suppose in this case, the air resistance is linear. Then the net force on the falling body depends on the velocity v according to

$$F(v) = -bv + mg \tag{2.5.14}$$

where we have defined $+y$ to be vertically downward so that the body's motion is in the positive direction. Find the displacement $y(t)$ and the velocity $v(t)$, with the initial conditions $v(0) = v_0$ and $y(0) = y_0$.

Solution:

We use (2.5.7) to find $v(t)$ with $t_0 = 0$:

$$t = m\int_{v'=v_0}^{v} \frac{dv'}{-bv' + mg} = -\frac{m}{b}\ln\left(\frac{\frac{mg}{b} - v}{\frac{mg}{b} - v_0}\right) \tag{2.5.15}$$

Note that we have factored out the constant b from the denominator of the integral before performing the integration.

Solving for $v(t)$ we obtain:

$$v(t) = \frac{mg}{b} + \left(v_0 - \frac{mg}{b}\right) e^{-\frac{bt}{m}} \tag{2.5.16}$$

We also find $y(t)$ from (2.4.4):

$$y(t) - y_0 = \int_{t'=t_0}^{t} v(t')\,dt = \int_{t'=t_0}^{t} \left[\frac{mg}{b} + \left(v_0 - \frac{mg}{b}\right) e^{-\frac{bt'}{m}}\right] dt' \tag{2.5.17}$$

which gives after integrating:

$$y(t) = y_0 + \frac{mg}{b}t + \left(\frac{m^2 g}{b^2} - \frac{mv_0}{b}\right) e^{-\frac{bt}{m}} \tag{2.5.18}$$

When $t \to \infty$ we can substitute $e^{-\frac{bt}{m}} \to 0$ in (2.5.16) to obtain the *terminal velocity*:

$$v_{\text{terminal}} = \frac{mg}{b} \tag{2.5.19}$$

Hence, our solution matches the expected physics. As the object falls, its velocity increases due to the force of gravity. However, at some point, the object is moving fast enough that the drag force equals that of the object's weight. At that point, the net force on the object is zero, and the velocity remains at a constant value called the terminal velocity, v_{terminal} for the duration of its fall.

Figure 2.3 shows the plots of $x(t)$ and $v(t)$ found in Example 2.5. Notice that both plots show the velocity of the object coming to a constant value as the object falls for a long period of time.

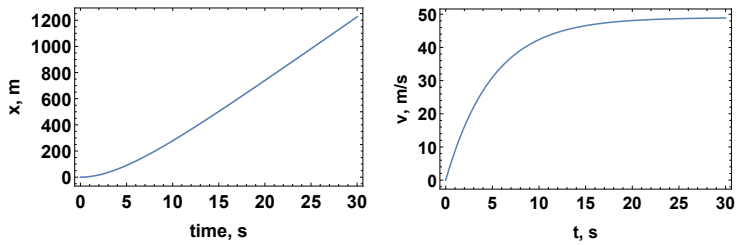


Figure 2.3: The plots of $x(t)$ and $v(t)$ from Example 2.5 using Mathematica and the numerical values $m = 1.0$ kg, $b = 0.2$ Ns/m, $x_0 = 0$ m and $v_0 = 0$ m/s.

Example 2.6: Air resistance proportional to v^2

A more realistic air resistance force is proportional to the square of the velocity v according to $F(v) = -cv^2 + mg$. Find $x(t)$ and $v(t)$ with the initial conditions $v(0) = 0$ and $x(0) = 0$.

Solution:

The terminal velocity is found in this case by setting $F = -cv_{\text{terminal}}^2 + mg = 0$:

$$v_{\text{terminal}} = \sqrt{\frac{mg}{c}} \tag{2.5.20}$$

For simplicity's sake, we will denote v_{terminal} as v_t. Next, we use (2.5.7) to find $v(t)$ by integrating from the initial velocity v_o to the final velocity v_f:

$$\begin{aligned} t =& m\int_0^{v_f} \frac{dv'}{-cv'^2 + mg} \\ =& \frac{m}{c}\int_0^{v_f} \frac{dv'}{v_t^2 - v'^2} \\ =& \frac{m}{cv_t}\tanh^{-1}\left(\frac{v_f}{v_t}\right) \end{aligned} \tag{2.5.21}$$

where we used integral tables to perform the integration (Note that a CAS could be used as well, see the code below). We can then solve for v_f to yield:

$$v_f(t) = v_t \tanh\left[\frac{cv_t}{m}t\right] \tag{2.5.22}$$

We also find $x(t)$ from (2.4.4):

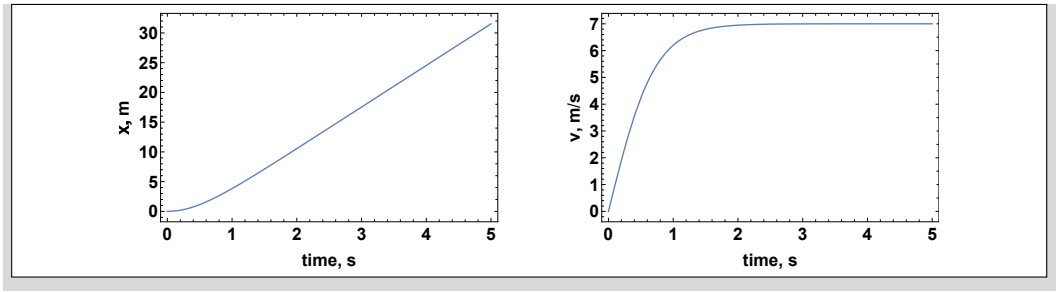
$$x(t) - x_0 = \int_0^t v(t')\,dt' = \int_0^t v_t \tanh\left[\frac{cv_t}{m}t'\right]dt' \tag{2.5.23}$$

which with $x_0 = 0$ gives:

$$x(t) = \frac{m}{c}\ln\left(\cosh\left(\frac{cv_t}{m}t\right)\right) \tag{2.5.24}$$

In the code below, we used Mathematica to perform some of the calculus needed to solve the problem.

```
integral = Assuming[(vf > 0)&&(vt > 0)&&(vf < vt), Integrate[1/(vt^2 - v^2), {v, 0, vf}]]

OUTPUT: ArcTanh[vf/vt]/vt

velocity = vt*Tanh[c*t*vt/m];

position = Integrate[velocity, t]

OUTPUT: m Log[Cosh[c t vt/m]]/c

m = 1; g = 9.8; c = 0.2; vt = Sqrt[m*g/c];

GraphicsGrid[{{Plot[position, {t, 0, 5}, Frame -> True, Axes -> False, FrameLabel ->
{"time, s", "x, m"}, BaseStyle -> {FontSize -> 20}, PlotRange -> All],
Plot[velocity, {t, 0, 5}, Frame -> True, Axes -> False, FrameLabel ->
{"time, s", "v, m/s"}, BaseStyle -> {FontSize -> 20}, PlotRange -> All]}}]
```



In Example 2.6, we used the command *Integrate* to perform the integrals. When you perform integrals you may not always think about the signs of variables. For example you know that the terminal velocity is positive. However, *Mathematica* (and other CAS programs) makes no such assumptions. In order to perform the integral, and get the above result, we needed to tell Mathematica that both the final velocity (vf in the code), and terminal velocity, (vt in the code) are positive, and that $vf < vt$. Physically, this is making the assumptions: $m, g, k, t, v > 0$ and $mg > kv^2$ (i.e. the air resistance can not exceed the weight $W = mg$). The symbol && represents the logical operator AND. These assumptions are provided in the *Assuming* command in the above code. So by placing the command *Integrate* inside the *Assuming* command, we are telling Mathematica to integrate with the listed assumptions. Notice that in the first line of code, the command *Integrate* is used with a second argument $\{v, 0, vf\}$, while the position calculation has the simple second argument of t. In the first case, we are asking Mathematica to perform a definite integral with variable of integration, v, and with a lower limit of $v = 0$ and an upper limit of $v = vf$. In the second case, we are asking Mathematica to perform an indefinite integral with variable of integration t.

A comparison between the velocity of the falling body in the cases of linear air resistance $F(v) = -bv + mg$ and quadratic air resistance $F(v) = -cv^2 + mg$ is shown in Figure 2.4 where $m = 1.0\,\text{kg}$, $b = 0.2\,\text{Ns/m}$, and $c = 0.2\,\text{Ns}^2/\text{m}$. The velocity on the y-axis has been normalized by dividing with the corresponding terminal velocity. Notice that the quadratic air resistance (dashed line) leads to the object obtaining terminal velocity in a significantly shorter time than linear air resistance (solid line). This is not surprising, the magnitude of the quadratic drag force will be higher than the linear force for a given velocity, v.

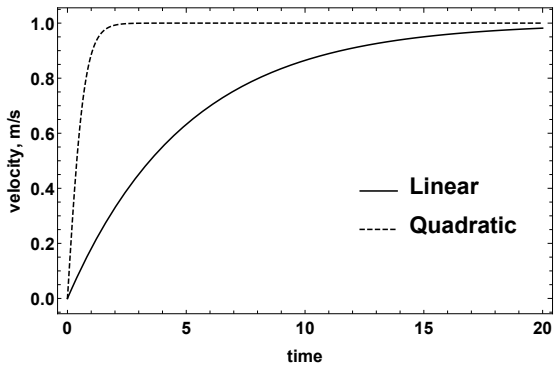


Figure 2.4: Comparison of the velocity of falling body in the presence of linear and quadratic air resistance. The two curves have been scaled, so that they produce the same result as $t \to \infty$.

2.6 POSITION-DEPENDENT FORCES

In the cases where the force is a function of the position $F = F(x)$, we can use Newton's Second Law in the form

$$F(x) = mv\frac{dv}{dx} \tag{2.6.1}$$

By separating the variables v and x, and using the initial conditions $x = x_0$ and $v = v_0$ when $t = 0$, we obtain:

$$\int_{x_0}^{x} F(x')dx' = \int_{v_0}^{v} mv'dv' = \frac{1}{2}mv^2 - \frac{1}{2}mv_0^2 \tag{2.6.2}$$

Solving for the velocity $v(x)$ as a function of the position x:

$$v\,(x) = \frac{2}{m}\sqrt{\int_{x_0}^{x} F(x')dx' + \frac{1}{2}mv_0^2} \tag{2.6.3}$$

By substituting $v = dx/dt$, separating the variables, and integrating with the initial condition $x = x_o$ when $t = t_o$, we obtain the relationship between position x and time t:

$$t - t_0 = \frac{m}{2}\int_{x_0}^{x}\left[\frac{dx'}{\sqrt{\int_{x_0}^{x} F(x')dx + \frac{1}{2}mv_0^2}}\right] \tag{2.6.4}$$

After performing the integral in equation (2.6.4), one would then try to invert this formula to find $x(t)$, something that is not always easy or possible to do. Often, one relies on computer methods in such cases.

The next example will involve one of the most important problems in all of classical mechanics, simple harmonic motion (SHM). In fact, we will devote a whole chapter to it! As we will see in a later chapter, SHM is a common model used for small amplitude oscillations. Although SHM is often thought of as a "mass on a spring," it is also a useful model for other small amplitude oscillations including pendulums, small amplitude water waves, and loudspeakers-to name a few. Besides presenting the SHM, the next example also includes a demonstration of using Python to find and plot the solution to a differential equation in closed form. Python's SymPy package was used in Chapter 1, to solve a first order differential equation. Here we will demonstrate the solution of a second order differential equation using SymPy.

Example 2.7: Simple harmonic motion

A mass m is under the influence of a position dependent force, $F = F(x) = -kx$, which is proportional to the distance x the mass is from an equilibrium located at $x = 0$. The mathematical form of F is, of course, the well-known Hooke's Law for springs. Find $x(t)$ and $v(t)$ with the initial conditions $v(0) = 0$ and $x(0) = x_0 \neq 0$ at $t = 0$.

Solution:

In this case (2.6.3) becomes:

$$v = \frac{2}{m}\sqrt{-\int_{x_0}^{x} kx'dx' + \frac{1}{2}mv_0^2} = \frac{2}{m}\sqrt{\frac{1}{2}k\left(x_0^2 - x^2\right)} \tag{2.6.5}$$

which gives the velocity $v(x)$ as a function of the position x. Next we will find $x(t)$ by using (2.6.4):

$$t = \int_{x_0}^{x} \frac{dx'}{\sqrt{\frac{k}{m}(x_0^2 - x'^2)}} = \sqrt{\frac{m}{k}}\sin^{-1}\left(\frac{x}{x_0}\right)\Bigg|_{x_0}^{x} = \sqrt{\frac{m}{k}}\left[\sin^{-1}\left(\frac{x}{x_0}\right) - \frac{\pi}{2}\right] \tag{2.6.6}$$

or after some rearranging:

$$x(t) = x_0 \sin\left(\sqrt{\frac{k}{m}}t + \frac{\pi}{2}\right) = x_0\cos\left(\sqrt{\frac{k}{m}}t\right) \tag{2.6.7}$$

The result is that the mass oscillates about the equilibrium position x_0. Hence we find that the natural frequency ω of the oscillator is $\omega = \sqrt{\frac{k}{m}}$.

Algorithm 1 is the solution using the SymPy library in Python. Also note that while not commented-out, the lines starting with the word OUTPUT: are not code, and are included only to show the output from the line above. Do not include such lines in your own code.

The code used to solve Example 2.7 is similar to what we used in Chapter 1. After importing everything in the SymPy library (the * means "everything"), we specify that the variables m, k, t are real and positive. Python needs to be told which variables are to be treated as symbols and which variables are to be treated as functions. Next, the differential equation is defined using the variable *diffeq*. The *Eq* command creates an equation with the first argument as the left-hand side of the equation and the second argument as the right-hand side. Hence, that line of code is the same as writing, $m\ddot{x} + kx = 0$. Note that the command *diff*, represents the derivative of x with respect to time (two t's in the argument means second derivative). The next line solves the differential equation using *dsolve*. At the time of this writing, the *dsolve* command in Python cannot include initial conditions, unless one wants to obtain power series solutions to the differential equation. Therefore, the next seven lines of code insert the initial conditions. To solve for the initial conditions, we need to first isolate the right-hand side of the solution. Next, we identify *C1* and C2 as symbols, even though they appear in the SymPy-produced solution. The lines starting with *ic1* and *ic2* apply the initial conditions by setting $x(0) = x0$ and $\dot{x}(0) = 0$, respectively. Then the resulting two equations are solved for the two unknowns, and the solution is stored in *solution_ics*. Finally, we obtain the particular solution by substituting the values of *C1* and *C2*, stored in *solution_ics*, into the general solution.

Like in Chapter 1, the code needed to solve Example 2.7 in Python is more complicated than that used by Mathematica to solve previous examples in this chapter. Python needs to be told which terms are symbols for manipulation and which terms are functions. In addition, SymPy's *dsolve* currently has no way of including initial conditions into the solution of the ODE, so more than half of the lines in Example 2.7's algorithm are used to find the arbitrary constants and plug them into the solution. As we've seen with Mathematica, finding the constants using initial conditions is easy by using its *DSolve* command.

```
from sympy import *

x = Function('x')
t = Symbol("t",real=True,positive=True)
k = Symbol("k",real=True,positive=True)
m = Symbol("m",real=True,positive=True)

diffeq = Eq(m*x(t).diff(t, t) + k*x(t), 0)

solution = dsolve(diffeq,x(t))
print(solution.rhs)

OUTPUT: C1*sin(sqrt(k)*t/sqrt(m)) + C2*cos(sqrt(k)*t/sqrt(m))

equation = solution.rhs
C1,C2 = symbols('C1,C2')
xo = symbols('xo')
ic1 = Eq(equation.subs(t,0),xo)
ic2 = Eq(equation.diff(t).subs(t, 0), 0)
solution_ics = solve([ic1,ic2], (C1, C2))
print(solution_ics)

OUTPUT: {C1: 0, C2: xo}

particular = simplify(equation.subs(solution_ics))
print(particular)

OUTPUT: xo*cos(sqrt(k)*t/sqrt(m))
```

Algorithm 1: The Python code for Example 2.7.

We didn't need to specify that t, m, and k are real and positive. If we had only defined those variables as symbols, Python would have still solved the differential equation. However, Python would produce a cosh function instead of the cos function. The argument of the cosh function produced has a negative sign under a radical. Using the relationship, $cosh(ix) = cos(x)$ we would get the same answer we obtained by hand and by specifying the nature of the variables. It is not unusual to have to do additional algebra on results provided by CAS algorithms, in order to get a result that is useful. This is an additional reason not to forget your math skills and not to rely solely on the computer!

Not all differential equations can be easily solved in closed form and some can not be solved in closed form at all. In cases where closed-form solutions are not possible to obtain, we can solve the differential equation using a numerical method. In the next section, we will discuss a simple method of numerically solving ordinary differential equations.

2.7 NUMERICAL SOLUTIONS OF DIFFERENTIAL EQUATIONS

Sometimes differential equations are either very difficult to solve in closed form, or have no closed-form solution. In these cases, we rely on numerical algorithms. The truth is that as a physicist, you will likely run into more differential equations that require numerical solutions than those that can be solved by hand! Therefore, it is important to understand how to solve a differential equation using numerical methods.

Consider an ordinary differential equation (ODE) of the following form:

$$\frac{dx}{dt} = f(x) \tag{2.7.1}$$

where $f(x)$ is a well-behaved function of x. Note that here $f(x)$ is an arbitrary function of position, not the force acting on the system. What we are about to demonstrate works on differential equations encountered in any field, not just physics. To solve (2.7.1) numerically, we need to approximate the derivative on the left-hand side. The approximation of the derivative is done by removing the limit in Newton's definition of the derivative:

$$\frac{dx}{dt} \approx \frac{x(t+dt) - x(t)}{dt}. \tag{2.7.2}$$

If the value of dt is small enough, the right-hand side of the above equation will give a good approximation of the derivative. We can then think of solving the differential equation by finding the current value of $x = x(t+dt)$ by using past values, $x(t)$. This process is called Euler's method for solving differential equations. By inserting (2.7.2) into (2.7.1), we obtain the formula for Euler's method:

$$x(t+dt) = x(t) + dt\, f(x(t)). \tag{2.7.3}$$

We can interpret (2.7.3) as using the tangent line at the current value of x, $x(t)$, in order to approximate the next value of x, $x(t+dt)$. Over a short distance dt, we can expect that this local linear approximation to be a good method of finding $x(t+dt)$ from $x(t)$.

To implement Euler's algorithm, follow these steps in your program of choice:

1. Define dt to be a value for the time interval small enough to produce the correct solution.

2. Define an empty array x for the function which is the solution to the ODE. Some programming languages require you to define the length, or number of elements in the array, when you define the array. If that is the case, define the length of x to be

equal to t_{max}/dt (rounded up to the nearest integer) where t_{max} is the largest value of t for which you want to know $x(t)$.

3. Loop over an index i in order to enter values in the array using:

$$x[i] = x[i-1] + dt * f(x[i]) \tag{2.7.4}$$

The i^{th} element of the array x, denoted by $x[i]$, contains the value of the function x at $t = idt$. How do you know if the value of dt that you chose gives a good solution? It is sometimes helpful to run your code with a first guess of dt, and then repeat for a smaller value of dt. If the solution doesn't change using the new value of dt, then that is a good sign that you have a good value of dt and an accurate representation of the solution. Of course it is not a proof, but the method can work well as you try to get a sense of the solution to a differential equation. It is worth mentioning here that the errors in numerical methods such as the Euler method are well known, and although we will not go into them here, they are covered in typical numerical analysis texts such as [Press et al.(2007)Press, Teukolsky, Vetterling, and Flannery].

You might wonder, why not just use a very small value say, $dt = 0.000001$? One reason is that if you did that, then to find the solution out to $t = 1$, the loop would take 10^6 iterations! However a more important reason is that if dt is very small, then global truncation errors become an issue for your solution. Local truncation error is the error that occurs at each iteration of a numerical method. Global truncation error is the cumulative error caused by many iterations. Hence, small values of dt both increase computing time and can introduce more error. Furthermore, the Euler method can be unstable, meaning that the numerical solution can grow very large when the exact solution does not. If you can not obtain a solution with reasonable accuracy using Euler's method, then you will need to investigate higher order numerical solvers such as the 4th order Runge-Kutta method, which we will discuss in a later chapter. Even more sophisticated methods involve an adaptive value of dt.

You might be wondering how to solve second order ordinary differential equations (ODE), which are more common in mechanics due to Newton's Second Law. To solve a second order ODE, we break it up into two first order ODEs and use the Euler method for each one. Consider the ODE resulting from Newton's Second Law in the previous example of linear air resistance:

$$ma = -bv + mg \tag{2.7.5}$$

$$m\frac{d^2x}{dt^2} = -b\frac{dx}{dt} + mg \tag{2.7.6}$$

where we have written $a = d^2x/dt^2$ and $v = dx/dt$. To create two first order ODEs from (2.7.6), we treat the velocity as a separate variable:

$$\frac{dx}{dt} = v \tag{2.7.7}$$

then (2.7.6) becomes:

$$\frac{dv}{dt} = g - \frac{b}{m}v \tag{2.7.8}$$

The resulting equations for the Euler method are therefore:

$$\begin{aligned} x(t+dt) &= x(t) + dt\, v(t) \\ v(t+dt) &= v(t) + dt\left(g - \frac{k}{m}v(t)\right) \end{aligned} \tag{2.7.9}$$

In Algorithm 2 we use Python to write an Euler's method algorithm which solves Example 2.5. In the next few paragraphs, we will explain Algorithm 2 line-by-line.

The Python library NumPy is imported and used for convenience, although it is not necessary. We also import and use the Python library Matplotlib which is useful for plotting functions. Matplotlib is also used here to export the graphs of $x(t)$ and $v(t)$ to a file. Exporting output (either arrays or images) to a file is useful, so that the output of the program can be used in other places without having to rerun the original program. After importing the libraries, we then define variables in the next block of code. Notice that we are choosing the step size to be $dt = 0.01$ and we want to integrate equations from $t = 0$ to $t = 30.0$ (t_{max}). The variable *num_steps = 3000* is the number of steps of size 0.01 between 0 and 30 and will be used to determine the length of arrays.

After defining variables in Algorithm 2, we defined the second term in the right-hand side of each equation in (2.7.9) as the functions *dxdt* and *dvdt*, respectively. Although it is not necessary to define those functions, it makes the lines in the upcoming for-loop cleaner and easier to read. Readability is important when writing algorithms, so that others can follow your work and so that you can follow your own work, if you need to reuse the algorithm later.

Next, we need to define the arrays x and v which will contain our solutions $x(t)$ and $v(t)$, respectively. We need the arrays to have enough elements to cover the time range from $t = 0$ to 30. We create the arrays using the command *zeros* from the NumPy library. The *np* in front of the command *zeros* tells the Python interpreter that the command *zeros* comes from the NumPy library. The command *zeros* creates an array in which all elements are 0. The length of the array is determined by the argument of the command, in this case, the arrays x and v have a length *num_steps* + 1. This may seems strange, but we can explain it using Python's method for indexing arrays. Python begins counting using 0, so the first element of an array has an index of 0 (note that some languages, like Mathematica, gives the first element of an array the index of 1). There are 3000 steps of size 0.01 between $t = 0$ and $t = 30$ and we need one more additional step for the time $t = 0$. Hence, in order to span the time from 0 to 30 in steps of 0.01, we need 3001 elements in our arrays. By initially assigning all elements the value of zero, we have automatically included the initial condition as the first element of each array.

Next, we use a for-loop to perform Euler's method. The *for* command tells Python to start a for-loop. The contents of the for-loop (the two-indented lines below the for statement) are repeated multiple times. The statement, *for i in range(1,num_steps+1)*, tells the Python interpreter that the value of i starts at $i = 1$. With each iteration of the loop, the value of i increases by 1. The loop stops when i = *num_steps.* The *range* command produces a list of values from 1 up to, but not including, *num_steps* + 1. Therefore, the loop is iterated *num_step* times. In each iteration of the loop, we calculate a new element of each array x and v. Hence, in the first iteration of the loop $i = 1$ and we compute $x[1]$ and $v[1]$, and the value of i is changed to 2. In the next iteration, $i = 2$ and we find $x[2]$ and $v[2]$ and the value of i is changed to 3. This process continues until i has the value *num_steps + 1.* Note that the value of i is increased after $x[i]$ and $v[i]$ are computed, so once i = *num_steps + 1* the loop stops without computing *x[num_steps +1]* and *v[num_steps + 1].*

In order to plot the functions x and v we need to tell Python to which time each array element corresponds. The *time* variable contains a list of times from 0 to 30 in steps of 0.01 using the command *arange*, which is similar to the *for* command, ends one step before the second argument, hence the *tmax + dt* term. The algorithm ends by creating the necessary plots and saving them as an encapsulated postscript file.

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

#define constants, time step size, and integration time
m, k, g = 1.0, 0.2, 9.8
dt = 0.01
tmax = 30.0
num_steps = int(tmax/dt)

#define functions
def dxdt(vel):
    return vel

def dvdt(vel):
    return g - k/m * vel

#define initial conditions and arrays
x = np.zeros(num_steps+1)
v = np.zeros(num_steps+1)

#perform the Euler step algorithm
for i in range(1,num_steps+1):
    x[i] = x[i-1] + dt * dxdt(v[i-1])
    v[i] = v[i-1] + dt * dvdt(v[i-1])

time = np.arange(0,tmax+dt,dt)

fig, axes = plt.subplots(1, 2, figsize=(10,4))
axes[0].plot(time,x)
axes[0].set_xlabel("time, s")
axes[0].set_ylabel("x, m")
axes[1].plot(time,v)
axes[1].set_xlabel("time, s")
axes[1].set_ylabel("v, m/s")
fig.savefig("linear_air_resistance.eps")
```

Algorithm 2: The solution to Example 2.5 using a Euler's method algorithm.

The output of Algorithm 2 is shown in Figure 2.5. Notice that the results obtained from the *Python* program look similar to the results obtained from Mathematica. That is a good sign! When we plot both results on the same graph, we find that the graphs are actually identical. When developing your own numerical analysis programs, such as the Euler method algorithm, it is often wise to check the results of your new algorithm with another algorithm that you suspect works fine. In this case, the Mathematica *DSolve* command produced the same analytical form as we obtained by hand. Therefore, we verified Mathematica's result with our own calculation, and then verified our Euler method result (found using Python) with that of Mathematica's result. Ultimately, we can be confident that our Euler method program works well, at least in the case of this one particular ODE. Furthermore, our Euler method's success here makes us more confident of the results we would obtain by using our algorithm on new problems.

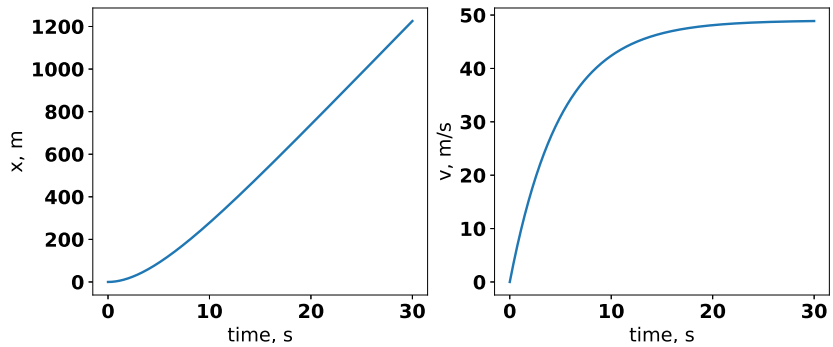


Figure 2.5: The result of Algorithm 2

The ability to solve differential equations numerically allows you to address interesting problems that you could not do by hand. The next example involves three different forces acting on a system, it demonstrates the power of Euler's method.

Example 2.8: Nonlinear forces

Consider a particle with mass $m = 1$ kg, which is experiencing a Hooke's Law restoring force, $F(x) = -kx$, a quadratic drag, $F(v) = -cv^2$, and a sinusoidal external driving force, $F(t) = A\cos(\omega t)$, where $k = 3\,\text{N/m}$, $c = 0.5\,\text{Ns}^2/\text{m}$, $\omega = 1.0$ rad/s, and $A = 1.0$ N. The resulting equation of motion can be found using Newton's Second Law:

$$m\ddot{x} + kx + c\dot{x}^2 = A\cos(\omega t) \tag{2.7.10}$$

Next, we insert the given values for each parameter to obtain:

$$\ddot{x} + 3x + 0.5\dot{x}^2 = \cos(t) \tag{2.7.11}$$

If the particle's initial position and velocity is $x(0) = 0.9\,\text{m}$ and $v(0) = 0.0\,\text{m/s}$, respectively, find the particle's position and velocity as a function of time.

Solution: