PHY371 Midterm Project
Modeling a Rocket Launch with Gravity and Air Resistance

Instructor: Ying Tang
Department of Physics
Gordon College

October 14, 2014

Instruction

1. In this project, you will write a python program to simulate a rocket being launched from the earth to the International Space Station (ISS). We will follow a chapter — ”Rockets: Complex physics made simple” — written by Larry Engelhardt (Francis Marion University). You should finish all tasks in this chapter step by step. Ignore all paragraphs related to Ejs (easy java simulation).
2. Write a Vpython program to create a 3D animation of your simulation.
3. After finishing the required assignments in ”Rockets: Complex physics made simple”, you are encouraged to do some creative work. Flying to the moon is a good idea (note that the moon is rotating around the earth). Feel free to refuel your rocket at the International Space Station (ISS) before you head out for another adventure. Designing a rocket that has throwaway tank(s) is nice too. You can also take the challenge of how to do a smooth landing on the ISS (harder than it sounds). Let your imagination take flight in the outer space. However, only do it after you have finished basic requirements of the project. If you have any doubt about whether your new idea fits into the project or not, please talk to me before you proceed. I will give 3 (out of 20) points for being creative (in good ways).
4. You and your teammate should contribute equally to this project. In your scripts, you should indicate author’s name for each part. Both people are expected to talk during the presentation.
5. You have the midterm presentation in class on Tuesday 10/28 and Thursday 10/30.

Groups that present on Tuesday 10/28 are:
1) Max Halik and Tim Jordan, 2) Joy Kimmel and Rachel You, and 3) Sumin Kyoung and Ji in So.

On Thursday 10/30:
1) Rebecca Li and Jonathan Zhang, 2) Daniel Martins and Chris Richardson, and 3) Matthew Georges and Nick Hammes.

6. Each group has 30 mins to present your work. Please prepare for a 20-25 minute talk. We will have some time for questions and the evaluation. In your presentation, you should include presentation slides, animations, and your collaborative scripts. You are welcome to include other material in your presentation as well.

7. Your grade will be evaluated based on the correctness of results, good presentation skills and nicely written programs. Peer evaluation counts 10 points and my evaluation 10 points as well. The total points are 20.

Submission

Please submit your python programs (.py file) and your presentation slides (in the format of Keynote, PowerPoint or PDF) to ying.tang at gordon.edu before **11:59pm Monday 10/27**. Your python program should allow me to reproduce data and plots in your presentation. A late submission will cause a 50% deduction of your grade.

Chapter 6

Rockets: Complex physics made simple (sort of)

In this chapter we examine the dynamics of rockets. This is a complex topic, but using a computer—and solving the problem numerically—we will actually be able to get results without an unreasonable investment of time and effort. (Translation: This will take some work, but you can do it!)

Since the dynamics of rockets are complex, we will approach our modeling in stages.[1] First we will introduce the theory and implementation of *thrust*, and using (only) the force of thrust, you will compare your numerical results with well-known analytical results. Next, we will include *gravity*, which is (obviously) important when on or near the earth. Assuming a constant gravitational force (which is appropriate as long as the rocket doesn't go up too high), you will again compare your numerical results with analytical results. Next, we will incorporate Newton's *universal* form of gravity, which is altitude-dependent. Finally, we will include a simplified—but still sophisticated—model of *air resistance*.[2] This force becomes important at high speeds, and it also depends on altitude in a non-trivial way. Before getting to any of these details though, here is your assignment...

[1]Slight pun intended.

[2]For a more sophisticated—yet relatively accessible—discussion of the physics of flight, see N. Harris McClamroch, "Steady Aircraft Flight and Performance," Princeton University Press (2011).

6.1 Assignment

The goal of this chapter will be for you to develop and implement a computer model to simulate a rocket being launched to the International Space Station (ISS), which is orbiting at an altitude of 355 km. The values of the parameters for one specific rocket are provided here, but your program should be constructed such that a user will easily be able to enter different values of these parameters to simulate different rockets.

The rocket that is going to be used for the launch that you are assigned to simulate has the following properties:[3]

- Empty mass (with no fuel and no payload) is 10,000 kg
- Maximum fuel capacity is 10,000 kg of rocket fuel
- Exhaust velocity is 7000 m/s
- Burn rate is 100 kg/sec
- Reference area (of the front of the rocket) is 10 m^2
- Drag coefficient is 0.5

Before the launch takes place, you have been asked to answer the following questions:

1. Using the rocket's full fuel capacity, what is the maximum payload that could be delivered to the ISS?
2. For this same payload, what would happen if we were to only use half of the fuel capacity? (Could we make it to the ISS? If not, how high could we get? Could we make it to the ISS using half the fuel capacity, but carrying a smaller payload?)
3. Create plots of height versus time, velocity versus time, and force versus time (with a separate curve representing each individual force).

To complete this assignment, you will need to add variables to the Model. You will need to enter the correct equations in the Evolution. You will need to create a method to calculate the forces acting on the rocket. You will need to add an event to print the maximum height that is achieved, and add an event that will pause the

[3]The significance of each of these properties is discussed in the following sections.

simulation if the rocket hits the ground. You will need to add plots to the View. Finally, you will need to add text fields to the View in order to be able to vary input parameters.

6.2 Rocket template—Flame and fuel

Begin by opening the Ejs program `RocketTemplate.ejs`, running it, and observing what happens. This program contains a visual representation of a rocket, but you will notice that very little of the physics has been implemented. In particular, when you run the program you will notice that rocket fuel is burned—and the mass of the remaining rocket fuel decreases—but the fuel never runs out! Instead, the rocket (apparently) continues to burn fuel, even though the mass of the fuel becomes negative. Unfortunately, this is not realistic. If it were, our energy problems would be solved!

Before looking at how to *fix* the fuel problem, let's start by taking a look in the View to see how the rocket visualization has been created. You will find an element in the View named "rocketGroup" which contains the visual representation of the rocket. Using a "group" element in the Ejs View provides a handy way to construct a picture—consisting of simpler View elements—that can be translated and/or rotated as a single entity.[4] In this way, all of the rocket's pieces are guaranteed to ascend together.[5] Within the rocketGroup element, double click on "flame" to view the properties of the image of the flame. The following (*correct*) line of code has already been entered to control the visibility of the flame.

```
(t>0) && (burnRate>0)
```

Listing 6.1: Rocket flame visibility.

Recall that in Java "&&" is the "*logical and*" operator. This means that *if* `(t>0)` is true *and* `(burnRate>0)` is true, *then* the flame will be visible. Otherwise, the flame will not be visible. The first part, `(t>0)`, guarantees that the flame will not be visible before the launch begins (at `t=0`). The second part, `(burnRate>0)`, specifies that the flame will be shown if (and *only* if) rocket fuel is being burned, so if `burnRate=0`, the flame will not be visible. This should give a clue as to what needs to happen in order to fix the problem of burning "negative" rocket fuel!

The reason that rocket fuel continues to be burned even after it has all run out

[4]Another nice example of combining View elements into a "group" can be found at www.compadre.org/osp/items/detail.cfm?ID=8243.

[5]If you wish to modify the drawing of the rocket, or replace it with an existing image, feel free. This won't affect the *physics* in any way.

has to do with the "burn rate". An important property of any rocket is the rate at which fuel is ejected, also called the *burn rate*, r. Mathematically, the burn rate is

$$r = \frac{dm}{dt}, \text{ or if } r \text{ is constant,} \tag{6.1a}$$

$$r = \frac{\Delta m}{\Delta t}. \tag{6.1b}$$

Here Δm represents the mass of the fuel ejected from the rocket during the time interval Δt. For example, if a rocket has a burn rate of $r = 100$ kg/sec, and it starts out with 10 kg of fuel, that fuel will last for 0.1 sec. Look in Ejs under `Model`, `Evolution`, and you will see where Eq. (6.1) is implemented. (The negative sign in the evolution specifies that the mass of the *remaining* fuel should *decrease* with time.)

In order to fix the fuel problem, what *should* happen when the mass of the rocket fuel is equal to zero? How can this be accomplished? Recall that Ejs has a built in mechanism to handle situations like this, called an "Event".[6] Go ahead and add an event to the simulation to fix the fuel problem. Then you will be ready to get started on the real physics, which begins in the following section.

6.3 Rocket thrust

6.3.1 Theory

As you know, exhaust is shot out of the bottom of a rocket, and—assuming the velocity of the exhaust, v_{ex}, is fast enough—the rocket goes up. (Simple enough!) This phenomenon is referred to as "thrust," and the material ejected from the bottom of the rocket is rocket fuel. In this section, we will derive the basic equation needed to implement the force of thrust in your simulation. (For additional information regarding rocket thrust, see `http://en.wikipedia.org/wiki/Rocket#Physics` and the links contained therein.)

The details of how and why a rocket moves can be described by Newton's laws of motion. Specifically, Newton's 2^{nd} law can be expressed as

$$F = \frac{dp}{dt}, \tag{6.2}$$

where F represents the net force on an object, p represents the object's momentum, and t represents time.

[6]See Sec. 5.2 for a reminder of how and why to use Events.

In order to simulate rockets, we will be solving equations numerically, by taking many small—but finite—steps. For this reason, it will be convenient to also introduce the finite difference form of Eq. (6.2),

$$F \approx \frac{\Delta p}{\Delta t}, \tag{6.3}$$

where the approximation will be accurate provided that the time interval, Δt, is sufficiently small. (Here Δp represents the change in momentum, or "impulse" that takes place during the time interval Δt.) Now one can easily rearrange Eq. (6.3) to obtain the impulse equation,[7]

$$\Delta p \approx F \Delta t. \tag{6.4}$$

Now that we have a few equations to work with, let's stop to think about how these apply to rockets. When considering a rocket launch, there are *two* things that are moving: The rocket is moving *up*, and a bit of the rocket fuel is moving *down*. Hence, Eq. (6.4) applies to both the rocket—which receives an impulse Δp_r in the time interval Δt—and to the bit of ejected rocket fuel—which receives an impulse Δp_f in the time interval Δt. Finally, Newton's 3^{rd} law tells us that the force exerted on the rocket by the fuel is equal in magnitude to the force exerted on the fuel by the rocket. Hence, $\Delta p_r = \Delta p_f$.

The previous paragraph might be *conceptually* challenging, but the result is simple: Each time step, the rocket's momentum increases by Δp_r, and this is equal to the impulse given to the bit of rocket fuel that was ejected during that time step. So can we calculate Δp_r? The answer is "yes!"... by calculating Δp_f, as described below.

Recall from Sec. 6.2 that a rocket's burn rate is represented by the equation $r = \Delta m/\Delta t$. Rearranging this equation, the mass of the "bit of fuel" ejected during one time step is $\Delta m = r\Delta t$. That bit of fuel is ejected at a known exhaust velocity, v_{ex}, so now we just need to recall that (for speeds that are small compared to the speed of light[8]) the product of mass times velocity *is* momentum. Hence, the impulse given to the bit of rocket fuel is

$$\Delta p_f = \Delta m \times v_{ex} \tag{6.5a}$$

$$\Delta p_f = (r\Delta t) \times v_{ex}, \tag{6.5b}$$

and since $\Delta p_f = \Delta p_r$, the impulse given to the rocket in one time step is

$$\Delta p_r = r v_{ex} \Delta t. \tag{6.6}$$

[7]The $\approx$ symbol in Eq. (6.4) can be replaced with $=$ in the case that F does not vary with time.

[8]For speeds close to the speed of light, the "relativistic" equation for momentum must be used which is $p = \gamma m v$. For "normal" speeds ($v < 10^6$ m/s), $\gamma \approx 1$, so this reduces to $p = mv$.

Finally, by combining Eq. (6.5) and (6.3), we can write down the *force of thrust*,

$$F_t = \frac{r v_{ex} \Delta t}{\Delta t}, \text{ or} \tag{6.7a}$$

$$F_t = r v_{ex}. \tag{6.7b}$$

Ultimately, after a page or two of mathematical gymnastics we have arrived at Eq. (6.7). The force of thrust is simply the product of *how much* rocket fuel is ejected per second and *how fast* the rocket fuel comes flying out! Before proceeding on, be sure to check that the right hand side of Eq. (6.7) does indeed have the correct units. If it doesn't, demand your money back for this text.

6.3.2 Implementation

Now it is time to "let there be thrust"! As we have seen in the previous section, the force of thrust exerted on a rocket is a product of two parameters: The burn rate, r, and the exhaust velocity, v_{ex}. We should appreciate what a simple result this is! To implement the force of thrust in an Ejs program, we simply need two input parameters. Where (in Ejs) will changes need to be made? (Hint: If you aren't sure what will be needed, look back at what you did to simulate the falling ball.)

Once you have the force of thrust implemented, run the program for various values of the parameters, and make sure that the results seem reasonable. You won't know yet whether or not the results are *correct*, but you should be able to assess whether or not they are *reasonable.* In order to do so, you will need some *output.* This would be a good time to create plots of position, velocity, and force as a function of time.

Note: At this point, even if you have done everything correctly, your results will only be valid in outer space. For the simulation to be useful on Earth, you will need to also include other effects, as discussed in the following sections.

6.3.3 Analytic result

At this point, you should have already implemented the force of thrust, and verified that your simulation yields reasonable results. Now we will check whether or not these results are *correct.* The "Tsiolkovsky rocket equation"[9] provides a simple

[9]See `http://en.wikipedia.org/wiki/Tsiolkovsky_rocket_equation` for a discussion of this equation.

analytical formula[10] for a rocket's change in velocity. This equation can be written

$$\Delta v_{an} = v_{ex} \ln\left(\frac{m_0}{m}\right), \tag{6.8}$$

where v_{an} represents the analytically calculated velocity, m_0 is the initial ($t = 0$) mass of the rocket, and m is the current mass of the rocket (after having ejected some of the rocket fuel). For a rocket that starts at rest ($v = 0$ at $t = 0$), this becomes

$$v_{an} = v_{ex} \ln\left(\frac{m_0}{m}\right). \tag{6.9}$$

To test your numerical results, include Eq. (6.9) in your program. Which Ejs window should you use to type the equation? Is there anything that needs to be added to your program before you include the equation?

After implementing Eq. (6.9), add a new curve (trail) to your plot of velocity versus time to show the analytic results. Do the numerical and analytic results appear to match? (If they are not at least *close*, something is wrong!) Also calculate the error (the difference between the numerical and analytic velocities), and plot this as a function of time. Does the error increase, decrease, or stay constant as a function of time? Does this make sense? Try different ODE solvers to see which ones yield the best results, and vary the value `dt` to test for convergence. (Best sure to include some *data* to support your conclusions regarding the ODE solvers and convergence.)

Note that—using Eq. (6.9) as a starting point—the height of the rocket can also be calculated analytically. This would require you to analytically evaluate the integral

$$y_{an}(t) = \int_0^t v_{an}\, d\tau, \tag{6.10}$$

where the variable of integration, τ, represents the time at each instant in the range $0 < \tau < t$. To carry out this integration, you would first need to write m in terms of time. Then you will probably need to consult an integral table or make use of a symbolic computer algebra system. (See Sec. 9.7 for an introduction to the SymPy library for symbolic mathematics.)

[10] Recall, this equation is "analytic" because it does not require "stepping" from a previous value.

6.4 Gravity

6.4.1 Theory

We will introduce—and you will be asked to implement—both of the familiar[11] models of gravity. First, as long as we keep the rocket relatively close to the surface of the earth, the force of gravity exerted on the rocket is

$$F_g = mg_s, \tag{6.11}$$

where m is the mass of the rocket and g_s is the gravitational force per unit mass at the surface of the Earth, $g_s \approx 9.81$ Newtons/meter.[12] The implementation of Eq. (6.11) is discussed in the next section.

As you know, when a rocket travels to high altitudes, or even leaves Earth entirely, the force of gravity becomes weaker. When this happens, Eq. (6.11) won't quite work. Specifically, the value of the gravitational force per unit mass gets smaller, so in Eq. (6.11) we will replace g_s with g, where $g = g_s$ near the surface of the Earth, but $g < g_s$ at high altitudes. The more general form of Eq. (6.11) then becomes

$$F_g = mg. \tag{6.12}$$

So how do we calculate the value of g for a rocket in flight? The answer is that we must use "Newton's universal law of gravitation,"

$$F_g = \frac{GMm}{d^2}, \tag{6.13}$$

where G is "the gravitational constant," M and m are the masses of the two attracting objects (i.e., the Earth and the rocket), and d is the distance between the centers of the two attracting objects. Comparing Eq. (6.12) and (6.13), we can easily identify that

$$g = \frac{GM}{d^2}, \tag{6.14}$$

and we note that—for the special case of being on the surface of the Earth—$d = R_e$, so Eq. (6.14) becomes

$$g_s = \frac{GM}{R_e^2}, \tag{6.15}$$

[11]There are also less familiar models, such as general relativity, but as long as we keep the rocket away from any black holes, the current models will suffice.

[12]The value of g_s does vary somewhat with location on Earth. By using a different value of g_s, Eq. (6.11) can also be applied near the surface of other large bodies (e.g., the moon or Mars). See `http://en.wikipedia.org/wiki/Gravity_of_Earth` for additional details.

where R_e is the radius of the Earth ($R_e \approx 6.37 \times 10^6$ m).[13]

We could be finished at this point, using Eq. (6.12) and (6.14) to calculate the force of gravity. However, the constants G and M are not very convenient ($G \approx 6.674 \times 10^{-11}$ N m^2/kg^2 and $M \approx 5.9722 \times 10^{24}$ kg), so we will do a bit more math to rewrite Eq. (6.14) to avoid using these constants.

Note that by rearranging Eq. (6.15), one finds $GM = g_s R_e^2$, which can be inserted into Eq. (6.15) to give

$$g = \frac{g_s R_e^2}{d^2}. \tag{6.16}$$

Finally, when simulating our rocket, we are interested in the *height* (or *altitude*) of the rocket, y—which is measured from the *surface* of the Earth, not the center. Hence we will make the replacement $d = R_e + y$ in Eq. (6.16). After a few additional lines of algebra,[14] this gives the convenient result,

$$g = \frac{g_s}{\left(1 + \frac{y}{R_e}\right)^2}. \tag{6.17}$$

Now we are done with gravity! Before proceeding to the implementation of Eq. (6.12) and (6.17), do check that Eq. (6.17) seems reasonable. Are the units correct? What value of g does it yield for $y = 0$? How about for $y = R_e$? How about for $y = 6$ meters? (This would correspond one one-millionth of the Earth's radius, which is about the top of a two-story house.)

6.4.2 Implementation

To implement gravity, you will need to calculate another force, just as you already calculated the force of thrust. The force of gravity is fairly easy to calculate, but be aware that—even when using Eq. (6.11)—this force is not constant. The rocket is losing mass every time step, so be sure to use the *current* mass of the rocket. Add a curve for F_g to your force versus time plot. How does F_g depend on time? Does this make sense? How does the value of F_g compare to F_t? After implementing gravity using Eq. (6.11), compare your results (for velocity versus time) with the analytic equation given in the following section before proceeding.

To accurately model the motion of a rocket at high altitudes, Eq. (6.12) and (6.17) should be used. As always, make sure that your results seem reasonable! How

[13]The radius of the Earth varies by about 0.5% between different locations. See `http://en.wikipedia.org/wiki/Earth_radius` for details.

[14]It is left to the reader to verify that Eq. (6.16) does indeed lead to Eq. (6.17)

does the gravitational force depend on time? Can you notice any differences between the current results and the results that you obtained using Eq. (6.11)? Explain.

6.4.3 Analytic result

Assuming a constant gravitational force per units mass, the force of gravity adds a very simple correction to Eq. (6.9). Namely, the gravitational force causes an additional (*constant*) acceleration equal to $-g_s$, so the analytic velocity is now given by

$$v_{an} = v_{ex} \ln\left(\frac{m_0}{m}\right) - g_s t. \tag{6.18}$$

Make this correction to your analytical calculation of velocity, and verify that the analytical results agree with the numerical results, assuming $F_g = mg_s$. Do the numerical results converge to the analytical results as you decrease the time step? If not, there is a problem... be sure to fix it!

Using Eq. (6.18), the height can also be calculated analytically, as described at the end of Sec. 6.3.3. However, we will *not* be able analytically calculate the velocity and height for the more general model of gravity (where g decreases with altitude according to Eq. (6.17)). Here's why: An analytical calculation of velocity involves the integral, $\Delta v = \int a(t)\,dt$. However, the acceleration, $a = F/m$ now depends on the height, y, in a complicated way; and in order to calculate the height, we need to know the velocity: $\Delta y = \int v(t)\,dt$. Things have suddenly become very complicated! Moreover, you will soon be implementing a sophisticated model of air resistance, causing the acceleration to become even more complicated.

There is an important lesson to be learned here regarding analytical results! Relatively simple analytical results serve an important role: They allow us to verify that a numerical solution is correct (that we haven't screwed something up!) and to test the numerical accuracy of the approximations. However, at a certain point we simply "run out" of analytical results. At this point, the hope is that our numerical solution has already been tested with sufficient rigor, such that we are relatively confident with our next results—even though we will not be able to compare them with an analytical solution.

6.5 Air resistance

6.5.1 Theory

At this point, you have already developed a fairly sophisticated model of a rocket being launched! However, when an object goes fast, air resistance is important—so we will now introduce air resistance. Note, however, that air resistance is complicated! It will be necessary to make several simplifying assumptions, but in the end, we will have a model that is simple enough to be implemented, yet "rich" enough to yield interesting results.

As we discussed in the previous chapter, there is not a single "correct" model of air resistance. However, the drag equation,[15]

$$\vec{F_d} = -\frac{1}{2}\rho A C_d v^2 \hat{\mathrm{v}}, \tag{6.19}$$

should provide an accurate model to describe the force of air resistance that is exerted on a fast-moving rocket. The $-\hat{\mathrm{v}}$ in Eq. (6.19) indicates that the direction of this force opposes the motion of the rocket, and the remaining $\frac{1}{2}\rho A C_d v^2$ is the magnitude of this force. Here A is the reference area and C_d is the drag coefficient, which are both (*constant*) properties of a rocket's shape. v is the speed of the rocket, and ρ is the density of the fluid through which the rocket is moving—i.e., air.

The one complicated piece of Eq. (6.19) is density: As altitude increases, air becomes less dense. This phenomenon is very relevant to mountain climbers, and is also the reason that very few commercial aircraft have retractable roofs![16] So how do we calculate ρ for our rocket in flight? On the microscopic scale, the density depends on the detailed interactions between the molecules in the air; but, these interactions are prohibitively complex. An alternative approach that will be simple enough for us to implement—yet rich enough to provide interesting results—will be to model the air as an *ideal gas*.[17] The ideal gas law states

$$PV = nRT, \text{ or} \tag{6.20a}$$

$$\frac{n}{V} = \frac{P}{RT}, \tag{6.20b}$$

[15]See `http://en.wikipedia.org/wiki/Drag_(physics)` a discussion of various types of drag, including Eq. (6.19).

[16]I'm sure this is not the only reason...

[17]The ideal gas law is based on the assumption that the individual atoms or molecules in the gas do not interact with one another, which should provide reasonably accurate results provided that the density of the gas is not too high. (If molecules are "squished together" too much, intermolecular interactions cannot be ignored.)

where P is the pressure of the gas, V is the volume of a certain amount of the gas, n is the number of moles contained in that volume, R is the "ideal gas constant" ($R = 8.314 \frac{\text{Joules}}{\text{Kelvin}\times\text{mole}}$), and T is temperature. From Eq. (6.20b), we can obtain density by multiplying both sides by the molar mass or air,[18] $M = 0.02896$ kg/mole. The quantity $\frac{Mn}{V}$ *is* density, so

$$\rho = \frac{PM}{RT}. \tag{6.21}$$

Note that Eq. (6.21) now contains *two* variables that will vary with altitude: temperature and pressure. This might seem like we have gone "backwards" from Eq. (6.19), but the advantage is that T and P can both be evaluated directly as a function of altitude, y. To do this, we will assume that the temperature decreases linearly[19] with increasing altitude, i.e.,

$$T = T_0 - Ly, \tag{6.22}$$

where T_0 is the temperature at sea level (which has an average value of $T_0 = 288$ K) and L is the "temperature lapse rate" ($L = 0.0065$ Kelvin per meter). Note that, according to Eq. (6.22), $T = 0$ will be reached at a certain (easily calculated) altitude. This is not physically accurate; in practice, "absolute zero" cannot ever be achieved, but it does provide a convenient way for us to identify the top of the Earth's atmosphere for our model. We will simply assume that the rocket has left the Earth's atmosphere if Eq. (6.22) gives a value of $T \leq 0$.

Assuming that temperature varies linearly with altitude, the pressure at any altitude is then given in terms of T by[20]

$$P = P_0 \left(\frac{T}{T_0}\right)^{\frac{gM}{RL}}, \tag{6.23}$$

where P_0 is the pressure at sea level (which has an average value of $P_0 = 1.01 \times 10^5$ N/m^2).

[18]Assuming that the chemical composition of air is constant, the molar mass is also constant. However, this molar mass does vary somewhat with humidity.

[19]The assumption of linear temperature decrease is actually only accurate inside the troposphere (the lowest layer of the Earth's atmosphere), as described at `http://en.wikipedia.org/wiki/Density_of_air#Altitude`. However, in order to avoid a more complex model, we will apply this model into the stratosphere (the next layer of the atmosphere) as well. If you wish to develop a more sophisticated model, you can begin to find out more about the stratosphere at `http://en.wikipedia.org/wiki/Stratosphere`.

[20]We will not derive Eq. (6.23), but we will briefly outline its origin. The pressure *gradient*, $\frac{dP}{dy}$, can be integrated to find the change in pressure: $\Delta P = \int \frac{dP}{dy}\, dy$. The infinitesimal change in pressure, dP, will depend on the weight of the air in a column immediately above the current altitude, which subsequently depends on pressure and temperature, leading to Eq. (6.23). (See `http://en.wikipedia.org/wiki/Troposphere#Pressure`.)

With Eq. (6.23), we now have all of the pieces that are necessary to implement air resistance! Clearly, air resistance is significantly more complicated than thrust or gravity; and for a truly accurate model of a *specific* launch, the current, local atmospheric conditions should be used for input parameters. However, we now have a model that is able to provide a rich—yet tractable—simulation of all three of the major forces that act on a rocket.

6.5.2 Implementation

From the discussion in the previous section, air resistance is more complex that thrust and gravity. . . but it's not actually *hard*. Using Eq. (6.19), the magnitude of the drag force depends on two constants (which are determined by the shape of the rocket) and two dynamical variables—both of which we can calculate.

At this point, go ahead and implement the drag force, being careful of its *direction* as well as its magnitude. As always, make sure that the results *make sense*. To complete this assignment, refer back to the information and questions provided in Sec. 6.1.

6.6 Reflection

Before leaving from the rocket, it will be useful to briefly reflect on a couple of important lessons/strategies that it has taught us. First, our simulation—and the implementation of air resistance in particular—provides a striking example of what a *model* actually is. Einstein famously said, "*Everything should be made as simple as possible, but not simpler*," and that is exactly what we have done. The detailed interactions of the molecules in the air—both with each other and with the rocket—are responsible for air resistance, but these interactions are too complex for us to simulate. We could ignore air resistance all together, but that would be *too* simple. Instead, we have constructed a model of air resistance that behaves correctly (increasing with speed, but decreasing with altitude), yet is simple enough to be implemented. This is what the process of "modeling" is all about, and it is extremely important to modern scientific inquiry.

Our rocket simulation has also provided an example of the usefulness—and limitations—of *analytical* results. Our main focus throughout this text is on solving problems *numerically* (not analytically): We take many small steps,[21] where in each step we obtain the next value by "iterating" on the previous value. The

[21] More accurately, we instruct the *computer* to take many small steps.

advantage of this numerical approach is that it can be applied to *arbitrarily complex* problems (even the altitude depended air resistance introduced in Sec. 6.5.1). However, numerical results involve certain unavoidable approximations, so it is important to always have a sense of what level of accuracy numerical results do—and do not—provide. This is where *analytical* results are useful. Analytical results can be evaluated directly (by simply plugging numbers into the right hand side of an equation), so they do not require approximations. A direct comparison between analytical and numerical results thus allows us to "test out" a numerical approach. The disadvantage of analytical approaches is that they can only be applied to relatively simple situations. Hence, the strategy that is commonly used is to start with a simple (analytically tractable) model, test a numerical implementation, then apply the numerical implementation to a somewhat more "beefed up" model.