MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 – Structure and Interpretation of Computer Programs
Spring Semester, 2005

Project 1 – Those amazing Red Sox!

1. Issued: Monday, February 7
2. To Be Completed By: Friday, February 18, 6:00 PM
3. Code to load for this project:
    1. A link to the code file `basebot.scm` is provided from the Projects link on the projects section. This file contains the skeleton of the procedures described here, and some utility procedures you will need.

Purpose

The purpose of Project 1 is for you to gain experience with writing and testing relatively simple procedures. You should create a file with your project solutions for uploading to the 6.001 on-line tutor. For each problem below, include your code (with identification of the problem number being solved), as well as comments and explanations of your code, **and** demonstrate your code's functionality against a set of test cases. On occasion, we may provide some example test cases, but you should always create and include your own additional, meaningful test cases to ensure that your code works not only on typical inputs, but also on "boundary" or difficult cases. Get in the habit of writing and running these test cases after **every** procedure you write – no matter how trivial the procedure may seem to you.

Additional guidelines for project submission are available under the "How to write up a project" link on the projects section.

Scenario

As you may have noticed this past fall, a remarkable event took place – the Boston Red Sox won the World Series for the first time in 86 years! You may also have noticed long time Boston residents (such as MIT professors) walking about in a state of bliss. Because many of these folks don't want to have to wait another 86 years for this to happen again, "Red Sox Nation" has hired us to provide some help. In particular, we are to investigate the possibility of perfecting a baseball robot ("basebot") that can accurately throw and can hit with power.

Problem 1: Some simple physics

We are going to begin by modeling how far a baseball can travel – the same physics will hold for both hitting a ball and throwing a ball. We are going to simplify things by

assuming that baseballs don't spin as they move (clearly false but it makes life much easier). This means we can treat the movement of a baseball as if it were restricted to a two-dimensional plane. So what happens when a baseball is hit? For the moment, we'll model a baseball as a particle that moves along a single dimension with some initial position *u*, some initial velocity *v*, and some initial acceleration *a*, as pictured in Figure 1 below. The equation for the position of the baseball at time *t*, given *a*, *v*, and *u* is $u_t = \frac{1}{2} a t^2 + v t + u$. Note that this denotes a first order differential equation in time. Later, we can apply this equation to either the horizontal (*x*) component of baseball motion, or the vertical (*y*) component of baseball motion.

Figure 1: Motion of a b in a generic direction.

Write a procedure that takes as input values for `a`, `v`, `u`, and `t` and returns as output the position of the baseball at time `t`.

```
(define position
  (lambda (a v u t)
    YOUR-CODE-HERE))
```

Test your position code for at least the following cases:

```
(position 0 0 0 0)      ; -> 0
(position 0 0 20 0)     ; -> 20
(position 0 5 10 10)    ; -> 60
(position 2 2 2 2)      ; ->
(position 5 5 5 5)      ; ->
```

The template code file `basebot.scm` will have these tests, and other test cases for other procedures, which you should run (you can add/show your output values). In addition, you should add some test cases of your own to these to cover other boundary and typical conditions.

Problem 2: Basic Math

One of our goals is to determine how far a baseball will travel in the air, if it is hit with some initial velocity at some initial angle with respect to the ground. To do this, we will need to know when the baseball hits the ground, and for that we'll want to find when the *y* coordinate of the baseball's position reaches zero. This can be discovered by finding the roots of the *y* position equation, and selecting the one that is larger (later in time). The proper tool for this is the quadratic formula. Given the coefficients of the quadratic

equation $az^2 + bz + c = 0$, write a procedure to find one of the roots (call this `root1`), and another procedure to find the other root (call this `root2`).

```
(define root1
  (lambda (a b c)
    YOUR-CODE-HERE))

(define root2
  (lambda (a b c)
    YOUR-CODE-HERE))
```

You may notice that, depending on how you wrote your procedures, for some test cases you get an error. For example, try (root1 5 3 6). What happens? If you get an error, which is likely if you wrote your code the straightforward way, figure out how to change it so that your procedure returns a false value in those cases where there is not a valid solution.

Problem 3: Flight Time

Given an initial upward velocity (in meters per second, or m/s) and initial elevation or height (in meters, or m), write a procedure that computes how long the baseball will be in flight. Remember that gravity is a downward acceleration of 9.8m/s^2. Note that to solve this you will need a root of a quadratic equation. Try using `root1`, and using `root2`. Only one of these solutions makes sense. Which one? And why? Use this to create a correct version of the procedure below.

```
(define time-to-impact
  (lambda (vertical-velocity elevation)
    YOUR-CODE-HERE))
```

In some cases, we may want to know how long it takes for the ball to drop to a particular height, other than 0. Using your previous procedure as a template, write a procedure that computes the time for the ball to reach a given target elevation.

```
(define time-to-height
  (lambda (vertical-velocity elevation target-elevation)
    YOUR-CODE-HERE))
```

Problem 4: Flight Distance

Suppose the baseball is hit with some velocity *v*, at a starting angle *alpha* relative to the horizontal (in degrees), and from an initial elevation (in meters). We wish to compute the distance in the horizontal direction the baseball will travel by the time it lands. Remember that some of the velocity vector goes into the *x* direction, and some into the *y*, as pictured in Figure 2 below.

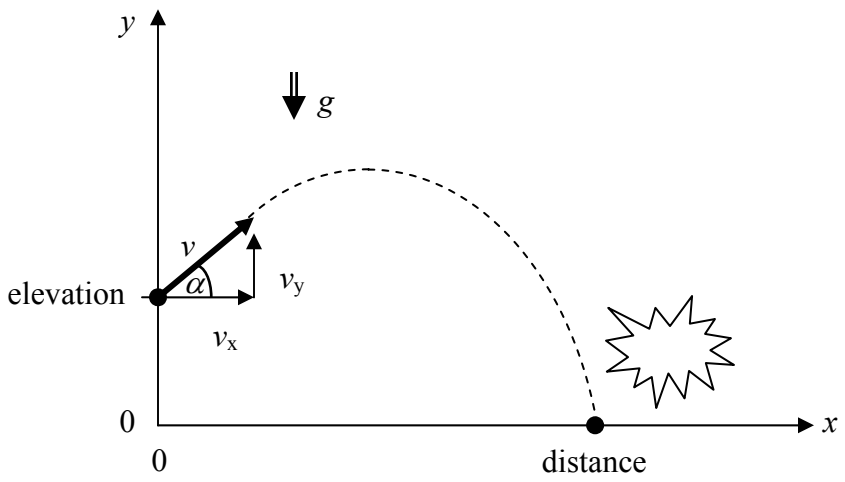

Figure 2: Motion of a baseball in two dimensions, acting under gravitational acceleration *g*.

Checking the Scheme manual, you will find procedures `sin` and `cos`. To use these (which require angles in radians rather than degrees), you may also find the procedure `degree2radian` useful. It is given below:

```
(define degree2radian
  (lambda (deg)
    (/ (* deg pi) 180.)))
```

Write a procedure `travel-distance-simple` that returns the lateral distance the baseball thrown with given velocity, angle, and initial elevation will travel before hitting the ground.

```
(define travel-distance-simple
  (lambda (elevation velocity angle)
    YOUR-CODE-HERE))
```

Try this out for some values. Note that we are doing everything in metric units (distances in meters, weight in kilograms). You may be more accustomed to thinking about baseball in English units (e.g. feet). So we have created some simple procedures to convert feet to meters and *vice versa* (see the code file for details).

Using your code, determine the time to impact of a ball hit at a height of 1 meter (right down the middle of the plate) with an initial velocity of 45 m/sec (about 100 mph – about what a really good professional player can do – without steroids), at angles of 0, 45 and 90 degrees (be sure to use radian units or degree units depending on how you provide input to *sin* and *cos*, but remember that those procedures expect arguments in units of radians).

How far does the baseball travel in each case? Provide answers both in meters and feet. Notice the distance traveled in feet for a ball hit at a 45 degree angle, with this bat speed. Seems incredible – right? We'll come back to this in a little bit.

Problem 5: What's the best angle to hit?

Before we figure out why professional players don't normally hit 700 foot home runs, let's first see if we can find out the optimal angle at which to launch a baseball, in order to have it travel the furthest. Write a procedure `find-best-angle` that takes as arguments an initial elevation and an initial velocity, and which finds the best angle at which to hit the baseball to optimize distance traveled. You will probably want to write a recursive procedure that tries different angles between 0 and pi/2 radians, sampled every 0.01 radians (say) or between 0 and 90 degrees, sampled every 1 degree (depending on whether your code works in radians or degrees – either way be sure that you provide the right kind of unit to your trigonometric functions).

```
(define find-best-angle
  (lambda (velocity elevation)
    YOUR-CODE-HERE))
```

Use this for same sample values of elevation and velocity. What conclusion can you reach about the optimal angle of hitting?

Problem 6: So why aren't baseball outfields 600 feet deep?

Let's go back to our distances. Why are these numbers for distances hit so unrealistic? -- because we haven't accounted for air friction or drag. (Of course there are some other effects, like spin, but we'll just stick with drag). Let's think about this. Newton's equation basically says that the movement of the ball will be governed by:

Drag + gravity = mass * acceleration

We can get the mass of a baseball (.15 kg). We know that force due to gravity – mass * 9.8 m/sec^2. The force due to drag is given by:

½ C rho A vel^2

where *C* is the drag coefficient (about 0.5 for baseball sized objects); *rho* is the density of air (about 1.25 kg/m^3 at sea level for moderate humidity – not a bad approximation for Boston, but about 1.06 for Denver); *A* is the cross-sectional area of the baseball (which is *pi D^2/4*, where *D* is the diameter of a baseball – about 0.074 m). Let's denote *½ C rho A* by the constant *beta*. Then we see that the drag on a baseball is basically proportional to the square of the velocity of the ball. So there is more drag when the ball is moving faster.

How can we compute the distance traveled by a baseball, but taking into account this drag effect? Basically we have four coupled linear differential equations:

Let's let x and y denote the two components of position of the baseball, and let's let u denote the velocity in the horizontal direction, and v denote the velocity in the vertical direction. We will let **V** denote the magnitude of the velocity. Then the equations of motion are:

$$dx/dt = u$$
$$dy/dt = v$$
$$du/dt = -1/m \; sqrt(u^2 + v^2) \; u \; beta$$
$$dv/dt = -1/m \; sqrt(u^2 + v^2) \; v \; beta - g$$

We can rewrite these as

$$dx = u \; dt$$
$$dy = v \; dt$$
$$du = - \; 1/m \; beta \; sqrt(u^2 + v^2) \; u \;\; dt$$
$$dv = - \; (1/m \; sqrt(u^2+v^2) \; v \; beta + g) \;\; dt$$

We also have some initial conditions on these parameters

$$x\_0 = 0$$
$$y\_0 = h$$
$$u\_0 = V \; cos \; alpha$$
$$v\_0 = V \; sin \; alpha$$

where *alpha* is the angle of the initial hit with respect to the ground, V is the initial velocity, and h is the initial height of the ball.

To find the distance traveled, we need to integrate these equations. That is, starting with the initial values, we want to move forward a small step in time (say 0.01 seconds), and compute the change in x and y, given the current estimates for velocity. This will gives us the new values of x and y. Similarly, we want to compute the change in u and v, and thus, the new values for u and v. We can keep recursively estimating these values until the value for y drops below 0, in which case the value for x tells us the distance traveled.

Based on this idea, write a procedure called `integrate,` which performs this computation. Using this, write a procedure called `travel-distance,` which given an initial elevation, an initial magnitude for the velocity, and an initial angle of launch, computes the distance traveled while accounting for drag.

Use this to determine how far a baseball will travel with an angle of 45 degrees, using initial velocities of 45 m/sec, 40 m/sec, 35 m/sec.

How quickly does the distance drop when the angle changes, i.e., how easily does a home run turn into a fly out? Run same examples and report on this. For instance, suppose that the outfield fence is 300 feet from home plate, and that the batter has very quick bat

speed, swing at about 100 mph (or 45 m/sec). For what range of angles will the ball land over the fence?

How much does this change if we were in Denver rather than Boston? Report on some examples.

Problem 7: Throwing instead of hitting

Now let's turn this around. Instead of worrying about how far the ball will carry when hit, suppose we want a robotic fielder that can throw the ball accurately and efficiently. For this, we want to determine the best angle to throw the ball at a given velocity in order to reach a target a given distance away in the shortest amount of time. We will assume the target is at height 0 (i.e. on the ground) – we could do this for a given height of the target but we'll assume that our fielders are good at catching things at ground level!

You need to write a procedure (or set of procedures) that use the same integration idea to accomplish the following. Given an input velocity and desired distance (plus the other parameters such as mass of the ball, the beta coefficient, gravity, and the height at which the throw was made), we want to try different initial angles (ranging from -90 to 90 degrees) at which to throw. If throwing at a particular angle will result in the ball traveling roughly the desired distance (up to some error) then we want to find the time it takes for the ball to reach the target using this trajectory (Hint: a variation on your integrate code will do this part). Finally, we want to find the trajectory that results in the shortest time, given a fixed initial velocity magnitude.

Write a set of procedures to do this. We suggest that if there is no angle in this range for which at the given velocity the player can throw the required distance, you return the answer 0 for the time so you can tell this case apart from the others.

You can use this to get a sense of times involved in baseball. For example, the distance from home plate to second base is roughly 36m. If your catcher has a gun for an arm, and can throw at 100 mph, (or 45 m/sec), how long does it take for the throw to reach second base? How long if he throws at 35 m/sec? of at 55 m/sec?

Note that a really good base runner should be able to get from first to second base in roughly 3 seconds. If the pitcher is throwing at 90 mph how long does it take to reach home? If the catcher throws at 90 mph, how much time does he have to catch and release the ball if he is going to put out a runner trying to steal second?

Now use your procedures to get some data on outfielders. Suppose an outfielder has a strong arm and can throw at 45m/sec. How quickly can he throw the ball to a target at a distance of 30m? 60m? 80m? What if he can throw 55 m/sec?

What about the same distances but with a weaker outfielder, who can only throw at 35m/sec?

Problem 8: Do it on a bounce

You should have noticed in the last problem that a weaker outfielder cannot in fact get the ball 90m in the air. So he may have to bounce it there. Let's model this effect.

Specifically, assume that when a ball bounces, it leaves the ground at the same angle as it was initially thrown (untrue but a reasonable approximation) but with half the velocity. Write a procedure that will determine the distance traveled, accounting for drag, given an initial velocity, an angle of throw, an initial height, and the number of bounces it will take. Remember that only on the initial stage is the ball released from a given height, for each subsequent bounce it is released from height 0. Remember as well that on each subsequent bounce, the velocity of the ball is dropping by one half. Use this to see how far a fielder can throw a ball on one bounce, on two bounces, on an arbitrary number of bounces until it stops moving. Do this for different initial velocities, and for different initial angles.

Problem 9: Do it on a bounce -- again

In Problem 8, we just assumed that the velocity would drop by one half on each bounce. But in fact if we are integrating trajectories in order to account for drag, we can actually compute the velocity of the ball when it bounces (since we know the x and y components of velocity when the ball hits the ground). Use this knowledge to refine your code from Problem 8, and rerun your test cases.

Submission

Once you have completed this project, your file should be **submitted electronically on the tutor**, using the `Submit Project Files` button. Remember that this is Project 1; when you are have completed all the work and saved it in a file, upload that file and submit it for Project 1.

Also be sure to identify any sources you have used in preparing this submission (other people you talked to, other sources you referenced). Be sure that you are in compliance with the state course policy on use of Bibles and on proper collaboration. If you did not use any other sources, state this in your submission.

Congratulations! You have reached the end of Project 1!