CS 439 Task 1: Dragster Performance

Description

This task addresses the simplistic performance of a dragster. The model has two parts: the engine and the transmission. The two engine models to evaluate are provided. The two transmission models are your responsibility.

The simulation executes a race against time on a straight track one mile (5,280 feet) long. The engine will increase from 1,000 to 9,000 revolutions per minute (RPM) in first gear, then do the same for the second through fifth gears. The combination of RPM and gear directly correspond to distance traveled and speed. There is no strategy or optimization involved in this task. You are coding a solution (the model), executing it (with the provided simulation), visualizing it (with Excel or something similar), and analyzing it (in a PDF report).

Lectures 9 and 10 work through the details. This document provides an overview. The spreadsheet includes all the results that your solution should produce (and more). The Java code provides everything except the transmission models for the dragster, which you need to write yourself.

Specifications

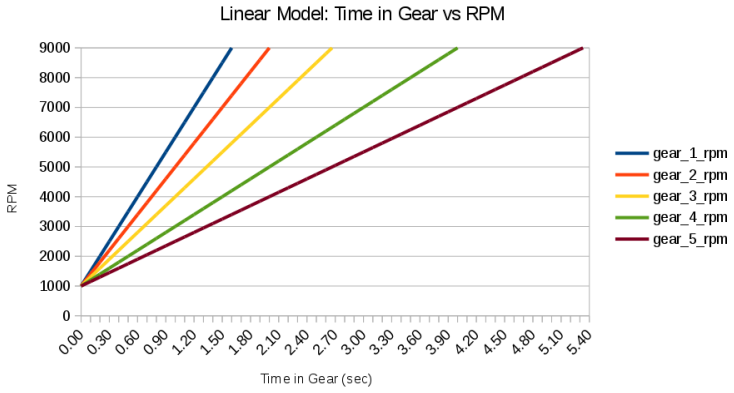
This solution is called a continuous time-stepped simulation, which we will investigate further in class. The idea is to maintain a simple clock that increments by a slight amount ("step") for each update. During this elapsed time since the previous update, the model recomputes whatever changed. For example, if the dragster moves at one mile per hour, and it is updated every minute, then it travels one-sixtieth of a mile per step. Time exists *only* at the updates. There is nothing in between these points. So in this case, the dragster continually disappears and reappears 88 feet farther down the track at each update (5,280 / 60). It never occupies the space in between. A smaller time step, say a second, reduces the hopping to 1.5 feet at each update (5,280 / 60 / 60). We do not use anything like a real clock or timer. Instead, the `execute()` method in `RaceSimulator` calls `update()` in the dragster in a loop as fast as Java runs until the dragster reaches the end of the track.

Parts 1 through 4 use a time step of 0.1 seconds; Part 5 uses 0.01 seconds.

Part 1: Automatic Shifting with Linear Engine Model

The linear engine model represents the increase in engine RPM in any gear as a straight line. The steeper the line is, the faster the increase.

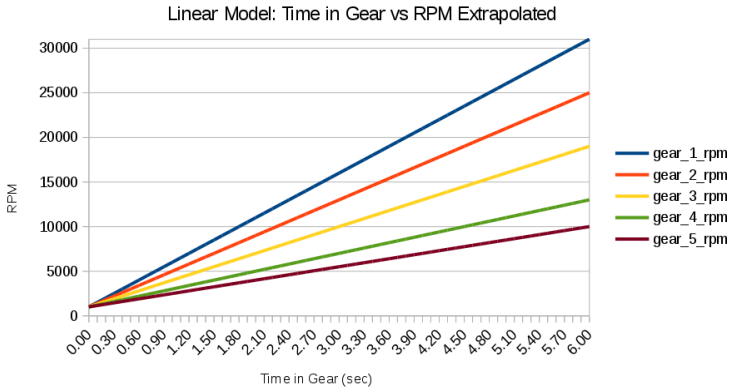
Graph 1 (in tab *linear time vs rpm*), based on `getRPM()` in `EngineModelLinear`, depicts RPM in any gear as a function of time. In other words, for any gear and time in this gear (starting from 0 seconds), it shows the corresponding RPM. The range stops at 9,000 because it is the threshold at which we automatically shift to the next gear.



Graph 1

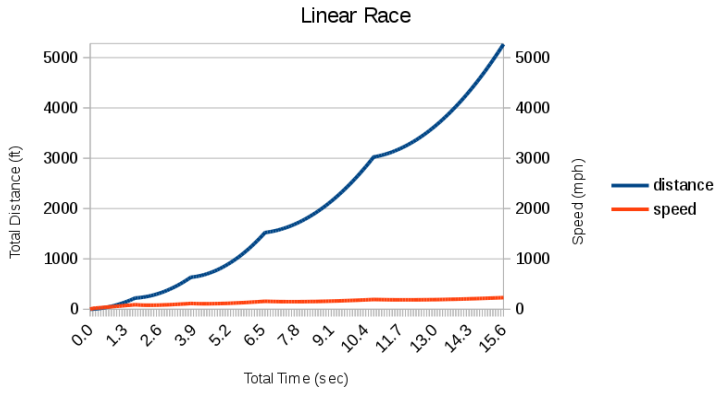
Graph 2 (in tab *linear time vs rpm extrap*) continues where Graph 1 stopped. It shows the RPM that is reached for any gear without consideration of whether this performance is physically possible or consistent with our model. For example, first gear reaches 31,000 RPM after 6 seconds, which would obliterate the engine. The graph is only for comparison with Graph 5 because we do not need the higher range.

Be careful in interpreting graphs generated automatically in Excel. The large apparent difference between the slopes in these two graphs is due only to the scaling; up to 9,000 RPM, they are actually identical.



Graph 2

The race over a mile shifts through all five gears as fast as possible. Tabs *linear race* and *linear race distance vs gear* show in detail how far and fast the dragster moves at each update and when the shifts occur. The shifts in Graph 3 appear as the little peaks on the blue line.



Graph 3

Speed on the orange line is difficult to interpret because the scale is dominated by the distance. Tab *linear race speed vs gear* breaks it out in more detail.

Implement your transmission model in the `update()` method in `DragsterAutomatic`. Here are some hints.

The distance traveled at each update is based on the elapsed time since the previous update, which is given by the simulation in argument `timeStep`:

```
distanceDelta = (rpm / SECONDS_PER_MINUTE) * timeStep * gearRatio * FEET_PER_REVOLUTION
```

The total distance at any time is the sum of these intermediate ("delta") distances.

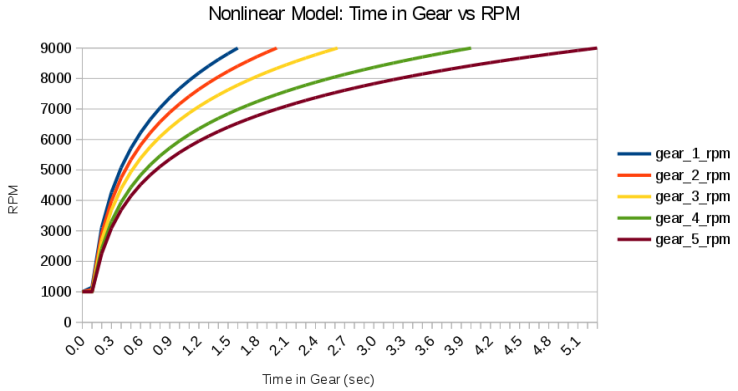
The total speed is the total distance divided by the total time:

```
speedTotal = (distanceTotal / FEET_PER_MILE) / (timeTotal / SECONDS_PER_MINUTE / MINUTES_PER_HOUR)
```

Part 2: Automatic Shifting with Nonlinear Engine Model

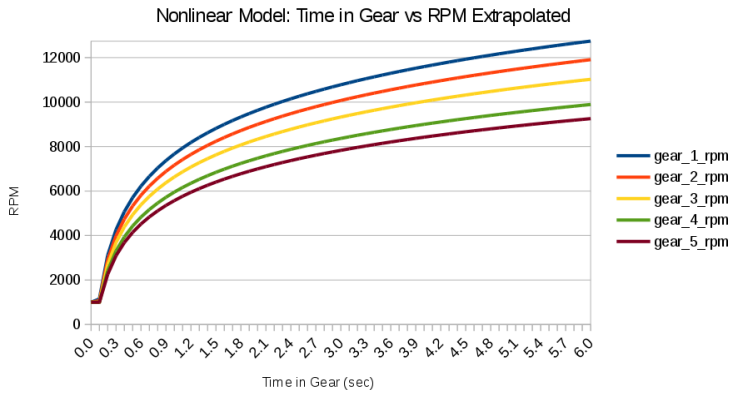
Part 2 is identical to Part 1, except that we use `EngineModelNonlinear` instead of `EngineModelLinear`. This is employing the Strategy design pattern to swap out a component. Everything else remains the same. In this way, we can compare the performance of the linear and nonlinear models objectively because they are the only difference.

Graph 4 (in tab *nonlinear time vs rpm*) is the same analysis as in Graph 1. In this case, however, the RPM does not increase as a straight line in each gear. This performance is more realistic, although still simplified.



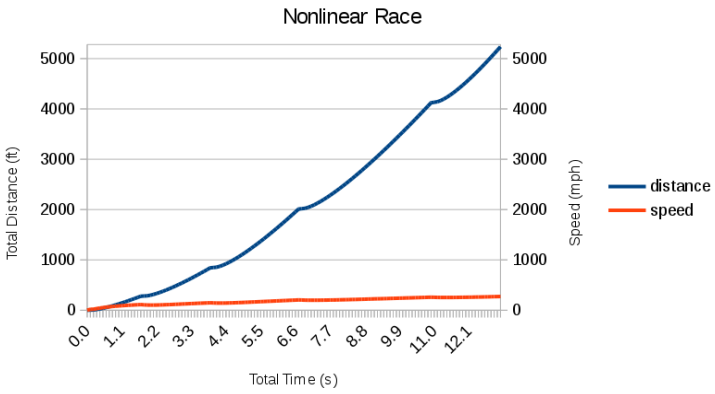
Graph 4

Graph 5 (in tab *nonlinear time vs rpm extrap*) is similar to Graph 2. Notice, however, that the extrapolated RPM beyond the 9,000 shift threshold is not at all unreasonable now. On the other hand, there is not a lot of performance gain above this point, which is why we need the next gear.



Graph 5

Graph 6 (in *nonlinear race*) is like Graph 3.



Graph 6

There should be no difference in the implementation in `update()` depending on the model. The only thing that changes is how the engine model reports RPM as a function of time, which is already done for you. Your job is to make sure the same implementation works with both engine models. If your solution needs to know which model it is using, then you are doing something wrong. Think about polymorphism in object-oriented programming.

Part 3: Manual Shifting with Linear Engine Model

The automatic transmission in Parts 1 and 2 is basically perfect. In other words, the program shifts as closely to 9,000 RPM as possible. (In reality, it sometimes overshoots a little because the time step does not conveniently land on this value, but this limitation is a result of the simulation, not the model.) There is no variation in performance because the computer is consistent. The manual transmission in Part 3, on the other hand, models the variation in a human performing the shifts. Here you will implement the `update()` method in `DragsterManual`.

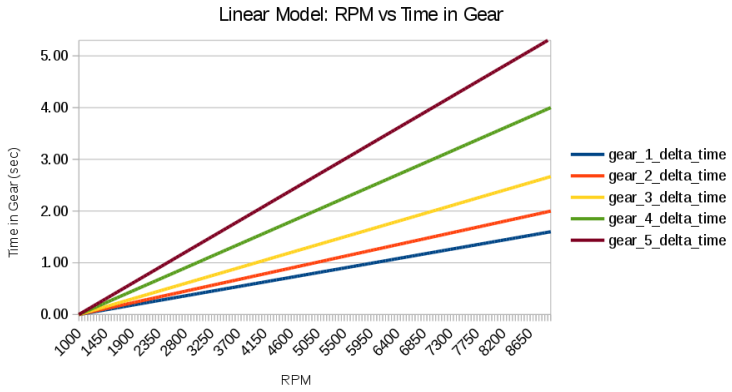
We assume that human reaction time for shifting is ±0.125 seconds. This definition means that the human driver random may shift early or late by up to an eighth of a second. (In extremely rare cases, it could be larger.) The random value is normally distributed with a bell curve that says most of the time, the error is around 0 seconds. Use the following equation:

```
shiftTimeOffset = random.nextGaussian() * SHIFT_RESPONSE_SECONDS
```

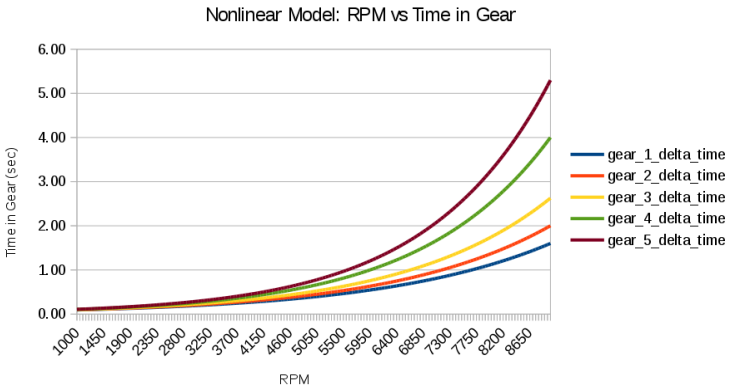
The `random` variable is an instance of `java.util.Random`. Precalculate the errors for the four shifts (first to second, etc.) before the race starts, not during it.

This error is a function of time because humans have natural reaction times in making decisions. The resulting error in the manual transmission, however, should be in RPM. In other words, shifting early or late by up to an eighth of a second results in shifting at less or more than the target 9,000 RPM, respectively. Less wastes engine performance; more could destroy the engine (although we ignore this aspect).

In order to translate a time error into an RPM error, we use the inverse method in the engine models. Graph 1 was based on `getRPM()` in `EngineModelLinear`, where providing the time in a gear produces the RPM. For the inverse method `getTimeInGear()`, providing the RPM produces the time at which it would be achieved in this gear. Graphs 7 and 8 (in tabs *linear rpm vs time* and *nonlinear rpm vs time*, respectively), show the inverse curves. The optimal shift RPM is always 9,000, which is at the right side of each graph. Method `getTimeInGear()` returns when this shift should occur in terms of time in gear. By adding the error (which can be negative), we get the time that it actually occurred at. Putting this value into `getRPM()` produces the RPM at which we actually shift.

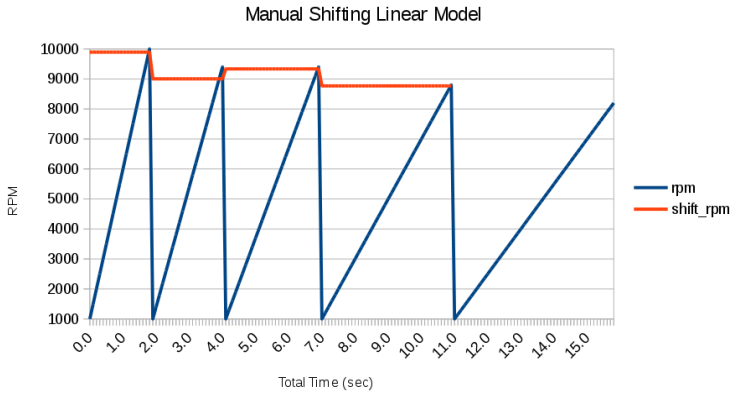


Graph 7



Graph 8

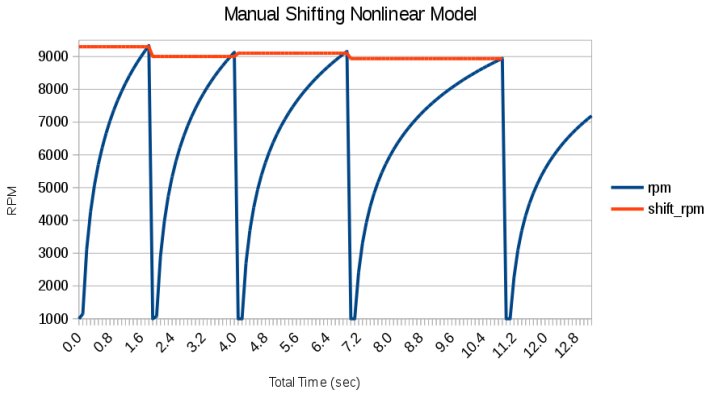
Part 3 needs to produce a single graph that characterizes an interesting shift pattern. For example, Graph 9 (orange line) overshot shifting into second gear by almost 1,000 RPM! (Interpret "interesting" to mean there is noticeable variation between gears. If your orange line is almost straight, as probability says it should be, choose a different random seed.)



Graph 9

Part 4: Manual Shifting with Nonlinear Engine Model

This part is identical to Part 3, except use `EngineModelNonlinear`. Again, there should be no difference in your `update()` method depending on the engine model. Graph 10 shows interesting performance. Notice also that Graphs 9 and 10 show that the dragster reached the end of the track before reaching maximum RPM in fifth gear. This analysis is not part of your task, but it is part of what we are learning to recognize.



Graph 10

Part 5: Automatic Shifting with Linear Engine Model and Better Resolution

This part is identical to Part 1, except use a time step of 0.01 seconds. Compare your results.

Deliverables

Submit your Java code for `DragsterAutomatic` and `DragsterManual` only. No other code should need changing except `RaceSimulator`. It does vary according to which part you are doing, but I will use my version.

Submit a nicely formatted PDF document that minimally contains the following for each part:

Part 1: your equivalent to Graph 3
Part 2: your equivalent to Graph 6
Part 3: your equivalent to Graph 9
Part 4: your equivalent to Graph 10
Part 5: your equivalent to Graph 3 and a comparison with Part 1

Also state any observations or issues.

Submit your spreadsheet in Excel or OpenOffice Calc format. Make sure it is clear which tab refers to which part.

Submit these four files through the Task 1 due date link on Shelby.