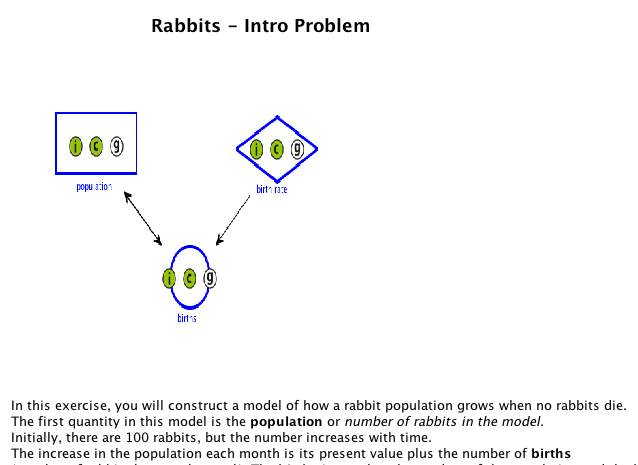
Introduction to Dragoon

# Initial training problem

With your browser, go to <http://dragoon.asu.edu>, select Introduction, enter your name, and select the first problem, “Rabbit Problem.” This will cause a file to be downloaded. On some web browsers, you will need to click on the downloaded file to get to start the Java program, otherwise you will get a series of security-related pop-up windows. After navigating past the security warnings, you should see this:



Dragoon is a tool for creating models. Your goal for this first exercise is to learn how to use the Dragoon user interface. This problem will give you step-by-step instructions for creating the model. These instructions will be absent during later exercises, so try to understand what they are teaching you about the model-building process. Go ahead and construct this model.

# Training problem with one accumulator node

In the preceding exercise, you learned how to use the Dragoon user interface. In fact, you used all the buttons, tabs, etc. that it has. This section, we will discuss what a model is and then discuss how to construct a model on your own.

As you have seen, a model consists of a set of nodes (shaped like diamonds, circles and rectangles) connected by links. Each node represents both (a) a numeric quantity whose value may change over time, and (b) a mathematical function for calculating the value of that quantity. The links represent inputs. That is, when node *A* needs a quantity represented by node *B* as an input to the function that calculates *A*’s value, then there is a link into *A* from *B*. You can think of numbers flowing along the links. When a model is complete, all the calculations are performed by the computer. You can see how each node’s value changes over time by clicking on the Show Graph button.

This kind of model is called a “system dynamics” model because it *represents how a system changes over time*, where the system is modeled (represented) by a set of quantities. System dynamics models are widely used in science, engineering, management, and government.

Now for the details: Over the next few exercises, you will learn how to decide what quantities to include in your model and how to define nodes that represent the quantities and their interrelationships.

## Exercise 2: Jared’s weight

Jared is concerned about his weight. Suppose he starts at 200 pounds and loses 2 pounds a week. In our modeling language, you would use an *accumulator* node to represent Jared’s weight. It would start with an initial value of 200 (we will use units of pounds throughout). Each week, the accumulator node’s value should have 2 subtracted from it, because Jared loses 2 pounds a week. That is, every week the calculation is:

Next week’s weight = this week’s weight – 2

In general, if you know how much a value *changes* each week, then you use an accumulator node. For instance, if you know that Jared’s weight *increases* each week by 1 pound, then you set up the accumulator to be:

Next value = current value + 1

Thus, the increase or decrease of a quantity is determined by the sign of the associated constant.

Now, suppose you are only told that Jared’s weight increases by *x*, where *x* is some complicated formula involving how much he ate during the week. If he eats too much, then *x* is positive and Jared gains weight. If he eats properly, then *x* is negative and he loses weight. To model this, you again use an accumulator and make its calculation be:

Next value = current value + *x*

During the next exercise, you will use the model editor to create a model for a simple system: Jared’s weight changes. A node has already been created that represents Jared’s weekly gain or loss, but it is named “weight\_change” instead of *x*. You can assume that Jared starts at 200 pounds. Please return to the Introduction page and select the second problem, “Jared’s weight.”

# Training problem with one function node (addition)

In the preceding exercise, you learned that you should use an accumulator node when you are told how much a quantity *changes*. Now let’s consider a different case.

Suppose Jared is a diet tester for Oprah Winfrey. He eats exactly what she eats, and does the same exercises too. He and Oprah compare weights each week, and it turns out that Jared is always 50 pounds heavier than Oprah. Thus, when Oprah weighs 120 pounds, then Jared weighs 170 pounds (170 is 120+50). When Oprah’s weight goes down to 100 pounds, then Jared’s weight goes down to 150 pounds. Now in this case, we use a function node to represent Jared’s weight, and the calculation is (in units of pounds):

Jared’s weight = 50 + Oprah’s weight

In general, we use a function node whenever we can calculate a quantity *directly* in terms of other quantities.

In the preceding exercise, you were told how much Jared’s weight *changed* each week e.g., he lost 2 pounds a week. Here you are *not* told anything about how Jared’s weight *changes* each week; you are told what his weight *actually is* each week, namely, 50 + Oprah’s weight. So the difference between using an accumulator node and using a function node depends on whether you are told about the *change* in a quantity’s value or about the *actual value itself.*

So go ahead and make a model assuming Jared always weighs 50 pounds more than Oprah. Please return to the Introduction page and select the third problem “Jared & Oprah.”

# Training problem with one function and one accumulator.

In the preceding two exercises, you learned when to use an accumulator node and when to use a function node. That is, you should use an accumulator node when you’re told about the *change* in value, and you should use a function node when you’re told that *the value itself is a function of other values*. You probably have guessed that you should use a fixed value node (diamond shape) when you are told a specific number for a quantity. These three kinds of nodes (accumulator, function and fixed value) are the only kinds. So you now know everything.

In general, a model can have several accumulator nodes, several function nodes and several fixed value nodes. The more nodes a model has, the more confusing it can be to construct it. In the remaining exercises, you’ll learn how to create gradually more complex models.

In this exercise, you’ll create a model with four nodes:

* *Jared’s weight*: Jared always weighs a certain amount more than Oprah.
* *Jared’s extra weight*: This is 50 pounds, which is how much more he weighs than Oprah.
* *Oprah’s weight*: It starts at 140 pounds and decreases by a certain amount a week.
* *Oprah’s weekly weight loss*: She loses 1.5 pounds a week.

In general, you should finish one node completely before starting to work on another node. For now, the software will require you to do this. Later you will have the freedom to edit nodes in any order you like. But it is still a good idea to complete one node before working on others.

Please return to the Introduction page and select the fourth problem “Jared & Oprah +.”

# Training problem with accumulator & one function (multiplication)

The previous Jared & Oprah exercises used addition and subtraction, but not multiplication and division. This exercise introduces the use of multiplication in calculations.

Suppose that Jared always weighs 1.3 times as much as Oprah. That is, when she weighs 100 pounds, then Jared weighs 130 pounds. When Oprah weighs 120 pounds, then Jared weight 156 pounds (156 is 1.3\*120). Note: Dragoon uses \* to denote multiplication.

Next, you will build a model involving Jared and multiplication. To save you work, the nodes for Oprah’s weight and weekly weight loss have been defined already. Please return to the Introduction page and select the fifth problem “Jared & Oprah \*.”

# Training problem with exponential decay

This exercise introduces a new kind of complexity: a loop in the links (this is sometimes called feedback). You may recall seeing such a loop in the first problem you did, where you created a model of rabbit population growth. The key assumptions of this model are:

* The rabbit population changes only when rabbits are born. Rabbits do not die, emigrate, immigrate or transmute themselves into princesses.
* The number of rabbits born each month is 10% of the current rabbit population. For instance, if the rabbit population is 150, then 15 rabbits are born that month, so next month the rabbit population is 165.



The function node, “births”, has as its calculation “population \* birth\_rate.” Thus, “population” must be an input to “births.” On the other hand, because the calculation for the accumulator node “population” also refers to “births,” “births” must be an input to “population.” The loop expresses the fact that the change in population depends on the population itself.

The last exercise has you create a model containing a loop. The model assumes that the Dodo (<http://en.wikipedia.org/wiki/Dodo>) became extinct because invasive species ate its eggs and thus prevented births. All the numbers are, however, made up just for this exercise. Please return to the Introduction page and select the sixth problem “Dodo extinction.”