Systems Dynamics Modeling with Dragoon: An introduction

Systems Dynamics Modeling is a method of understanding systems (i.e., parts of the world that you are interested in) that change over time (i.e., they are dynamic). It works by viewing the system as interacting quantities—called “nodes” in the Dragoon software. A model is made by defining both nodes and the relationships between them. In particular, if Node A has an effect on Node B, Node A is considered an “input” of Node B. Once all the nodes and inputs are defined, the computer can calculate the change in the system over time and provide you with graphs showing the results.

Parameters, Accumulators, Functions

Models in Dragoon are made up of three types of nodes: Parameters, Accumulators, and Functions.

- Parameters are the simplest type of node. A parameter represents a **fixed quantity that never changes**. Parameters are used to represent constant values (e.g., height of building being 12 m).

- An accumulator represents a **quantity that accumulates the values of its inputs over time**. As each input arrives, it changes the value of the accumulator, so inputs represent changes to the accumulator’s value. At each time step, the inputs are added or subtracted from the value for the accumulator at the current time step—that determines its value at the next time step.

- Functions represent a **value that is directly related to the values of its inputs, without regard to its own previous value**. The inputs to functions determine the value of the function. A function differs from an accumulator in that it does not need to know its previous value in order to determine its next value.

Accumulators and functions seem very similar at first, and it can take some practice to determine if a particular quantity should be represented as one or the other. To determine if a quantity should be an accumulator, ask the question: “Does this quantity add or subtract new values to itself each time step to find its new value?” If it does, then the quantity is an Accumulator. If it does not, then the quantity is a Function. Additionally, Accumulators typically have an initial value associated with them, whereas functions do not.

**Example system:** Let’s start by modeling a simple system: a population of dodo birds as they were becoming extinct. Suppose that there were 1000 dodo birds (pronounced “doh doh”) in the population initially and that 20% of the dodo birds
died each year. Thus, the number of deaths in the first year was \(1000 \times 20\% = 200\) deaths, leaving 800 dodos alive at the beginning of the second year. The number of deaths in the second year was \(800 \times 20\% = 160\) deaths, leaving 640 dodos alive for the start of the third year. Because the number of dodo birds and the number of deaths change over the years, this is a dynamic system.

**Problem 1:** To make sure you understand the system's behavior, please fill in the next few years in the table below.

<table>
<thead>
<tr>
<th>Year</th>
<th>dodo population at the start of the year</th>
<th>Deaths during the year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year 0 (starting values)</td>
<td>1000</td>
<td>200</td>
</tr>
<tr>
<td>Year 1</td>
<td>800</td>
<td>160</td>
</tr>
<tr>
<td>Year 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It would be tedious to continue calculating this by hand, so let us build a model to represent this dynamic system of a shrinking dodo population. Let’s start by defining our nodes, of which there are 3:

- **Death rate** is the percentage of the dodo population that dies each year. This is given to us as a constant value of 20% each year, so it is a **Parameter**. Because Dragoon doesn’t understand percentages, the value of death rate is 0.2.
- **Population** represents the number of dodos still alive. We are given an initial value for this quantity, 1000 dodos, and the number of dodos alive at the end of each year is dependent on the number alive in the previous year (new population = old population – deaths). Thus we know the population node is an **Accumulator**.
- **Deaths** represents the number of dodos that die in a given year. This is a quantity that is completely determined by the value of two other inputs (population \(\times\) death probability). Note that the number of deaths each year does not require knowledge of the deaths of the previous year to compute; this means the deaths node is a **Function**.

A Dragoon model for this system would look like this:
This model has the Parameter, Accumulator, and Function nodes discussed above. Different types of nodes have different shapes. Parameter nodes are diamonds, and the inside of a Parameter node shows its value. Function nodes are circles. The inside of a Function node shows the mathematical operation that combines its input. The inside of an Accumulator node, which is a square, shows both its mathematical operation and its initial value.

You create such nodes by clicking on a “Create node” button and filling out the form that it pops up. The form is called the node editor. You can also click on an existing node to edit it. Below are snapshots showing the node editor just as each node has been fully defined and the node editor is about to be closed. See if you can figure out which node editor snapshot goes with which node above.

Notice that the node editor snapshots look slightly different from each other. Do the differences make sense to you?

If you could click on the Table button, you would see that the software calculates the declining population identically to your work by hand (see left figure, below); the Graph button would display the same information in graph form (see right figure).
While the dodos extinction system is simple, other systems may be much more complex. Their predictions would be exceptionally difficult to compute by hand, but by constructing a Dragoon model, we can have Dragoon calculate the predictions which makes it much easier to understand the system.

How to open problems

Now let's start learning how to make models using Dragoon. The rest of this worksheet will alternate between reading about model construction and solving model construction problems on Dragoon.

When the worksheet asks you to solve a problem, it will give you a problem category name (such as “Miscellaneous”) and a problem name (such as “dodo extinction”). However, exactly how you open that problem depends on what type of user you are. If you don’t know what type of user you are, please click on the menu item Introduction at the top of dragoon.asu.edu. User types are explained at the bottom of that page.

- If you are a student, you should log in via the URL that your instructor gave you, and then navigate to the problem.
- If you are a camper, then you should login via the Login tab at the top of dragoon.asu.edu, and then navigate to the problem.
- If you are a visitor, you should navigate to the problem via the Problem menu item at the top of dragoon.asu.edu.

A linear increase problem
The simplest type of system is one where there is a single quantity that goes up or down by the same constant amount each time step. For example, consider this system:

Rod is a bookworm. He already has 100 books. Each week, he buys two more books. Construct a model that shows how many books he has each week.

The time step here is a week. The main quantity here is the number of books that Rod owns. It starts at 100, and then rises to 102 during week 1, to 104 during week 2, to 106 during week 3, etc. If you were to graph the number of books Rod owns versus the weeks, the graph would be a straight line tilted upward. That’s why this kind of system is said to exhibit linear growth or linear increase.

One way to solve such problems is to think about each quantity, decide how it changes, and define a node for it. Let’s try that method on the bookworm problem.

The problem has two quantities. Please decide which type of node is appropriate for each quantity, and circle it in the table below:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Node Type (Circle One)</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of books Rod owns</td>
<td>Accumulator - Function - Parameter</td>
</tr>
<tr>
<td>number of books Rod buys each week</td>
<td>Accumulator - Function - Parameter</td>
</tr>
</tbody>
</table>

Hopefully, you decided the number of books Rod owns is an accumulator and the books he buys each week is a parameter. Now we need to decide about inputs for each node. A parameter never has inputs, because its value is just a constant (2 in this case). The accumulator, namely Rod’s books, goes up by 2 each week, so it needs an input whose value is 2. Thus, the accumulator needs to have the parameter as its input. The resulting model is:

Now solve the “Bookworm” problem in the “Miscellaneous” problem category, selecting “Immediate Feedback (Student)” mode if you are given a choice. Start by clicking on the Create Node button. Your model need not look exactly like the one above—the positions and colors of the nodes do not affect the calculations of the model. You can move nodes by dragging on them.
When you have finished constructing the model, click on the Graph button to see how the main value of “books” changes over time. Please manipulate the sliders on the right side of the Graph window; do the movements of the graph make sense to you?

This kind of model is called *linear* because the graph of the main quantity, books, is a straight line. Because the main quantity increases, it’s called a *linear growth* model.

Now click on the Done button.

**Generic models**

In order to model a complicated system, you must decompose it into elementary processes that can each be easily modelled. In the case of Rod’s books, the system is so simple that there is just one process: Books changing state from being owned by a bookseller to being owned by Rod, and the key feature of this process is that the amount of change per time step is constant, namely 2 books per week. A handy format that summarizes this analysis is:

- **Process:** Buying books
- **Source of books:** Irrelevant to this problem (probably a bookseller)
- **Destination of books:** Rod’s book collection
- **Number changing per time step:** a constant, 2 books, per week

Once you have such an analysis, it is straightforward to convert it to a Dragoon model. In fact, for simple analyses like this, you can look up a generic model in the problem category “Generic changes” and just copy it. Here is the generic model named “Constant” in that problem category:

There are numbers and names shown here, but they are just examples. When you copy the model, you’ll use your own names and numbers. For the Bookworm problem, we only need part of the model, because the source of Rod’s books isn’t specified or relevant. Thus, you would copy only the nodes “changing” and “destination.” Compare the right half of the generic model above to the Bookworm model shown earlier. See how the copy/change idea works?

**A linear decrease problem**
Now let’s look at another linear model. In the problem category “Mastering model construction: Problem set 1” open the problem “MMC: Problem 1.3”. Please select “Immediate Feedback (Student)” as the mode if you are given a choice.

Please try solving the problem. Note that because Jared’s weight is decreasing, a subtraction is required. When you get around to defining the expression for Jared’s weight, you’ll need to click on the “—” button first and then select “weight change”. When you get done, you’ll see “new Jared weight = old Jared weight + - weight change.” The fact that there is a plus sign followed by a minus sign seems odd to some people, but it is actually completely proper math notation.

When you get done, your model should look something like this:

![Diagram of a linear decay model](image)

You can drag nodes around to make the model easy to understand. Positions don’t affect the computations of the model at all. Also, your colors might be different from the ones shown above. Here’s the code:

- A dotted border means incomplete. That is, at least one of the entries inside the node has been left blank.
- A red border means something inside the node is still wrong.
- A yellow border means that at least one of the entries inside the node was supplied by Dragoon because the user guessed too much.
- A green border means the user got everything right, eventually.
- A green interior means the user got everything right on the first try—congratulations!

Notice the minus sign inside the circle near the arrowhead. This indicates that the input is subtracted from the accumulator’s old value. When the minus sign is absent, as it was in the Bookworm model, then the input is added to the accumulator’s value. These details mean that you don’t have to open a node in order to see what expression it has inside it.

This model is called a linear decay or linear decrease model because the graph of the main quantity is a line and it goes downward.

A linear transfer problem
Now let’s build a third kind of linear model. Please open the problem “MMC: Problem 1.9” in the problem category “Mastering model construction: Problem set 1,” selecting “Immediate feedback (Student)” as the mode if you are given a choice. You can use either the first solution method (figuring out a definition for each node) or the second solution method (copying the appropriate generic model).

When you have finished constructing the model, be sure to look at the graphs and play with the sliders. This kind of system exhibits a linear transfer behavior, because it includes both a source and a destination, and the graphs of both quantities are straight lines.

**Exponential change problems**

The dodo extinction problem, which was discussed earlier, is an example of a system with an exponential decrease in its main variable. That is, the amount of decrease in the main variable is proportional to its value. The corresponding generic model is called “proportional to source.” Try solving the problem “dodo extinction” in the problem category “Miscellaneous,” selecting “Immediate feedback (student)” if given a choice of mode. If you move the sliders, is the graph always an exponential curve?

Now try the problem “two barrels” in the problem category “Miscellaneous,” selecting “Immediate feedback (student)” if given a choice of mode. Notice that this problem requires you to use units (i.e., gallons) whenever it makes sense to do so. This system exhibits exponential transfer, where the source is governing the rate of change of the system.

In a system that exhibits exponential growth, the amount of change per time step is a constant proportion of the destination. The generic model is called “proportional to destination.” Please try the problem “Giardia” in the problem category “Miscellaneous,” selecting “Immediate feedback (student)” if given a choice of mode. You may be surprised to find that there are more node descriptions than you need.

**Combining two or more change processes**

Systems can combine two or more change processes. Please try the problem “MMC: Problem 2.2” in the problem category “Mastering model construction: Problem set 2,” selecting “Immediate feedback (student)” if given a choice of mode.

**The Target Node Strategy**

We have done several problems now, all corresponding to the generic models. As you can see, copying a generic model can be easier than figuring out the model by considering each quantity individually. However, sometime the system doesn’t
correspond to any generic model. Then you have no choice and must figure out the model quantity by quantity.

When systems get complicated, or fail to match a generic model or have extra information that isn’t needed in the model, you can waste a lot of time creating unnecessary nodes. This can also create a confusing mess. To avoid creating unnecessary nodes, you may use the Target Node Strategy. The basic idea is to start with a quantity that you know you will need, namely the quantity that the problem asks you to graph, and make it your “target.” Create a node for it, and be sure to create the expression it needs. As a side effect of creating the expression, you will probably get some new nodes created, which appear as triangles. When you are completely done with the first target node (the border is not dotted), then choose any one of the triangular nodes as your next “target.” Repeat this process until there are no triangular nodes left.

☐ Try the problem “MMC: Problem 3” in the problem category “Mastering model construction: Problem set 3.3”. This time, choose “Coached” as the mode if you are given a choice. You may see a new color, blue, which means that the entry you made is not wrong, but it is premature according to the Target Node Strategy. You should define it later.

Adding ordinary algebraic relationships

Sometimes it is necessary to include ordinary algebraic relationships in a model. For instance, the next problem asks you to graph the ratio of two quantities. Thus, you’ll have to define a function node to represent that ratio, and its expression will have a division in it.

☐ Please solve “MMC: Problem 3.6” in the problem category “Mastering model construction: Problem set 3,” selecting “Coached” as the mode if you are given a choice. When you get done, look carefully at the node that represents the ratio. It should reveal its contents by showing an multiplication symbol inside it and one of the inputs will have a division sign near its arrowhead, indicating that it’s quantity is inverted before being included in the product. These little details allow you to see all the math in a model without having to open the node editor.

More generic models

☐ Generic models for linear change and exponential change have been introduced, but there are other generic models as well. For instance, one of them is called “propor. to set point minus destination” in the “generic models” problem category. This kind of change is also known as “goal seeking” or “homeostasis.” Try the problem “MMC: Problem 2.7” in problem category “Mastering model construction: Problem set 2” and choose “Coached” as the mode if you have a choice.
Simple system for which there are no generic models

Although generic models, alone or in combination, work for many systems, there are many other systems that cannot be modelled in such a way. Thus, you will sometimes need to figure out how to solve such problems by considering each quantity and deciding how to model it. The Target Node Strategy can be a big help.

Try the problem “MMC: Problem 3.8” in problem category “Mastering model construction: Problem set 3” and choose “Coached” as the mode if you are offered a choice.