Tutorial : dynamicalSystems API

Author : Wellington Bruno da Silva (brunoufop@msn.com)

Leader : M.Sc. Bernardo Giori Ambrósio : Master in Computer Science

Co-supervisor: M.Sc. Bruno Rabello Monteiro *:* Master in Computer Science

This tutorial aims to show how to use the dynamicalSystems API.

Will be shown three examples on Stock and Flow diagram, and after each one, its relative source code on how you would use on the dynamicalSystems API. The models were constructed using the Vensim Software.

Example 1

A Rabbit Population Model



The equations for the Model above:

average lifetime = 8

birth rate = 0.125

births = Population \* birth rate

deaths = Population / average lifetime

Rabbit Population = INTEG(births - deaths,1000)

The Simulation information:

Time Step = 0.125

Final Time = 30

Below, the source code for this model, using the dynamicalSystems API.

**package** example1;

**import** java.io.IOException;

**import** dynamicalSystems.\*;

**public** **class** Main {

**public** **static** **void** main(String [] args){

/\* The source code using the dynamicalSystems API must be surrounded

\* by a try/catch, because may occur a IllegalArgumentException

\*/

**try** {

/\*

\* Firstly, is needed construct the model's elements.

\* Here, are constructed the Converters

\* Birth Rate initializes with the value 0.125

\* Average LifeTime initializes with the value 8

\*/

Converter c1 = **new** Converter("Birth Rate",0.125);

Converter c2 = **new** Converter("Average LifeTime",8);

/\* Here, we construct the Flows

\*/

Flow f1 = **new** Flow("Births");

Flow f2 = **new** Flow("Deaths");

/\*Finally, we construct now the Stock

\* Population has 1000 as an initial value

\*/

Stock s1 = **new** Stock("Rabbit Population",1000);

/\*After constructs all the model's elements,

\* we must connect each one with its determinants.

\* A determinant is one element which influences the value of another

\* The constants, don't need to be connected ( Note that c1 and c2 are constants)

\*/

f1.insertDeterminant(s1);

f1.insertDeterminant(c1);

f2.insertDeterminant(s1);

f2.insertDeterminant(c2);

/\* For the Stocks, we don't insert determinants.

\* Indeed, we insert inFlows or OutFlows.

\*/

s1.insertInflow(f1);

s1.insertOutflow(f2);

/\*Now, the elements are connected with their relative determinants,

\* we are going to create the relationships (equations) for each one.

\*/

/\*Creating a Relationship for the Flow 'Births'

\*

\* births = Rabbit Population \* birth rate

\*/

f1.connect(s1,Relationship.*PRODUCT*,c1);

/\*Creating now, the relationship for the Flow 'Deaths'

\*

\* deaths = Rabbit Population / average lifetime

\*/

f2.connect(s1,Relationship.*DIVIDED*,c2);

/\*The relationship for Stocks is automatically created

\* when you insert the in and out flows.

\*/

/\* Now, your source code model is finished.

\* All you got to do is create a Simulation object and

\* call the 'start' method to run the simulation

\*/

Simulation simulation = **new** Simulation();

**int** finalTime = 30,initialTime = 0;

simulation.start(Simulation.*\_0125*, initialTime, finalTime);

/\*Saving the result of simulation,

\* in a file on same directory, we're using

\* 'result.txt' is any name you want give to your file

\*/

Simulation.*sendToFile*("result.txt");

} **catch** (IllegalArgumentException e) {

e.printStackTrace();

}

**catch** (IOException io) {

io.printStackTrace();

}

}

}

Example 2

A changed Rabbit Population Model



The Equations for the model Above :

Rabbit Population = INTEG (births-deaths,initial population)

initial population = 1000

birth rate= 0.23

average lifetime = 8

births = Rabbit Population \* birth rate

deaths = (Rabbit Population / average lifetime) \*effect of rabbit crowding on deaths

carrying capacity = 1000

effect of rabbit crowding on deaths = WITH LOOKUP(

Rabbit Population/carrying capacity,(0,0.9), (1,1), (2,1.2), (3,1.5), (4,2))

FinalTime = 30

Time Step = 0.125

The source code using the dynamicalSystems API is shown below

**package** example2;

**import** java.io.IOException;

**import** dynamicalSystems.\*;

**public** **class** Main {

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

**try**{

/\*

\* Firstly, we construct the Elements of the model

\*/

Converter birthRate = **new** Converter("Birth Rate",0.23);

Converter averageLifeTime = **new** Converter("Average lifeTime",8);

Converter initialPopulation = **new** Converter("Initial Population",1000);

Converter carryingCapacity = **new** Converter("Carrying Capacity",1000);

Converter effectOfRabbit = **new** Converter("Effect of rabbit crowding on deaths");

Flow births = **new** Flow ("Births");

Flow deaths = **new** Flow ("Deaths");

Stock rabbitPopulation = **new** Stock("Rabbit Population",initialPopulation.getValue());

/\*

\* We are going to insert the elements determinants now.

\*/

effectOfRabbit.insertDeterminant(rabbitPopulation);

effectOfRabbit.insertDeterminant(carryingCapacity);

births.insertDeterminant(birthRate);

births.insertDeterminant(rabbitPopulation);

deaths.insertDeterminant(averageLifeTime);

deaths.insertDeterminant(rabbitPopulation);

deaths.insertDeterminant(effectOfRabbit);

rabbitPopulation.insertInflow(births);

rabbitPopulation.insertOutflow(deaths);

/\*

\* We are going to connect now, the elements.Constants don't need to be connected

\*/

/\*

\* The object 'effectOfRabbit',needs a LookUp function.

\* Here,it is show how to create this function

\*/

// creates the LookUp object

LookUp lookUp = **new** LookUp();

//inserts now, the rows to the lookUp table

lookUp.getTable().add(**new** Row(0,0.9));

lookUp.getTable().add(**new** Row(1,1));

lookUp.getTable().add(**new** Row(2,1.2));

lookUp.getTable().add(**new** Row(3,1.5));

lookUp.getTable().add(**new** Row(4,2));

/\* now, we just connect the determinants of 'effectOfRabbit'

Note that the input to the LookUp function is 'Rabbit Population/carrying capacity'

So, we have to create a relationship containing this equation, and then,

connect to 'effectOfRabbit'

\*/

effectOfRabbit.setLookUp(lookUp);

Relationship r1 = **new** Relationship(rabbitPopulation,Relationship.*DIVIDED*,carryingCapacity);

effectOfRabbit.connect(r1,Relationship.*WITH\_LOOKUPS*);

/\*

\* Connecting the Flows now

\*/

births.connect(rabbitPopulation, Relationship.*PRODUCT*, birthRate);

/\*

\* The equation for deaths is (Rabbit Population / average lifetime)\*effect of rabbit crowding on deaths

\* So, we have to create a relationship for (Rabbit Population / average lifetime) and then

\* connect this relationship to the effect of rabbit crowding on deaths

\*/

Relationship r2 = **new** Relationship(rabbitPopulation,Relationship.*DIVIDED*,averageLifeTime);

deaths.connect(r2,Relationship.*PRODUCT*,effectOfRabbit);

/\*The relationship for Stocks is automatically created

\* when you insert the in and out flows.

\*/

/\* Now, your source code model is finished.

\* All you got to do is create a Simulation object and

\* call the 'start' method to run the simulation

\*/

Simulation simulation = **new** Simulation();

**int** finalTime = 30,initialTime = 0;

simulation.start(Simulation.*\_0125*, initialTime, finalTime);

/\*Saving the result of simulation,

\* in a file on same directory, we're using

\* 'result.txt' is any name you want give to your file

\*/

Simulation.*sendToFile*("result.txt");

}**catch**(IllegalArgumentException inv){

inv.printStackTrace();

}**catch** (IOException io) {

io.printStackTrace();

}

}

}

Example 3

Now, A simple model using If Then Else. ‘Work To Do’ Model.



The Equations for the model above :

max work accomplishment=8

task is active=IF THEN ELSE( Work Done < task size, 1, 0)

task size=180

work accomplishment= max work accomplishment \* task is active

Work Done = INTEG(work accomplishment,0)

Work To Do = INTEG(- work accomplishment,task size)

Final Time = 100

Time Step = 0.125

The source code using the dynamicalSystems API is shown below.

**package** example3;

**import** java.io.IOException;

**import** dynamicalSystems.\*;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**try**{

/\*

\* Firstly, we construct the Elements of the model

\*/

Converter taskSize = **new** Converter("Task Size",180);

Converter taskIsActive = **new** Converter("Task Is Active");

Converter maxWorkAccomplishment = **new** Converter("Max Work Accomplishment",8);

Flow workAccomplishment = **new** Flow("Work Accomplishment");

Stock workToDo = **new** Stock("Work To Do",taskSize.getValue());

Stock workDone = **new** Stock("Work Done",0);

/\*

\* Inserting now the element's determinants

\*/

taskIsActive.insertDeterminant(taskSize);

taskIsActive.insertDeterminant(workDone);

workAccomplishment.insertDeterminant(maxWorkAccomplishment);

workAccomplishment.insertDeterminant(taskIsActive);

workToDo.insertOutflow(workAccomplishment);

workDone.insertInflow(workAccomplishment);

/\*

\* Now, we create the relationships among the elements

\*/

/\*

\* The taskIsActive object, uses a IfThenElse condition.

\* Here, is explained how to create relationships containing IfThenElse.

\*/

Condition condition = **new** Condition(workDone,Condition.*MINOR*,taskSize);

/\*

\* We create now, the complete IfThenElse object. Note that, the second and third parameters

\* are constants (1,0). After it, we just connect the ifThenElse object to the taskIsActive object.

\*/

IfThenElse ifThenElse = **new** IfThenElse(condition,**new** Converter("Converter1",1),**new** Converter("Converter0",0));

taskIsActive.setIfThenElse(ifThenElse);

taskIsActive.connect(Relationship.*IF\_THEN\_ELSE*);

workAccomplishment.connect(maxWorkAccomplishment,Relationship.*PRODUCT*,taskIsActive);

/\*The relationship for Stocks is automatically created

\* when you insert the in and out flows.

\*/

/\* Now, your source code model is finished.

\* All you got to do is create a Simulation object and

\* call the 'start' method to run the simulation

\*/

Simulation simulation = **new** Simulation();

**int** finalTime = 100,initialTime = 0;

simulation.start(Simulation.*\_0125*, initialTime, finalTime);

/\*Saving the result of simulation,

\* in a file on same directory, we're using

\* 'result.txt' is any name you want give to your file

\*/

Simulation.*sendToFile*("result.txt");

}**catch**(IllegalArgumentException inv){

inv.printStackTrace();

}**catch** (IOException io) {

io.printStackTrace();

}

}

}