NetLogo-*Mathematica* Link

Tutorials and Examples

This document will show how to use the NetLogo-*Mathematica* add-on to application interface. The notebook has examples that make use of sample models included with NetLogo. Click the right-most bracket of each cell to collapse a section.

Warning: It is important that you follow the procedure described in *Installing NetLogo-Mathematica Link* and execute the code in *Starting NetLogo* before continuing with the tutorial. You may skip around sections, but within each section, you must start from the top and execute each consecutive command. Sections will *not work properly* if you do not follow the instructions step-by-step.

Installing NetLogo-*Mathematica* Link

To install the NetLogo-*Mathematica* link, go to the menu bar in *Mathematica*, click on *File* and select *Install...* In the Install *Mathematica* Item dialog, select *Package* for *Type of item to install*, click Source, and select *From file...* In the file browser, go to the location of your NetLogo installation, click on the Mathematica Link subfolder, and select NetLogo.m. For *Install Name*, enter *NetLogo*. You can either install the NetLogo link in your user base directory or in the system-wide directory. If the NetLogo link is installed in the user base directory, other users on the system must also go through the NetLogo-*Mathematica* link installation process to use it. This option might be preferable if you do not have permission to modify files outside of your home directory. Other-wise, you can install NetLogo-*Mathematica* link in the system-wide *Mathematica* base directory.

Starting NetLogo

Once installed, the NetLogo package can be loaded at any time with the following command:

```
<< NetLogo`
```

To start NetLogo simply type the following command, and use the file browser to locate the NetLogo parent directory, typically located in "/Applications/NetLogo 4.0" or "C:\Program Files\NetLogo 4.0", on Macintosh and Windows systems, respectively.

```
NLStart[]
```

The NetLogo-*Mathematica* Link will store this path in **$NLHome**

```
$NLHome
```

One can also manually specify the **$NLHome** directory by hard coding in your NetLogo installation directory. This is preferable in many real-world scenarios when one uses the NetLogo-*Mathematica* Link often.

```
$NLHome = "/Applications/NetLogo 5.3.1/";
$NLHome = "C:\\Program Files\\NetLogo 5.3.1";
```

Once again, to start NetLogo using the default path (now specified by $NLHome) enter

```
NLStart[]
```

An Overview of the NetLogo-*Mathematica* Link using Fire

Loading a NetLogo model

Use **NLLoadModel[]** to load the Fire example from the models library.

```
NLLoadModel[
  ToFileName[{$NLHome, "models", "Sample Models", "Earth Science"}, "Fire.nlogo"]];
```

Executing NetLogo commands

The **NLCommand[]** function lets you execute any NetLogo command as if you were typing from the command center.

```
NLCommand["setup"];
```

```
NLCommand["set density 25"];
```

The function **NLCommand[]** automatically splices expression into NetLogo strings, making it easy to pass sequences of strings, numbers, lists, and colors into NetLogo without having to manually convert data types and join strings.

```
NLCommand["set density", 50];
```

Splicing can be very useful for setting NetLogo sliders using *Mathematica* variables

```
d = 65;
NLCommand["set density", d];
```

It is also possible to specify several command sequences using a single **NLCommand[]**

```
NLCommand["set density", d - 10, "show density", "setup"];
```

Repeatedly executing NetLogo commands

This loop calls **NLCommand[]** 10 times.

```
Do[NLCommand["go"], {10}];
```

The command **NLDoCommand[]** is an easier and efficient way to execute a command repeatedly.

```
NLDoCommand["go", 10];
```

Reporting data from NetLogo

You can retrieve data from NetLogo using **NLReport[]**

```
NLReport["count turtles"]
```

Repeat reports *n* times

One of the simplest uses of the NetLogo-*Mathematica* link is to repeat a command and report information after each successive command.

A way carry out these kinds of repetitive tasks is to use **NLCommand[]** and **NLReport[]** in combination with **Table[]**.

```
Table[NLCommand["go"];
 NLReport["(burned-trees / initial-trees) * 100"], {20}]
```

Tasks like these can be more easily and efficiently executed with **NLDoReport[]**, which will successively execute a command and return a reporter *n* times.

```
NLDoReport["go", "(burned-trees / initial-trees) * 100", 20]
```

Repeat reports until a condition is met

NLDoReportWhile[] is similar to **NLDoReport[]**, but rather than executing n times, it executes until a condition is [not] met.

The following executes "go" and reports back the % of trees burned until there are no turtles (embers) left.

```
NLCommand["Setup", "set density", 55];
NLDoReportWhile["go", "(burned-trees / initial-trees) * 100", "any? turtles"]
```

- ### Defining a simple experiment

An interesting phenomena in the Forest Fire model is the abrupt change that occurs in size of forest fires as the density increases. In this example, we will write a short function which sets up the model and returns a list of the percentage of trees burned at each time step, until all embers have burned out.

```
FireTimeSeries[density_] := Module[{},
   NLCommand["set density ", density, "setup"];
   NLDoReportWhile["go", "(burned-trees / initial-trees) * 100", "any? turtles"]
  ];
```

Generate a list of densities to run the model with, ranging from 50 to 70 in increments of 2

```
densities = Table[density, {density, 50, 70, 2}]
```

Carry out the **`FireTimeSeries[]`** function with each density

```
NLCommand["no-display"];
fireData = Map[FireTimeSeries, densities];
```

- ### Plot time dynamics of each run

Now that we have recorded the time dynamics of each configuration, let's take a look at how the fire spreads in first configuration (density = 70)

```
ListPlot[
 First[fireData],
 AxesLabel → {"Time", "% Burned"}, PlotLabel → "Burn time series at density = 50"
]
```

With a little bit more work, we can plot all the time series data simultaneously.

```
(* create a color for each density *)
numColors = Length[densities];
densityColors = Table[Blend[{ {1, Yellow}, {numColors, Red}}, n], {n, numColors}];

(* makes each run equal length *)
maxSteps = Max[Length /@ fireData];
completedData = (PadRight[#, maxSteps, Last[#]] &) /@ fireData;

ListLinePlot[Thread[Tooltip[completedData, densities]],
 PlotStyle → densityColors, AxesLabel → {"Time", "% Burned"},
 PlotLabel → "Burn time series with varying tree densities"]
```

Each line represents the time dynamics of the Forest Fire model run with a different density. Lines are colored by density, ranging from low (yellow) to high (red). Put your mouse over a line to see a tooltip of the density used in each run.

- ### Plot the phase transition by plotting how (final states) % burned vary with density

```
(*pair each density with the final % burned from each run *)
finalStates = Map[Last, fireData];
densityBurnedPairs = Transpose[{densities, finalStates}];

ListPlot[densityBurnedPairs, AxesLabel → {"Density", "Final % Burned"},
 PlotRange → {0, 100}, PlotLabel → "Phase transition in the forest fire model"]
```

Comparing Empirical and Analytic Distributions in GasLab

Use the Histograms package

```
Needs["Histograms`"];
```

Load GasLab Free Gas, set it up with 100 particles, and let it run for a little while

```
NLLoadModel[
  ToFileName[{$NLHome, "models", "Sample Models", "Chemistry & Physics", "GasLab"},
   "GasLab Free Gas.nlogo"]
 ];

NLCommand["set number-of-particles 100", "no-display", "setup"];
NLDoCommandWhile["go", "ticks < 20"];
```

Reporting lists of values from NetLogo

The NetLogo-Mathematica link automatically converts NetLogo lists into Mathematica lists.
This is can be useful for examining distributions. Here, we execute the model for 20 "ticks" and report back the speed of each particle

```
NLReport["[speed] of particles"]
```

- ### Symbolic computing and NetLogo: validating the Maxwell-Boltzmann distribution

Set up the model with 500 particles and collect 40 readings of each particle's speed, every 50 steps.

```
NLCommand["set number-of-particles 500", "no-display", "setup"];

speeds = Flatten[ NLDoReport["repeat 50 [go]", "[speed] of particles", 40] ];
```

Compare distribution of speeds with the theoretical Maxwell-Boltzmann distribution for a 2D gas, $B(v) = v\, e^{\frac{-m v^2}{2 k T}}$

```
{k, m, T} = {1, 1, NLReport["mean [energy] of particles"]};

B[v_] := v E^((-m v^2)/(2 k T));
normalizer = ∫_0^∞ B[v] dv;
theoretical =
  Plot[B[v]/normalizer, {v, 0, Max[speeds]}, PlotStyle → {Darker[Red], Thickness[0.008]}];
empirical = Histogram[speeds, HistogramScale → 1];

Show[empirical, theoretical,
 PlotLabel → "GasLab energy distribution and the Maxwell–Boltzmann distribution"]
```

Screenshot Sequences with Termites

```
NLLoadModel[
  ToFileName[{$NLHome, "models", "Sample Models", "Biology"}, "Termites.nlogo"]];
NLCommand["setup", "no-display"];
```

Capturing NetLogo patch colors

One can use **NLGetPatches[]** to get values from patches.
In this case we are reporting back NetLogo patch colors.

```
ArrayPlot[NLGetPatches["pcolor"], ColorRules → {0. → Black, 45. → Yellow}]
```

- ### Collecting multiple "screenshots"

CaptureTermiteProgress[] asks the turtles to "go" 20 times and take a "screenshot" using **NLGetPatches[]**.

```
CaptureTermiteProgress[] := Module[{},
   NLDoCommand["ask turtles [go]", 20];
   NLGetPatches["pcolor"]
  ];
```

Set up the model, and repeat **CaptureTermiteProgress[]** six times to capture several "screen shots".

```
NLCommand["setup"]
patchShots = Table[CaptureTermiteProgress[], {6}];
renderedShots =
  Map[Rasterize[ArrayPlot[#, ColorRules → {0. → Black, 45. → Yellow}]] &, patchShots];
```

- **Display screenshot simultaneously in a grid**

Display each consecutive screenshot simultaneously in a grid

```
GraphicsGrid[Partition[renderedShots, 3], ImageSize → 500]
```

- **Animate screenshots**

Animate the screenshots and replay the model backwards and forwards

```
ListAnimate[renderedShots]
```

Plotting Terrain in 3D

Load the erosion model and set it up

```
NLLoadModel[
 ToFileName[{$NLHome, "models", "Sample Models", "Earth Science"}, "Erosion.nlogo"]]
NLCommand["set terrain-smoothness 15", "set rainfall 0.30",
 "set soil-hardness 0.8", "no-display", "setup"]
NLDoCommand["go", 120]
```

Execute with the new setup

- **Plotting elevation information in 3D**

NLGetPatches[] can report any kind of patch data, not just colors. For example, one can plot the patch variable *elevation* to construct a 3D terrain plot.

```
elevations = NLGetPatches["elevation"];

ListPlot3D[elevations, Mesh → None, ColorFunction → "Topographic",
 Mesh → None, Axes → None, Boxed → False, ViewPoint → {0.8, -1.5, 2.9}]
```

Plotting Networks with Preferential Attachment

Load the Preferential Attachment model

```
NLLoadModel[ToFileName[
  {$NLHome, "models", "Sample Models", "Networks"}, "Preferential Attachment.nlogo"]]
```

Set up the model and generate about 2000 nodes.

```
NLCommand["setup set layout? false set plot? false no-display"];
NLDoCommand["go", 2000];
```

Capturing Graphs in NetLogo

Capture the network with **NLGetGraph[]**

```
network = NLGetGraph["links"];
```

By default, **NLGetGraph[]** uses the generic link breed, *links*

```
network = NLGetGraph[];
```

NLGetGraph[] returns a list of rules of the form *outNode→ inNode,* which can be used by NetLogo's visualization functions, where *outNode and inNode* are the who numbers of agents in the network.

```
Short[network]
```

- **Visualizing NetLogo graphs**

Let *Mathematica* automatically pick a layout.

```
GraphPlot[network]
```

Or choose your own layout

```
GraphPlot[network, Method → "SpringElectricalEmbedding"]
```

- **Sparse matrix representation of NetLogo graphs**

```
Needs["GraphUtilities`"];
```

Rule-based network specifications, like the ones returned by **NLGetGraph[]** can easily be converted into sparse matrices

```
netMatrix = AdjacencyMatrix[network]
```

Plot the adjacency matrix

```
MatrixPlot[netMatrix]
```

- **Comparing binning methods to find power laws in networks**

Mathematica can operate on sparse matrices to find power laws in networks and other related phenomena

```
linearBinPlot = ListLogLogPlot[BinCounts[Total /@ netMatrix], PlotStyle → PointSize[0.019],
   PlotLabel → "Preferential Attachment with linear binning"];
bins = Table[2^i, {i, 8}];
expBinPlot =
  ListLogLogPlot[Transpose[{bins, PadRight[BinCounts[Total /@ netMatrix, {bins}], 8]}],
   PlotStyle → PointSize[0.019],
   PlotLabel → "Preferential Attachment with expoential binning"];

GraphicsRow[{linearBinPlot, expBinPlot}, ImageSize → 600]
```

Advanced Features: Headless Mode

To begin in headless mode, use option **Headless→True**

```
NLStart[$NLHome, Headless → True]
```

This mode is preferable for situations in which you do not need to interact directly with the NetLogo graphical interface.