Automated Data Analysis in NetLogo

Glenn Ledder

Department of Mathematics
University of Nebraska-Lincoln
gledder@unl.edu

March 17, 2023

Talk Structure

0. Introduction
1. The Base Simulation
   - Out in the Open
   - Under the Hood
2. Mathematical Modeling
3. Automated Data Analysis
   - Out in the Open
   - Under the Hood
4. Wrapping Up
   - Full Automation
   - Take-Home Messages

Introduction

- **NetLogo is a platform for coding agent-based models.**
  - Primary control is in the **Interface** tab, which contains a display window, **data entry boxes**, **buttons**, and **output**.
  - An **Info** tab contains program documentation.
  - A **Code** tab contains the program code.
- **forager.nlogo** runs a foraging simulation.
  - A forager wanders among food patches and empty space.
  - It stops to feed when it finds a patch.
  - Feeding ends when the patch resource level drops to **X**.
  - The forager loses energy while traveling.
  - Each empty patch regains resources over time.
  - We want to know the mean energy gain per time for given **X**.

forager Interface Components

1. A **slider** allows the user to set **X**, the resource level at which foragers leave a patch.
2. **setup** prepares the display.
3. A **display window** shows a forager and patches in real time.
4. **go** runs the simulation.
5. A **plot window** shows the average resource gain over time.
6. A **monitor window** shows the current average resource gain.

- Pressing **go** again stops the simulation.
    - The symbol in the lower right corner of the **go** button shows that the simulation will run until stopped manually.

NetLogo Code Structure

- Variable Declarations
  - Global Variables
  - Patch Variables
  - Turtle (agent) Variables
- Button Procedures
  - Buttons trigger blocks of code.
    - It is best to break the code for a button into separate tasks.
- Procedures
  - Blocks of code for components of the button procedures.
- Reporters
  - Functions that do calculations and are called by procedures.

forager Patch and Turtle Variables

```
patches-own ; variables belonging to each patch
[
  visit? ; true if occupied, false if vacant
  penergy ; decreases/increases when occupied/vacant
]

turtles-own ; variables belonging to (each) forager
[
  move? ; true if moving, false if feeding
  energy ; initially 0 – increases/decreases when feeding/moving
]
```

- **pcolor** *is a built-in patch variable that assigns the patch color*

ABM Structure (**forager** Procedure '**go**')

to go

tick ; *mark time and update visual display*

check-food ; *check patch energy of feeding forager, change* **move?** *to* **true** *if* **penergy**$<$**X**

feed ; *if* **move?**$=$*false, run DE model for energy transfer*

move-turtles ; *if* **move?**$=$*true, move forager*

grow-patch ; *run DE model for patches not being visited*

set-color ; *set* **pcolor** *to indicate energy level*

update-totals ; *set* **mean-energy** *to total energy / time*

end

- The steps repeat until stopped by the user.
- Each bold item is a procedure; we'll look at **feed**.

forager Procedure 'feed'

```
to feed
 ask turtles with [not move?]
 [
  let slopess slopes penergy ; let is for local variables
  let de-patch (item 0 slopess) * dt
  set penergy (penergy + de-patch) ; de-patch<0
  ...
 ]
end
```

- The bracketed code applies to all turtles that are not moving and the patches they are in.
- The incremental patch energy change (**de-patch**) is a nonlinear function of patch energy, calculated by the reporter **slopes**.

forager Reporter 'slopes'

```
to-report slopes [xx] ; uses rk4 to calculate mean slopes
  let kx1 xprime xx ; x-prime is a scalar function for x'
  let ky1 yprime xx
  let kx2 xprime (xx + 0.5 * kx1 * dt)
  ...
  let slope-x (kx1 + 2 * kx2 + 2 * kx3 + kx4 ) / 6
  let slope-y (ky1 + 2 * ky2 + 2 * ky3 + ky4 ) / 6
  report list slope-x slope-y
end
```

Modeling Real Experiments in Virtual Worlds

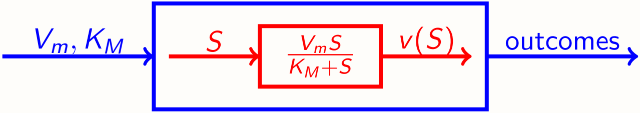
- The best way to learn mathematical modeling is to collect real world data and then build a model that helps us understand the data.
  - Except real world experiments are slow, expensive, and sometimes dangerous. ☹
  - And there are confounding variables that are hard to control. ☹
- A virtual world like **forager** offers a fast, cheap, safe, and controllable opportunity to do real science experiments. ☺
  - For example, is there an optimal choice of **X**?
  - If so, how does that optimal value change if we change system parameters?

Mathematical Models as Nested Functions

How do we (mathematically) view the (Michaelis-Menten) model

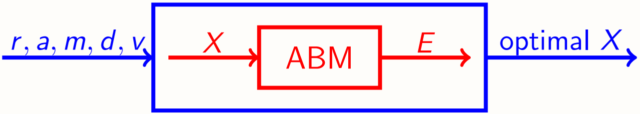
$$v(S) = \frac{V_m S}{K_M + S} \quad ?$$

- **Narrow** view: Function $v(S)$, with parameters V_m and K_M.
- **Broad** view:
  Function that maps V_m and K_M to the graph of $v(S)$.



- **Broad** question: At what S is v half of its maximum?
  - **Modeling questions are in the broad view.**

Optimal Foraging Model



- r is the growth rate of an empty patch.
- a is a parameter in the consumption vs food density model.
- m is the forager metabolism rate while feeding.
- $d > m$ is the forager metabolism rate while moving.
- v is the velocity of the forager while moving.

Studying the **forager** Virtual World

- Three ways to do virtual experiments with **forager**:
  1. Keep running **forager** with different **X** and plot values by hand. 😐
     - Very slow and boring. ☹ ☹ ☹
  2. Use NetLogo's BehaviorSpace facility to automate the experiments. ☺
     - Data is saved to a file and must be analyzed elsewhere. ☹ ☹
  3. Analyze the data within NetLogo! ☺ ☺
     - **There is no obvious way to do that, nor are there examples in the NetLogo library.** ☹ ☹
- **But there IS a way to use NetLogo to collect and analyze data.** ☺ ☺ ☺

Introducing **foraging.nlogo**

- **foraging** uses the same agent-based model as **forager**.
- But with three foragers instead of one.
  - This requires only a minimal change in the code.
- It keeps time without resetting all variables.
  - That means variables can be used to store data from all simulations.
- It plots only the final simulation result, as one point on a graph of **mean-energy** vs **X**.
- It has an **analyze** button that triggers a regression analysis.

foraging Global Variables

```
globals
[
  ;; Variables that are reset for each experiment
  start-ticks ; start tick for current run
  mean-energy ; mean energy per time across turtles
  ...
  ;; Variables that are not reset for each experiment
  runtime ; number of steps in a run
  xlist ; list of X values
  ylist ; list of mean-energy values
  ...
]
```

- Each run goes from **start-ticks** to **start-ticks** + **runtime**.

foraging Procedure 'go'

```
to go
  if ticks>0
  [
   reset-experiment ; reads new X from slider, ...
   populate
  ]
  foreach range runtime [run-one-step]
  output-and-save
end
```

- **run-one-step** is repeated **runtime** times.
- **output-and-save** writes results to **xlist**, **ylist**, and a **monitor window**; it also adds a point to the **plot**.

foraging Procedures 'analyze' and 'get-results'

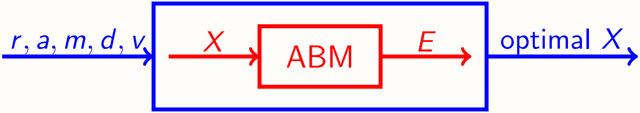
```
to analyze
  get-results ; fits least squares parabola
  plot-parabola ; adds parabola to plot
  write-results
end

to get-results ;; on analyze
  let fit LS-parabola xlist ylist
  set aa item 0 fit
  . . .
end
```

- **LS-parabola** is a **reporter** that fits parabola parameters for a set of **xlist** and **ylist**.

“Full” Automation with foraging_auto.nlogo

```
to go
  if ticks > 0 [set X X + 0.02] ; automatically increment X
  ...
  ifelse (mean-energy < 0 or X = 0.98) ; end condition
  [
    analyze ; if done
    stop
  ]
  [output-and-save] ; if not done
end
```

Full Automation



- r is the growth rate of an empty patch.
- m is the forager metabolism rate while feeding.
- $d > m$ is the forager metabolism rate while moving.

- **foraging_auto.nlogo** only fully automates the finding of optimal X for one set of parameters.
- We could more fully automate the experiment by embedding the optimality routine of **foraging_auto.nlogo** into a program with automated incrementing of a parameter.

Take-Home Messages

- Virtual laboratories give students a way to collect real data, albeit not for a real-world setting.
- NetLogo is a convenient platform for creating virtual laboratories using agent-based models.
- The agent-based models used in NetLogo can include differential equations.
- Experiments and data analysis can be automated in NetLogo if you know how to do it.