CSCU9YM: Modelling for Complex Systems

Lecture 2: Introduction to NetLogo

NetLogo basics

- History: descendent of the Logo language
  - (Any of you used Logo in school? Remember turtles?)
- Domain specific language for programming ABM
- Not a general purpose programming language like Java
- (Though NetLogo is itself implemented in Java.)
- NetLogo's IDE comes with a built in GUI for visualizing ABM
- NetLogo programs (called models) can be run within the GUI or can be run headless (without the GUI)
  - This allows them to be controlled by other programs that run on the JVM (a very useful feature, but outside the scope of this module)
- NetLogo Web makes it easy to export models to HTML5
- The language includes many powerful constructs that make it easy to program ABM,
- but can take some getting used to for Java programmers!

NetLogo basics (2)

- The practicals will introduce you to the NetLogo tool and IDE, and how to build GUIs for your models
- This lecture contains a very brief introduction to some basic concepts of the NetLogo programming language
  - agents and breeds
  - procedures, commands, reporters and contexts
  - managing time
  - agentsets
  - randomness
- As experienced programmers, you will be expected to learn the NetLogo language on your own, using online tutorials and examples.
- NetLogo has many advanced features not covered in this module (e.g. Mathematica link; GIS extension). For full details see the Programming Guide at http://ccl.northwestern.edu/netlogo/docs/

Agents, Breeds, variables

- The basic concept in agent-based programming is the agent. NetLogo has 4 types of agent:
  - patches (immobile, make up the background)
  - turtles (mobile, located on patches)
  - links (used for connecting turtles to build networks)
  - the observer (implicit default agent, unique)
- Turtles and links can be of different breeds. For example:

```
breed [wolves wolf]

breed [sheep a-sheep]
```

- Agents (and breeds) can have their own variables, both built-in and user-defined. For example:

```
patches-own [elevation]

sheep-own [age, energy]

ask turtle 1 [set color red]
```

Commands and Reporters

- NetLogo expressions may be commands or reporters.
- Commands cause an agent to carry out some action.
- Reporters cause an agent to calculate and return some value.
- NetLogo comes with many primitive commands and reporters, listed in the NetLogo Dictionary.
- Programmers can write their own commands and reporters.
- Commands and reporters are executed in a context. The context is the agent which must carry out the command or reporter
- Examples:

```
create-turtles 10   ;; observer command (default)
ask patches [set pcolor green]  ;; observer command, [patch command]
let  peak-height max [elevation] of patches
                 ;; observer command, observer reporter, [patch reporter]
```

User-defined Procedures and Reporters

- Multiple commands can be grouped together to form a named procedure.
- Procedures are somewhat like void methods in Java.
- Most NetLogo models have at least two procedures, commonly called setup and go (but you can give them different names if you wish).
- Procedures are invoked in two ways
  - using their names as commands in other procedures
  - via buttons in the interface tab
- Procedures have implicit contexts. For example, a turtle procedure contains only commands that a turtle can carry out.

- It is also possible to create user-defined reporters. These are somewhat like non-void methods in Java (i.e., ones that return a value).

Time and ticks

- Many models have time passing in discrete steps, representing some appropriate unit of time for the system being modelled.

- NetLogo allows these to be modeled as “ticks”

- There is a built in tick counter.

- `ticks` reporter shows how much time has passed

- `reset-ticks` command restarts the clock

- `tick` command advances the clock to the next step.

Agentsets

- An **agentset** is a set of agents
- The programmer can select and manipulate a set of agents at once.
- The `ask` command can be used to make all the agents in an agentset perform some action.
  - The programmer does not have to explicitly iterate through the agents in the set. The iteration is built into the `ask` command.
- Agents in an agentset are randomly ordered. This means that they are visited in a different order each time the `ask` command is used.
- Examples using agentsets. The agentset is underlined

```
ask turtles [forward 10]     ;; set of all turtles
ask patch 23 45 [set pcolor blue]    ;; set containing one patch
count other turtles-here with [age > 20]    ;; set of turtles
ask neighbors with [elevation < 20] [set pcolor green] ;; set of patches
```

Randomness

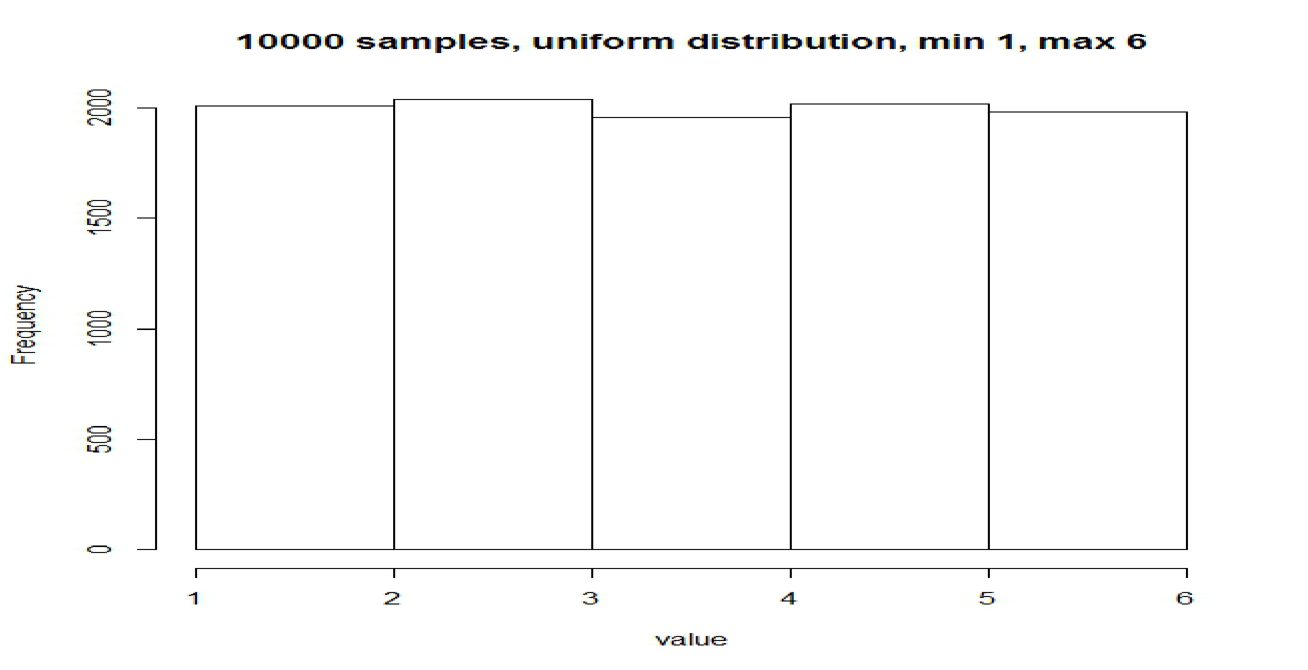
- Many systems modelled by ABM require random behaviour.
- NetLogo has several constructs that generate random behaviour:

- Explicit randomness (random number generator)
  - random 100 ;; a random integer from 0 to 99
  - random-float 100 ;; a random floating point number in [0,100)

- Many NetLogo constructs have some implicit random behaviour
  - ask turtles [action]
    - » all turtles perform action sequentially in some random order
  - n-of 10 wolves
    - » returns an agentset consisting of 10 randomly chosen wolves

Randomness: uniform distribution

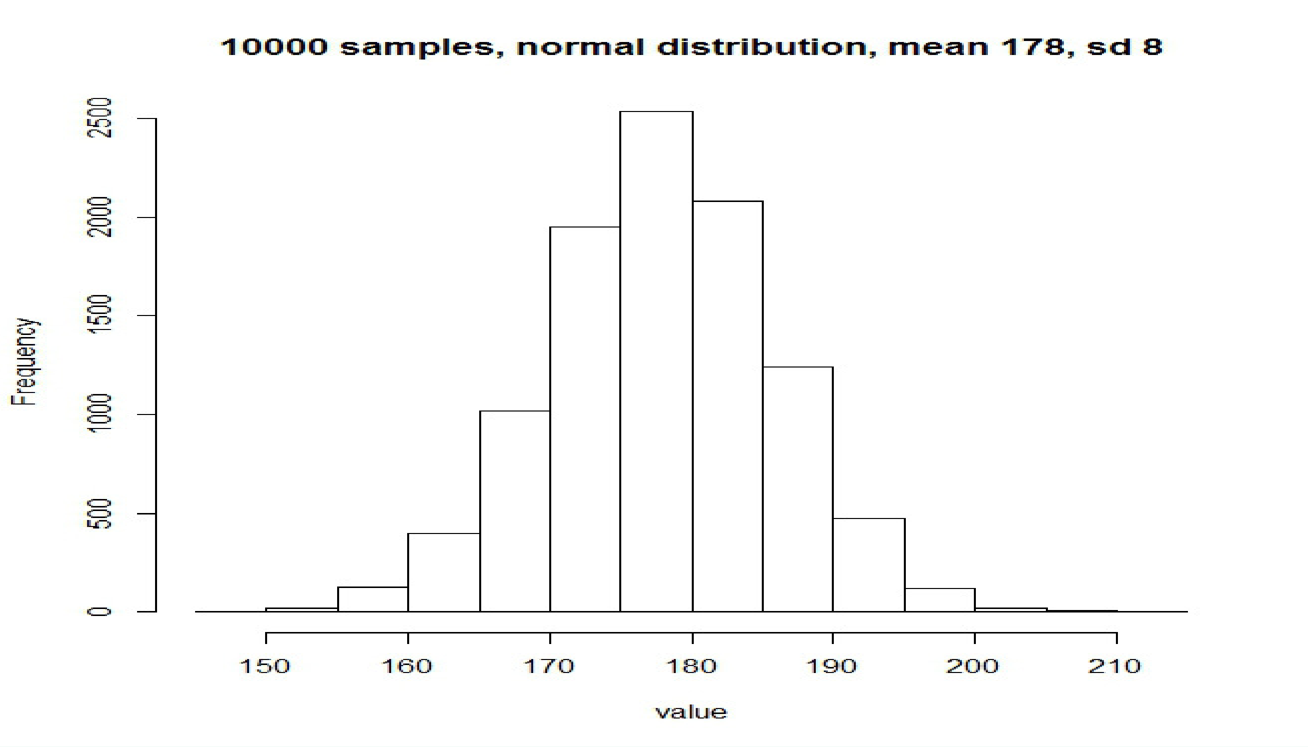
- The random and random-float reporters generate random numbers from a *uniform* distribution.
  - This is useful if we want equal probabilities for all possible numbers
  - For example, we can use this to simulate rolling a fair die
- Most programming languages have a random number generator which generates samples from a uniform distribution.



Randomness: normal distribution

- Sometimes, other probability distributions are more appropriate.
- For example, heights of US adult males follows a normal (or Gaussian) distribution, with mean 178 cm and standard deviation 8 cm (approx).
- In NetLogo, we could use

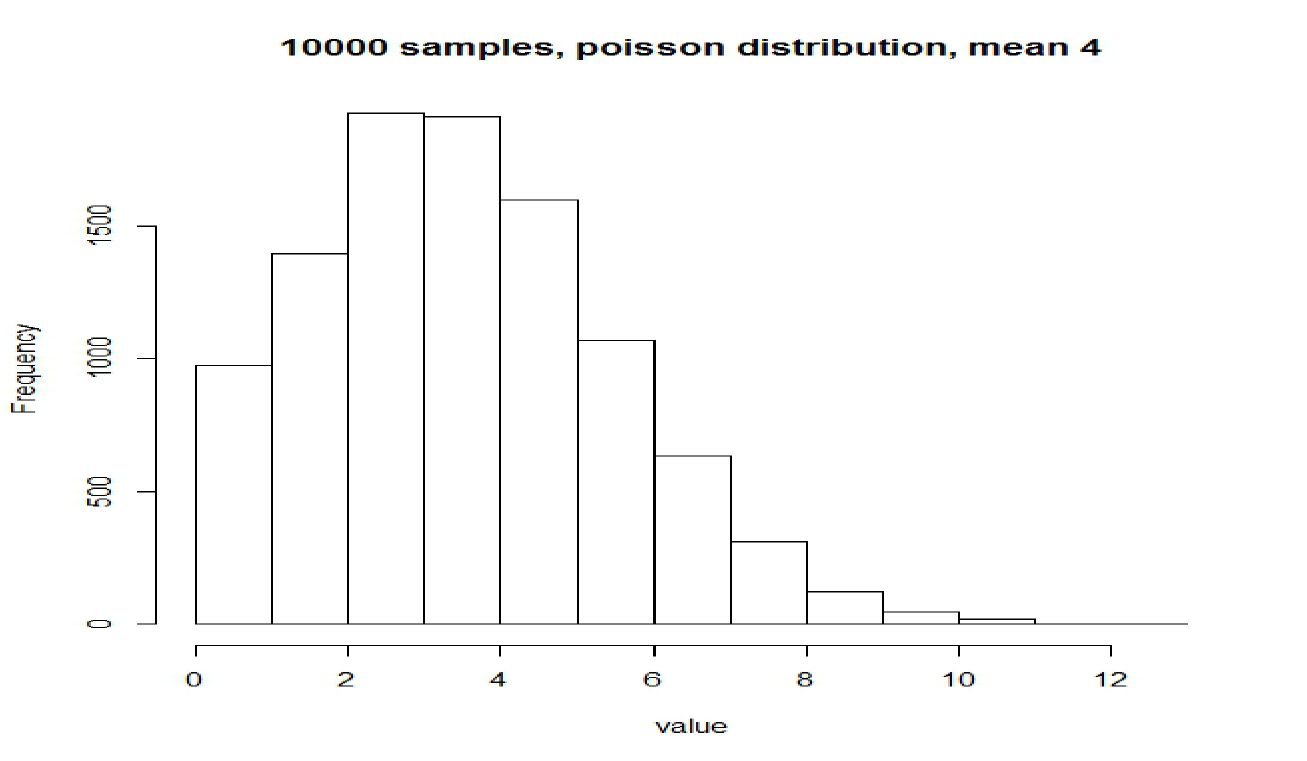
```
ask turtles [set height random-normal 178 8]
```



Randomness: Poisson distribution

- Poisson distributions are useful for modelling events that occur randomly with a known mean rate. For example, we might want to model cars arriving at a junction at a mean rate of 4 cars per minute.
- In NetLogo, we could use

```
let arrivals random-poisson 4
```



End of lecture