Tutorial: Intro to Agent-Based Modeling (ABM) through Modeling Mass Opinion Dynamics

---

Getting Started

Acknowledgements:

This tutorial was prepared for the 2019 SBP-BRiMS conference (http://sbp-brims.org/2019/) based on research done with Dr. William Burns of Decision Research (https://www.decisionresearch.com/).

Materials

- Download and Install Netlogo: https://ccl.northwestern.edu/netlogo/
- Obtain these tutorial files: https://github.com/dalefrakes/netlogo-tutorial
- Paper: *Extremism without extremists: Deffuant model with emotions* (Pawel Sobkowicz) https://www.frontiersin.org/articles/10.3389/fphy.2015.00017/full

---

Writing Your First Models

A Basic Template for Nearly Every Model

A newly started NetLogo model has no buttons nor any code. But in my experience, nearly every model has a `setup` and `go` buttons. Likewise, there are `to setup` and `to go` procedures to go with those buttons. Most models also tend to have some `global`, `patch`, and `turtle` variables.

Instead of creating these items every time I start a model, I like to have a basic template to start from. Such a template might look like this:

And the corresponding code block looks like this.

File: 00-base_template.nlogo:

```
;; basic NetLogo model template
;; Author:  First Last
;; Date:  8 July 2019

globals [
]

patches-own [
]

turtles-own [
]
```

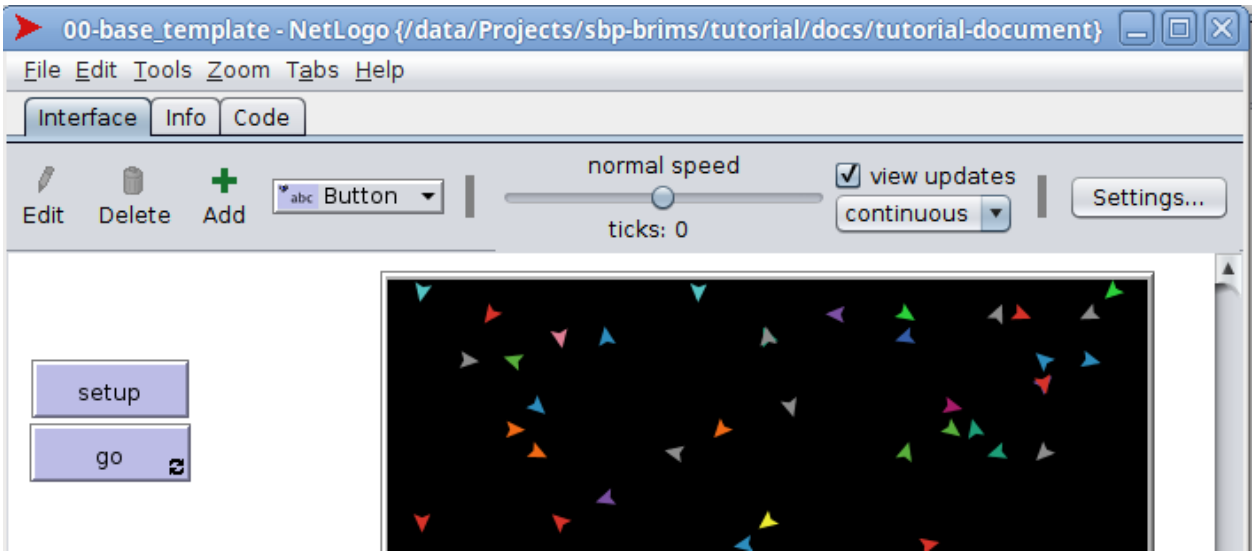


Figure 1:

```
to setup
  clear-all


  reset-ticks

end

to go

  tick

end
```

Note that this NetLogo model does pretty much nothing except clear the environment.

To use this template, open it up and `save-as` using the name you want for your new model.

A Basic Template that *Does Something*

Many models feature a population of turtles that randomly move around. So this slightly more advanced template adds a slider to provide the count of the population, along with code in the `setup` procedure to create that population (and randomly place them in the world). It also adds a new `wiggle` procedure that causes the turtles to randomly turn a bit, along with code in the `go` procedure to ask the turtles to `wiggle` then move forward.

Things to note:

- sliders show up as `global` variables, but they're defined in the interace and not in the code itself)
- the `wiggle` procedure was borrowed from the `Ants` model that can be found in the Model Library - the Model Library is an excellent source of ways to solve a huge variety of problems
- `wiggle` works by having the turtles turn to the right at a random angle up to 40 degrees, then to the left at another random angle up to 40 degrees.

And the corresponding code block looks like this.

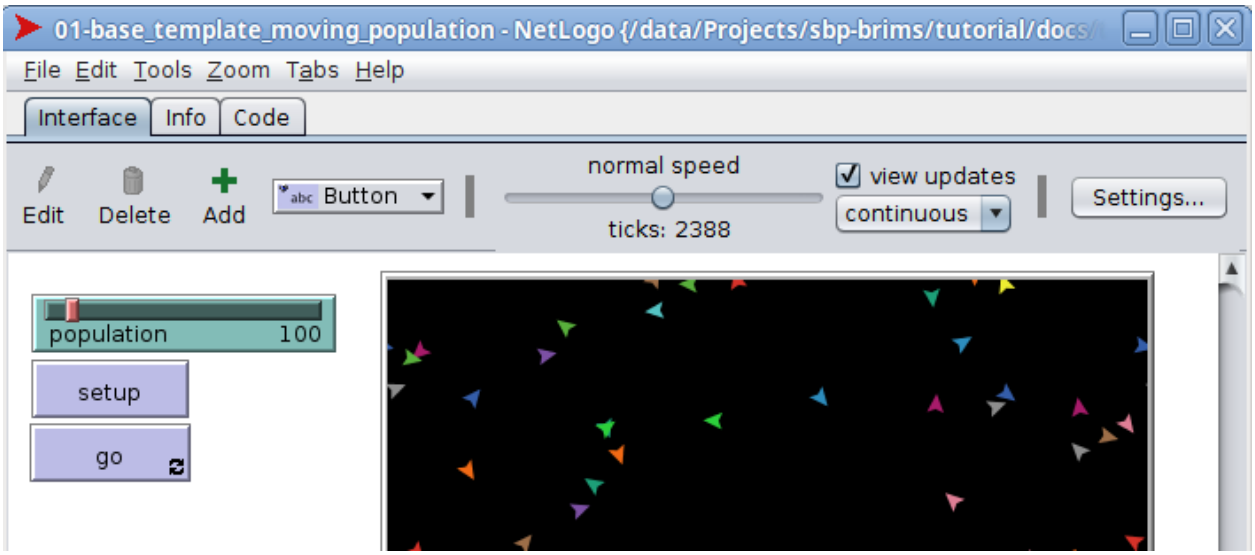


Figure 2:

File: 01-base_template_moving_population.nlogo:

```
;; basic NetLogo model template, with randomly moving population
;; Author:  First Last
;; Date:  8 July 2019

globals [
]

patches-own [
]

turtles-own [
]

to setup
  clear-all

  create-turtles population [
    setxy random-pxcor random-pycor
  ]

  reset-ticks

end

to go

  ask turtles [
    wiggle
    forward 1
  ]

  tick
```

```
end

to wiggle  ;; turtle procedure - borrowed from ants model
  right random 40
  left random 40
  if not can-move? 1 [ rt 180 ]
end
```

A First Population Opinion Model

For our first model that exhibits some Population Opinion Dynamics, we'll start by considering some of the models described in a paper by Pawel Sobkowicz of the National Centre for Nuclear Research, Warsaw, Poland, "Extremism without extremists: Deffuant model with emotions", published in Frontiers in Physics in 2015.

(https://www.frontiersin.org/articles/10.3389/fphy.2015.00017/full)

Opinions, Thresholds, and Exchanges

Things to Consider (Leading to Variables!)

From the description of the model in the paper, we can start to think about what might be needed to implement a NetLogo model.

First let's consider the agents:

- there's only one type of turtle (e.g. a person), so there's no need to make new `breeds`
- a person has an opinion ($person_i$ has an opinion o_i)

Next, let's think about global variables. The paper mentions a threshold d and an convergence rate u. Since both of these can take on a range of values set by the model's user, it makes sense to make these into sliders.

It also might be helpful to define some hard limits like the bounds of opinion (-1 to 1) as global variables that are managed strictly within the code.

And while the agents wil move around and encounter one another randomly, there's no information needed about where they are, so we won't need any patch variables.

Setting up these variables, we have something that looks like this:

And the corresponding code block looks like this.

File: 03-simple-global-threshold.nlogo:

```
;; basic population opinion dynamics with global threshold
;; Author:  First Last
;; Date:  8 July 2019

globals [
  o-max   ;; maximum possible opinion
  o-min   ;; minimum possible opinion
]

turtles-own [
  opinion  ;; opinion on spectrum from -1 to 1
```

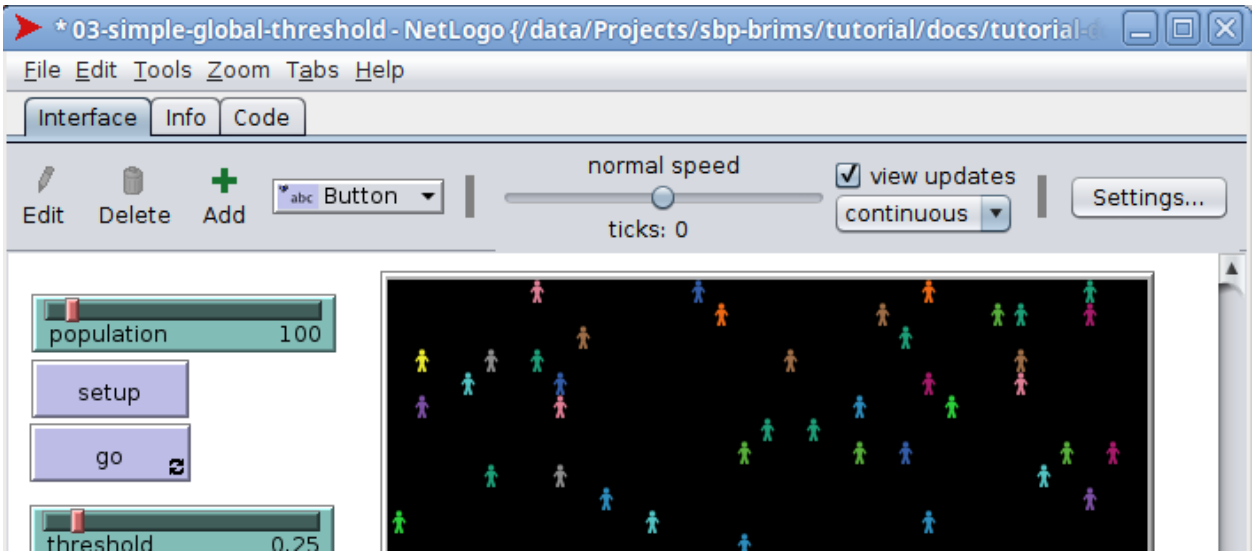


Figure 3:

```
]

to setup
  clear-all

  ;; set global variables
  set o-max 1
  set o-min -1

  create-turtles population [
    set shape "person"
    setxy random-pxcor random-pycor

    ;; set turtle's opinion as uniform-random between opinion bounds
    set opinion random-float (o-max - o-min) + o-min
    set-turtle-color
  ]

  reset-ticks

end

to go

  ask turtles [
    wiggle
    forward 1
  ]

  tick

end
```

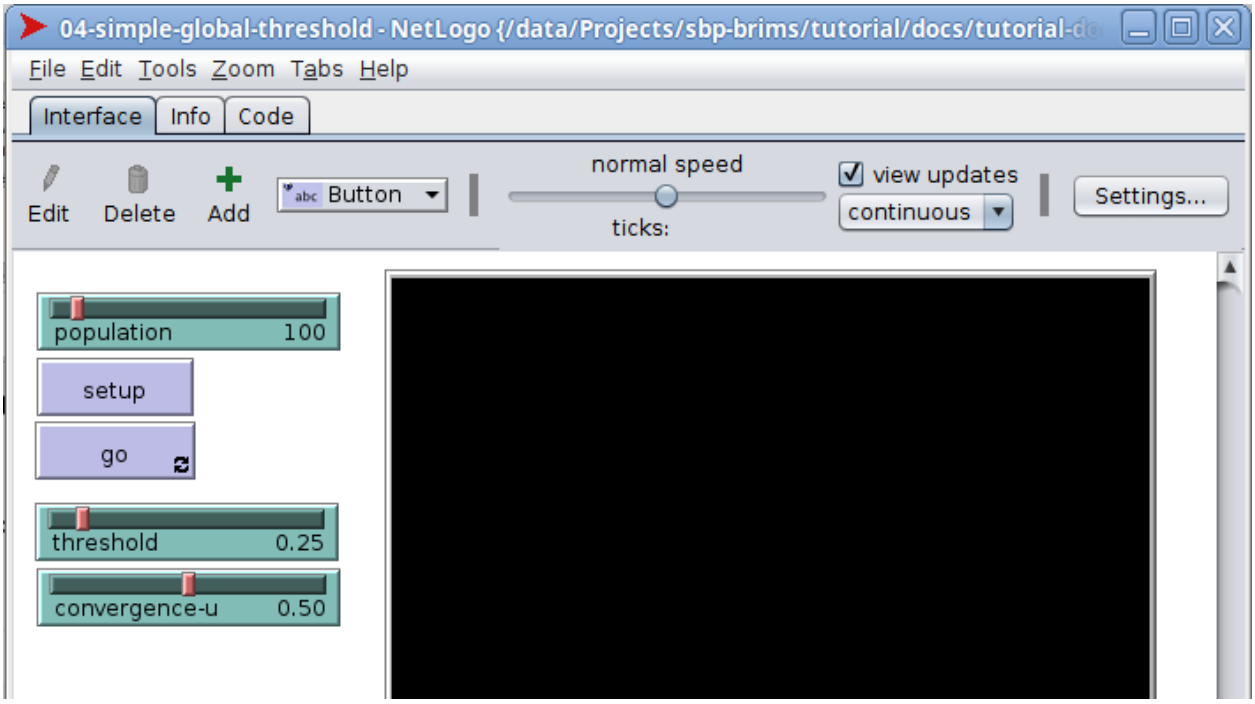


Figure 4:

```
to wiggle  ;; turtle procedure - borrowed from ants model
  right random 40
  left random 40
  if not can-move? 1 [ rt 180 ]
end

to set-turtle-color  ;; turtle procedure
  ifelse opinion > 0
  [ set color scale-color red opinion (o-max * 1.5)  0 ]
  [ set color scale-color blue (0 - opinion) (o-max * 1.5) 0 ]

end
```

Note:

- the opinions of turtles are set randomly, drawing from a uniform distribution (`random-float`)
- the shape of the turtles were changed to `people`
- we've added a turtle procedure to set their color based on the sign and intensity of their opinions

Now the Interactions

$$opinion_i(t+1) = opinion_i(t) + u(opinion_j(t) - opinion_i(t))$$
$$opinion_j(t+1) = opinion_j(t) + u(opinion_i(t) - opinion_j(t))$$

And the corresponding code block looks like this.

File: 04-simple-global-threshold.nlogo:

```
;; basic population opinion dynamics with global threshold
```

```
;; Author:  First Last
;; Date:  8 July 2019

globals [
  o-max   ;; maximum possible opinion
  o-min   ;; minimum possible opinion
]

turtles-own [
  opinion  ;; opinion on spectrum from -1 to 1
]

to setup
  clear-all

  ;; set global variables
  set o-max 1
  set o-min -1

  create-turtles population [
    set shape "person"
    setxy random-pxcor random-pycor

    ;; set turtle's opinion as uniform-random between opinion bounds
    set opinion random-float (o-max - o-min) + o-min
    set-turtle-color
  ]

  reset-ticks

end

to go

  ask turtles [

    ; if there is another turtle here, interact and adjust opinion
    if (any? other turtles-here)
    [
      ;; this is an interaction between two agents
      let other-turtle one-of turtles-here
      let oi [opinion] of self
      let oj [opinion] of other-turtle
      if (abs (oi - oj) <= threshold)
      [
        set opinion oi + convergence-u * (oj - oi)
        ask other-turtle [set  opinion oj + convergence-u * (oi - oj)]
      ]

      set-turtle-color

    ]

    wiggle
```

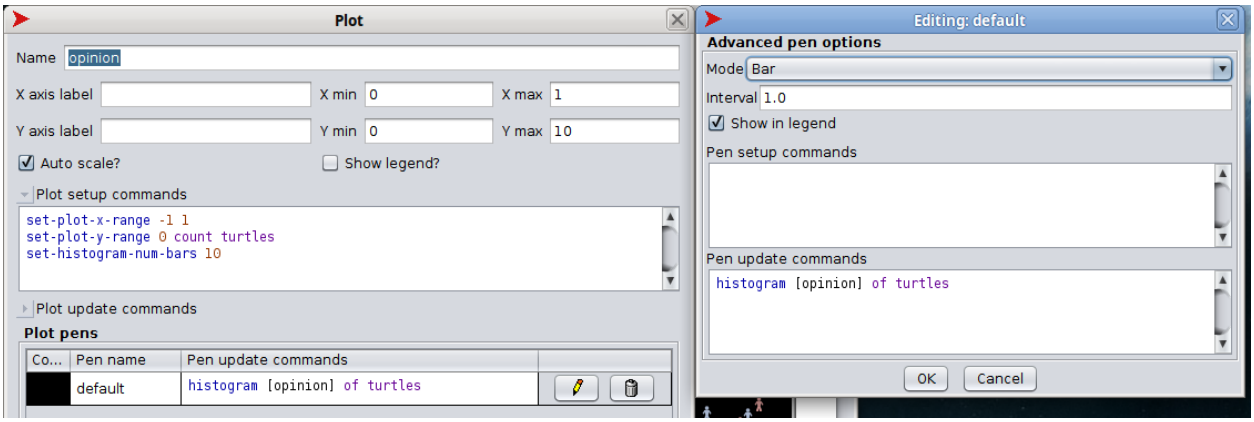


Figure 5:

```
    forward 1
  ]

  tick

end

to wiggle  ;; turtle procedure - borrowed from ants model
  right random 40
  left random 40
  if not can-move? 1 [ rt 180 ]
end

to set-turtle-color  ;; turtle procedure
  ifelse opinion > 0
  [ set color scale-color red opinion (o-max * 1.5)  0 ]
  [ set color scale-color blue (0 - opinion) (o-max * 1.5) 0 ]

end
```

Adding a Histogram

It's more useful to see a histogram of opinions.

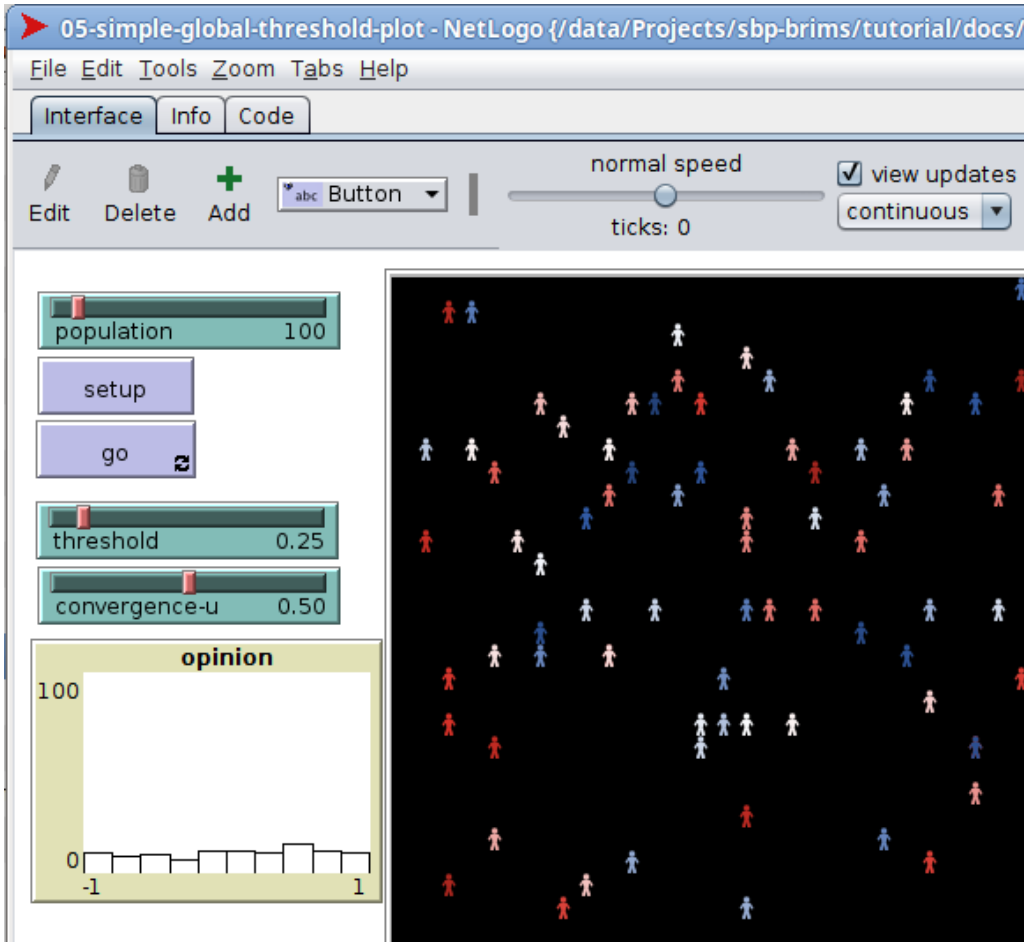
The code pieces there look like this:

Plot Setup Commands:

```
set-plot-x-range -1 1
set-plot-y-range 0 count turtles
set-histogram-num-bars 10
```

Pen Update commands:

```
histogram [opinion] of turtles
```



Which will give us somthing like this:

05-simple-global-threshold-plot-b.nlogo

Different Probability Distributions

The initial population's opinion might not best be described by a uniform distribution. Next we'll look at how to explore using different distributions.

And the corresponding code block looks like this.

File: 06-with_probability_distributions.nlogo:

```
;; basic population opinion dynamics with global threshold
;; Author:  First Last
;; Date:  8 July 2019

globals [
  o-max   ;; maximum possible opinion
  o-min   ;; minimum possible opinion
]

turtles-own [
  opinion  ;; opinion on spectrum from -1 to 1
]

to setup
```

```
  clear-all

  ;; set global variables
  set o-max 1
  set o-min -1

  create-turtles population [
    set shape "person"
    setxy random-pxcor random-pycor

    ;; set turtle's opinion as uniform-random between opinion bounds
    set opinion random-float (o-max - o-min) + o-min

    ;; find the mean value/"location" of the distribution
    let dist-location (o-min + (o-max - o-min) / 2)

    ;; now randomly set the opinions based on chosen distribution
    ifelse  initial-opinion-dist = "uniform" [ set opinion random-float (o-max - o-min) + o-min ]
    [ifelse initial-opinion-dist = "normal"  [
      set opinion random-normal-bounded dist-location dist-scale o-min o-max]
      [ifelse initial-opinion-dist = "laplace" [
        set opinion ( random-laplace-bounded dist-location dist-scale o-min o-max ) ]
        [set opinion 0  ;; this option shouldn't happen
    ]]]

    set-turtle-color
  ]

  reset-ticks

end

to go

  ask turtles [

    ; if there is another turtle here, interact and adjust opinion
    if (any? other turtles-here)
    [
      ;; this is an interaction between two agents
      let other-turtle one-of turtles-here
      let oi [opinion] of self
      let oj [opinion] of other-turtle
      if (abs (oi - oj) <= threshold)
      [
        set opinion oi + convergence-u * (oj - oi)
        ask other-turtle [set  opinion oj + convergence-u * (oi - oj)]
      ]

      set-turtle-color

    ]

    wiggle
```

```
    forward 1
  ]

  tick

end

to wiggle  ;; turtle procedure - borrowed from ants model
  right random 40
  left random 40
  if not can-move? 1 [ rt 180 ]
end

to set-turtle-color  ;; turtle procedure
  ifelse opinion > 0
  [ set color scale-color red opinion (o-max * 1.5)  0 ]
  [ set color scale-color blue (0 - opinion) (o-max * 1.5) 0 ]

end

to-report random-normal-bounded [#u #sd #min #max]

  let temp-value random-normal #u #sd
  while [(temp-value < #min) or (temp-value > #max)]
  [
    set temp-value random-normal #u #sd
  ]
  report temp-value

end

to-report random-laplace-bounded [#u #b #min #max]

  let temp-value random-laplace #u #b
  while [(temp-value < #min) or (temp-value > #max)]
  [
    set temp-value random-laplace #u #b
  ]
  report temp-value

end

to-report random-laplace [ #u #b ]
  ; draw U (unif) from uniform (1/2 to 1/2]
  ; X = u - b * sign(U)*ln(1 - 2*abs(U))

  let unif -0.5 +  random-float 1
  let signU 1
  if unif < 0
  [set signU -1]

  report #u - #b * signU * ln( 1 - 2 * abs(unif))

end
```

Continuing in Ad-Lib Mode

We'll continue the rest of this tutorial in an ad-lib mode.