Learning Simulator
User's Guide

Markus Jonsson

October 7, 2018

1 Installation instructions

2 How to run the program

Use the control command `lesim` to run Learning Simulator. Below are the available options.

Listing 1: `lesim` syntax

```
python lesim.py
Short for "python lesim.py gui"

python lesim.py gui
Start the Learning Simulator gui

python lesim.py run file1 [file2, file3, ...]
Run the script files file1, file2, ...

python lesim.py help
Display this help and exit
```

3 Learning models

This section will introduce the learning models that can be simulated in the program. Figure 1 illustrates the system we want to study. The *organism* has an output function that generates behavior and state transition functions that update memories and other internal states. The world, which often is defined by an experiment, has an output function that generates stimuli and state transitions functions that update the state of the world, Both are

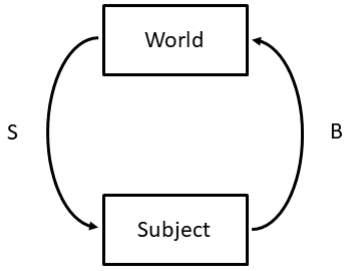

Figure 1: The world and the subject.

partly influenced by the behavior of the organism. Stimuli may not be fully informative about the state of the world.

A complete description of the whole system requires output functions and state transition functions for both the organism and the world including specifications of *behavior* and *stimulus repertoires*

The system will operate in *discrete time steps* and we need to specify a *time scale*. This scale could be different in different applications of the program and could be small. A natural time scale is the rate whereby the organism can respond or make decisions. An alternative is to alternate stimuli and responses: $S_1 \rightarrow B_1 \rightarrow S_2 \rightarrow B_2 \rightarrow S_3 \rightarrow ....$

On top of this is the *experimental structure* in terms of trials and, training and test phases.

3.1 Stimuli and behavior

Stimuli may consist of combinations of *stimulus elements* (i.e. compound stimuli) and these elements must also be specified. There may also be variation in *stimulus intensity* (develop now or later?). Appendix 3 in Enquist et al. describes how combinations of elements and stimulus intensities can operate together.

3.1.1 Notation

Table 1 shows the notation used.

3.2 The organism

The organism makes decisions about behavior B and learns from observations S. In dynamical systems terms decision making is an an output

Symbol	Description
E	A stimulus element
S	A stimulus which consists of one or more stimulus elements $\{E_1, E_2, \ldots, E_k\}$. The stimulus repertoire is a set of stimuli $\{S_1, S_2, \ldots, S_n\}$
I	Intensity?
B	A behavior. The behavior repertoire is a set of behaviors $\{B_1, B_2, \ldots, B_m\}$.

Table 1: The notation for stimuli and behavior.

function and state transition functions updates memories and other state variables in the organism.

3.2.1 Notation

Table 2 shows the notation used.

Symbol	Description
$v_{S \to B}$	Learned value of choosing B in response to S (has inborn start value)
u_S	The inborn (primary, initial) value of S
w_S	The learned contribution/modification to the value of S (initial value 0)
$u_S + w_S$	The value of choosing B in response to S
r_S	The value of choosing B in response to S without a u and w division
$\alpha, \alpha_v, \alpha_w$	Learning rates.

Table 2: The notation for stimuli and behavior.

Traditionally v is referred to a stimulus-response associations. We sometimes refer to stimulus-response value because v can be interpreted as an estimate of the value of responding with B towards S.

3.2.2 The output function (generates stimuli)

Stimulus representation: In all mechanisms developed so far, responding is only based on the current stimulus. Possible developments that would change this include the introductions of **stimulus traces** and **stimulus**

sequences. (we leave this for the future). For decision making (output function) we for now, use a version of the soft max rule:

$$\Pr(S \to B_i) = \frac{\text{Support}(B_i)}{\sum_j \text{Support}(B_j)} = \frac{\exp(\beta v_{S\to B_i})}{\sum_j \exp(\beta v_{S\to B_j})}$$

where β regulates the amount exploration or degree of variation in responding (lower β more exploration). If $S = (E_1, E_2, \dots)$ is a compound of stimulus elements the expression changes to

$$\Pr(S \to B_i) = \frac{\exp(\sum_k \beta v_{E_k\to B_i})}{\sum_j \exp(\sum_k \beta v_{E_k\to B_j})}$$

Possible developments (to be implemented later) include making the value of β dependent on E and/or B.

This would introduce genetic predispositions that could guide exploration in profitable directions. Other developments include adding internal states such a clocks and regulatory states.

3.2.3 State transition functions (memory updates etc.)

The learning described in the table occurs after observing

$$S \to B \to S'.$$

3.2.4 Adding cost to behavior

In some cases it is important to add a cost to certain responses. This can be done in the following way in our model by replacing

$$\Delta v_{S\to B} = \alpha_v(u_{S'} + w_{S'} - v_{S\to B})$$

with

$$\Delta v_{S\to B} = \alpha_v(u_{S'} + w_{S'} - c_B - v_{S\to B})$$

[1] Bush and Mosteller [1951], Rescorla & Wagner [1972]
[2] Watkins [1989], Watkins & Dayan [1992]
[3] Rummery & Niranjan [1994]
[4] van Seijen & van Hasselt & Whiteson & Wiering [2009]
[5] Witten [1977], Barto et al. [1983]
[6] Enquist et al. [2016]

Mechanism	Memory states	Memory updates
Stimulus-Response Learning[1]	$v_{S\to B}$	$\Delta v_{S\to B} = \alpha(r_{S'} - v_{S\to B})$
Q-Learning[2]	$v_{S\to B}$	$\Delta v_{S\to B} = \alpha(r_{S'} + \max_i v_{S'\to B'_i} - v_{S\to B})$
SARSA[3]	$v_{S\to B}$	$\Delta v_{S\to B} = \alpha(r_{S'} + v_{S'\to B'} - v_{S\to B})$ *
Expected SARSA[4]	$v_{S\to B}$	$E(v_{S'\to B'}) = \sum_i \Pr(S' \to B'_i) v_{S'\to B'_i}$ $\Delta v_{S\to B} = \alpha(r_{S'} + E(v_{S'\to B'}) - v_{S\to B})$
Actor-critic[5]	$v_{S\to B}$, w_S	$\delta = u_{S'} + w_{S'} - w_S$ $\Delta v_{S\to B} = \alpha_v \delta$ $\Delta w_S = \alpha_w \delta$
Our model[6]	$v_{S\to B}$, w_S	$\Delta v_{S\to B} = \alpha_v(u_{S'} + w_{S'} - v_{S\to B})$ $\Delta w_S = \alpha_w(u_{S'} + w_{S'} - c_B - w_S)$

* In SARSA, the updating is done after observing $S \to B \to S' \to B'$.

Table 3: Learning mechanisms and their memory updates.

where c_B is the cost of B. I guess one should also change the updating of w and v to

$$\Delta w_S = \alpha_w(u_{S'} + w_{S'} - c_B - w_S)$$

$$\Delta v_{S\to B} = \alpha_{(v)}(\ldots - c_B)$$

Such cost can also be introduced into the other models.

3.3 The world

The world receives behavior from the organism and responds with stimuli. A description of a world specifies how stimuli are generated and how state variables are updated.

3.3.1 Pavlovian world

3.3.2 Linear world

3.3.3 Social learning world

4 The scripting language

The input to Learning Simulator is a script. It is specified as plain text in the main window. If a script is stored in a text-file, this file can be opened

into the main window using the **File**-menu. It is also possible to run the script in a text-file using the command line syntax

```
python lesim.py run file1 [file2, file3, ...]
```

as shown in Listing 1.

4.1 Lists

In several parts of the script, a list of items should be specified. What we mean by a list is a list of items separated by space, comma, tab, or new line. For example, a list of the stimulus elements $S1$, $S2$, $S3$, and $S4$ may look like

```
S1, S2, S3, S4
```

or

```
S1 S2, S3
S4
```

4.2 The script sections

A Learning Simulator script consists of a number of sections. Each section starts with a keyword and each keyword starts with `@`. The keywords are `@VARIABLES`, `@PHASE`, `@RUN`, and `@FIGURE`.

Below follows the description of each of these sections.

4.3 Comments

On each line in the script, any text to the right of a hash-character `#` is ignored, and may therefore be used to make your own comments to the script. For those of you familiar with the Python programming language, this corresponds to code comments. All leading and trailing spaces and tabs on each line are also ignored.

To make multi-line comments, use two triple-hashes `###` on separate lines, one before the multi-line comment, and one after.

Listing 2: Script comments

```
@VARIABLES
stop_cond: 10  # The number of repeats
var1: 42       # Variable 1
# Here is another variable
var2: 33

###
This is a way to write
```

```
a multiline comment.
The three hashes must come in pairs.
###

# Of course, one can also do
# this for multiline comments.
```

4.4 @VARIABLES

The `@VARIABLES` section sets the numeric values of variables that can be used throughout the script. The variables are specified as a list of `name:value` after the `@VARIABLES` statement. Thus, the line has the form

```
@VARIABLES name1:value1, name2:value2, ...
```

The variable names must start with a letter, followed by letters, digits, or underscores. The value `value` must be interpretable as a number.

Listing 3: @VARIABLES example

```
@VARIABLES var1:1.2  var2:3.4  var_3:567
```

4.5 Parameters

The *parameters* to a simulation specifies which learning algorithm to use, how many subjects to simulate, the behavior repertoire of the subject, available stimulus elements in the world representation, etc. Each parameter specification has the form

```
parameter_name: parameter_value
```

and can be placed on a new line anywhere in the script. However, for readability, it may be a good idea to put them all together in a group in the beginning of the script. However, it is possible to change the value of any parameter anywhere in the script. The valid parameters and values can be found in Table 4. Parameters without a default value are required.

An example can be found in Listing 4.

Listing 4: An example of setting parameters

```
# Parameters
subjects          : 10
mechanism         : GA
behaviors         : R0 R1 R2
```

```
stimulus_elements : S1 S2 reward
start_v           : -1
alpha_v           : 0.1
alpha_w           : 0.1
beta              : 1
behavior_cost     : R1:1, R2:1, default:0
u                 : reward:10, default:0
```

4.5.1 Stimulus elements and the behavior repertoire

The behavior repertoire `behaviors` as well as the stimulus elements `stimulus_elements` are specified as a list of names. These names must be valid variable names (see Section 4.4), but they cannot be parameter names, e.g., `u`, `beta`, or `bind`. See listing 5 for an example that also uses comments.

Listing 5: `behaviors` and `stimulus_elements`

```
behaviors: escape, stay
stimulus_elements:
   snake   # Subject sees a snake
   neutral # A neutral stimulus
   warning # Warning sound
   bitten  # Subject being bitten
```

4.5.2 Initial v-values

The initial value for $v_{S,B}$ is specified as `S->B:val1` where `S` is a stimulus element, and `B` is a behavior. To specify several different values, use a list:

```
start_v: S1->B1:val1, S2->B2:val2, ...
```

The v-value for all remaining (S, B)-pairs is specified using the keyword `default`:

```
start_v: S1->B1:val1, S2->B2:val2, ..., default:default_value
```

This will set the v-values of each specified pair to the specified value, and all others to `default_value`. If *all* initial v-values are the same, for example 1, use the form

```
start_v: default:1
```

or, shorter

```
start_v: 1
```

Listing 6: `start_v`

```
start_v: snake->escape:-1, warning->escape:-1, default:0.5
```

4.5.3 Response requirements

Use the parameter `response_requirements` if not all behaviors are possible responses to each stimulus element. For example, if the behavior `B` is a possible response to only a subset `1, S2, ...S1, S2, ...` of the stimulus elements, this is specified as

```
B: S1, S2, S3, ...
```

Each behavior that is restricted to a subset of the stimulus elements is specified as a list of items of the above type, so for several behaviors it looks like this:

```
B1: S11, S12, S13, ...
B2: S21, S22, S23, ...
...
```

The behaviors in `behaviors` that are not included in this list are assumed to be possible responses to all stimulus element in `stimulus_elements`.

See listing 7 for an example.

Listing 7: `response_requirements`

```
response_requirements: escape: snake, warning
                       stay:   snake, warning
                       0:      neutral, new_trial, end, bitten
```

4.5.4 Bind learning between trials

When a phase (see Section 5) uses trials, use the parameter `bind_trials` to control whether or not to update the v- and w-values also when reaching the first stimulus in a trial. In other words, in the situation

$$S \to B \to S',$$

where S' is the first stimulus in a trial, `bind_trials` controls whether S' should affect the updating of $v_{S\to B}$ and w_S.

In most cases, this updating should not be done, which corresponds to `bind_trials: off`, which is the default.

5 @PHASE

A world, which presents stimuli to the subject (where a stimulus may or may not depend on the subject's response to the previous stimulus), consists of one or more *phases*. The `@PHASE` section in a script specifies which stimuli are presented, in which order and how they depend on responses. Each phase also has a *phase label* and a *stop condition*.

A `@PHASE` section consists of a number pf *phase lines*. Each phase line consists of a label, an optional stimulus and a logical part. The latter specifies the subsequent stimulus (through a phase line label) and how it depends on the subject's response (if it does). The phase starts at the phase line called `@start_trial`, if there is one (see Section 5.1). Otherwise the phase starts at the first (topmost) phase line. The basic syntax of a `@PHASE` section is as follows:

Listing 8: The basic syntax of a `@PHASE` section

```
@PHASE phase_label
stop:stop_condition
lbl1   stimulus1   |   logic1
lbl2   stimulus2   |   logic2
lbl3   stimulus3   |   logic3
...
```

The label (`phase_label` in the above listing) should be a string (without separators such as commas and spaces) that provides a means of specifying which phases to include in a simulation (see the `@RUN` statement in Section 6). The stop condition (`stop_condition` in the above listing) has the form str=N where str is an event (a stimulus element, a response, or a phase line label), and N is a positive integer. When the specified event has occured N times, this condition is fulfilled and the phase ends. If there is a phase following the ended phase, the first line of that phase will be the current one. Otherwise it ends the simulation. For example, `stop: reward=20` ends the phase after 20 exposures to the stimulus `reward`. When a phase uses trials, the stop condition is typically of the type `@start_trial=N`.

Each stimulus (`stimulus1`, `stimulus2`, ... in the above listing) is specified either as a single stimulus element or a compund stimulus consisting of a number of simultaneous stimulus elements. To specify a compond stimulus, separate the elements with comma, for example `E1,E2`.

Each logic part (`logic1`, `logic2`, ... in the above listing) consists of one or more cases, separated by |. Each case must have the form

```
condition: goto
```

or simply

`goto`

The format of `condition` and `goto` can be found in Tables 5 and Tables 6, respectively.

When using multiple cases, separated by |, this is interpreted as an if-else statement. For example, the interpretation of the logic part

`R1:row1 | R2:row2 | row3`

can be found in Listing 9.

Listing 9: Interpretation of `R1:row1 | R2:row2 | row3`

```
if the response was R1:
    go to row1
else if the response was R2:
    go to row2
else:
    go to row3
```

Example 1 The phase line

`L1 s1 | L2`

exposes the subject to the stimulus element `s1`, then proceeds to the phase line with label `L2`.

Example 2 The phase line

`L1 s1 | countrow(5):L2(0.2),L3(0.8) | L1`

exposes the subject to the stimulus element `s1` five times, then proceeds to the phase line with label `L2` with probability 20% and to `L3` with probability 80%.

Note The three logical parts in Listing 10 are equivalent.

Listing 10: Three equivalent logical parts

```
row1(1/3),row2(1/3),row3(1/3)
row1(1/3) | row2(1/2),row3(1/2)
row1(1/3) | row2(1/2) | row3
```

5.1 Trials

It is common to divide a phase into repeated *trials*. A trial is a sequence of stimulus-response pairs representing ...

To use trials in a phase, label the phase line that starts the trial with `@new_phase`. The phase will then start at this phase line. The parameter `bind_phases` controls whether the u- and w-values will be updated between phases.

Example...

5.2 Counting events with `count` and `countrow`

During the course of a phase, all events (phase labels, stimulus elements, and behaviors) are counted. You can access these numbers using the `count` and `countrow` keywords. These may be used in the logic part of a phase line, in a condition for the subsequent phase line.

The function `count` counts the number of occurrences of an event since the beginning of the phase. You can reset this counter within a phase using the `count_reset` keyword. The `count_reset` keyword can only be used in a non-stimulus phase line, to the left of the `|` character.

The function `count_row` counts the number of occurrences of an event since the current phase line was entered, and is automatically reset when leaving the phase line. This counter cannot be manually reset.

The functions `count` and `countrow` can be used in conditions in the logic part of a phase line:

```
ROW1 S1 | count(ROW1)=5:ROW2 | ROW1
```

which repeats the stimulus `S1` five times, then proceeds to row `ROW2`.

Note that variables (see Section 4.4) may be used:

```
ROW1 | S1 | count(S1)=var1:ROW2 | ROW1
```

5.3 Generating a random integer

The function `random` can be used to set a variable (see Section 4.4) to a random integer. The syntax is

```
var = random(x,y)
```

which generates a random integer from `x` to `y` (including `x` and `y`) with equal probability, and sets `var` to that value. For example,

```
var1 = random(1,3)
var2 = random(10,11)
```

sets `var1` to 1, 2 or 3, each with probability 1/3, and `var2` to 10 or 11, each with probability 1/2.

5.4 Help lines

A phase line does not have to contain a stimulus. You may use a *help line* in a phase description to handle more complicated conditions than what is available using the if-elseif-else interpretation of repeated | (see Listing 9). The logical `OR`

```
if response was A OR B:
    go to ROW2
else:
    go to ROW1
```

can be accomplished without a help line:

```
ROW1    S1 | A:ROW2 | B:ROW2 | ROW1
```

However, to accomplish the logical `AND`

```
if response was A AND count(S1)=5:
    go to ROW2
else:
    go to ROW1
```

the following construction with the help line `A_TRUE` can be used:

```
ROW1    S1 | A:A_TRUE          | ROW1
A_TRUE     | count(S1)=5:ROW2 | ROW1
```

A help line must also be used when resetting a counter using `count_reset`, which cannot be done on a regular phase line. For example,

```
ROW1 count_reset(A) | ROW2
```

A help line must also be used when setting a variable defined in the `@VARIABLES` section. This cannot be done on a regular phase line. For example,

```
ROW1 count_reset(A) | ROW2
```

5.5 Phase examples

A few examples of `@phase` sections can be found in Listings 11 to 18.

Listing 11: Three `@phase` sections for classical conditioning

```
@phase {'labels':'pretraining', 'end':'reward=25'}
CONTEXT 'context'             | 25:US       | CONTEXT
US      ('US','context')      | 'R': REWARD | CONTEXT
REWARD  ('reward','context') | CONTEXT

@phase {'label':'conditioning', 'end':'CS=25'}
CONTEXT 'context'             | 25:CS       | CONTEXT
CS      ('CS','context')      | US
US      ('US','context')      | 'R': REWARD | CONTEXT
REWARD  ('reward','context') | CONTEXT

@phase {'label':'test', 'end':'CS=25'}
CONTEXT 'context'       | 25:CS   | CONTEXT
CS      'CS','context' | CONTEXT
```

Listing 12: A `@phase` section for fixed interval

```
@phase {'label':'fixed_interval', 'end':'reward=25'}
OFF    'lever'  | 4:ON         | OFF
ON     'lever'  | 'R': REWARD | ON
REWARD 'reward' | OFF
```

Listing 13: A `@phase` section for fixed ratio

```
@phase {'label':'fixed_ratio', 'end':'reward=25'}
OFF 'lever'      | 'R'=4: ON   | OFF
ON 'lever'       | 'R': REWARD | ON
REWARD 'reward' | OFF
```

Listing 14: A `@phase` section using a probability schedule

```
@phase {'label':'prob_schedule', 'end': 'reward=25'}
LEVER  'lever'  | 'R': REWARD(0.2) | LEVER
REWARD 'reward' | LEVER
```

Listing 15: Two equivalent `@phase` sections for variable interval

```
@phase {'label':'variable_interval1', 'end': 'reward = 25'}
FI3     'lever'     | FI3=2:ON   | FI3
FI2     'lever'     | FI2=1:ON   | FI2
ON      'lever'     | 'R':REWARD | ON
REWARD 'reward' | ON(1/3),FI2(1/3),FI3(1/3)
```

```
@phase {'label':'variable_interval2', 'end': 'reward = 25'}
T3 'lever'         | T2
T2 'lever'         | ON
ON 'lever'         | R:REWARD | ON
REWARD 'reward'    | ON(1/3),T2(1/3),T3(1/3)
```

Listing 16: Two equivalent `@phase` sections for variable ratio

```
@phase {'label':'variable_ratio1', 'end': 'reward = 25'}
FR3 'lever'      | 'R'=2:ON    | FR3
FR2 'lever'      | 'R'=1:ON    | FR2
ON 'lever'       | 'R':REWARD | ON
REWARD 'reward' | ON(1/3),FR2(1/3),FR3(1/3)

@phase {'label':'variable_ratio2', 'end': 'reward = 25'}
R3 'lever'       | 'R':R2    | R3
R2 'lever'       | 'R':ON    | R2
ON 'lever'       | R:REWARD | ON
REWARD 'reward' | ON(1/3),R2(1/3),R3(1/3)
```

Listing 17: A `@phase` section for reward after a fixed time

```
@phase {'label':'fixed_time', 'end':'reward = 25'}
LEVER  'lever'  | 5: REWARD | LEVER
REWARD 'reward' | LEVER
```

Listing 18: A `@phase` section for reversal learning

```
@phase {'label':'lever_1_rewarded', 'end': 'CHOICE = 100'}
CHOICE  'two_levers' | 'lever 1':REWARD | CHOICE
REWARD  'reward'     | CHOICE

@phase {'label':'lever_2_rewarded', 'end': 'CHOICE = 100'}
CHOICE 'two_levers' | 'lever 2':REWARD | CHOICE
REWARD 'reward'     | CHOICE
```

6 `@RUN`

The `@RUN` section specifies and runs a simulation. See Listing 19 for the syntax.

Listing 19: Syntax for a `@RUN` section

```
@RUN
```

```
@RUN run_label

@RUN run_label
phase1, phase2, ...
```

`run_label` is the name of the simulation run. If there are several `@RUN` sections in a script, `run_label` are used in the postprocessing commands to, e.g., plot the output from one specific run. If `run_label` is omitted, the simulation will be given the automatic label `run1`, `run2` and so on, numbered consecutively in order.

The phases to use in the simulation are specified as a list of phase names on a separate line within the `@RUN` section. This is optional – if there is no such line of phase names, all phases defined above the `@run`-statement will be used.

It is also possible to override any parameter in the `@PARAMETERS` section, which then is only used locally within this `@RUN` section. For example, in the `@RUN` section in Listing 20, the mechanism `SR` and the β-value 0.2 is used, despite any other values specified in `@PARAMETERS`.

Listing 20: Example of overriding parameters in a `@RUN` section

```
phase = phase1, phase2
mechanism: SR
beta: 0.2
@RUN
```

7 Visualization commands

The commands for visualizing simulation data can be found in Table 7.

The commands `@vplot`, `@wplot`, `@pplot` and `@nplot` produces plots in the current axes. Axes objects are created using the `@subplot` command (see section 7.4). If a plot command is not preceeded by a `@subplot` command, a default axes is created.

7.1 `@vplot`, `@wplot`, `@pplot`

The syntax for `@vplot`, `@wplot`, and `@pplot` can be found in Listing 21.

Listing 21: Syntax for `@vplot`, `@pplot` and `@wplot`

```
@vplot E->R options
@pplot S->R options
@pplot E->R options
@wplot E    options
```

Here, `E` is a stimulus element, `R` is a behavior, and `S` a list of stimulus elements. The argument `options` is a list of `option:value` where possible options and values are described in section 7.1.1.

7.1.1 The options

The supported options to the visualization commands can be found in Table 8.

The option `subject` only has effect if the option `subjects` is > 1. When specifying a certain subject, use a zero-based index. For example, if the parameter `subjects` is 4, the valid integer values for the option `subject` are 0, 1, 2 and 3. If `subject` is `all`, one plot per subject will be rendered. If `subject` is `average`, the plotted quantity is the average over the subjects.

When a simulation has been completed after a `@run` statement, the simulation history

$$H = (S_1, R_1, S_2, R_2, S_3, R_3, \dots) \tag{1}$$

is a sequence of alternating stimuli and responses, where each R_i is the response to S_i. We call H the *history sequence* for the simulation.

Each stimulus-response pair constitutes a time-step in the simulation, starting with time step 0. Below, the time-steps in the sequence history (1) are indicated.

$$\underset{t=0}{\Big|} S_1, R_1, \underset{t=1}{\Big|} S_2, R_2, \underset{t=2}{\Big|} S_3, R_3, \underset{t=4}{\Big|} \dots$$

The option `steps` controls at which time-steps to plot the quantity in question. The first (time-step 0) and the last time step are always included. The default value is `all` which plots at each time-step, i.e. the value after each stimulus-response pair. In this case, the x-axis will be from 0 to the total number of time-steps. If `steps` is a string or a tuple of strings, the plot will only display the value after each occurence of this string/tuple (and at the first and at the last time-step). In this case, the value at $x = i$ in the plot is the value after the ith occurence of the string/tuple in the history sequence. If `steps` is a list where every other element in a stimulus and ever other element a response, the plot will only reflect the value after each occurence of the stimulus-responses in the history sequence H.

The option `xscale_match` only has effect when `xscale` is not `all`, in other words when `xscale` is a search pattern.

If `xscale_match` is `off`, when searching for a string s, it is also counted as a hit if a tuple S in H (a stimuli composed of a number of stimulus

elements) *includes* s, i.e. $s \in S$. When searching for a tuple t, it is also counted as a hit if a tuple S in H *includes* t (as sets), i.e. $t \subseteq S$.

If `xscale_match` is `on`, the pattern searched for must exactly match the history sequence H.

For example, if the history sequence is

$$\underset{t=0}{\Big|} E_1, R_1, \underset{t=1}{\Big|} (E_1, E_2), R_1, \underset{t=2}{\Big|} E_2, R_3, \underset{t=3}{\Big|} (E_1, E_3), R_3, \underset{t=4}{\Big|} E_2, R_1, \underset{t=5}{\Big|} E_3, R_3 \underset{t=6}{\Big|}$$

and `xscale` is E_1, the plot will be rendered at time steps $t = 0, 1, 2, 4, 6$ if `xscale_match` is `subset`. If `xscale_match` is `exact`, the plot will be rendered at time steps $t = 0, 1, 6$.

7.2 `@nplot`

The command `@nplot` searches for specific elements in the history sequence H, counts the number of hits at time steps specified with the `steps` option, and plots the result.

The syntax of `@nplot` is

Listing 22: Syntax for `@nplot`

```
@nplot expr options
```

Here, the `options` are the same as in section 7.1.

The argument `expr` is either a stimulus element, a response, a list of stimulus elements (for a compound stimulus), or a consecutive subsequence `S1->R1->S2->R2->...` of the history sequence H, where each `S`i is a stimulus element or a list of stimulus elements (a compund stimulus). In other words, it works similarly to the `xscale` property described in section 7.1.1. Thus, it can search for a specific stimulus element (for example `reward`), a specific response (for example `R`) or a specific compund stimulus consisting of several stimulus elements (for example `E1,E2,E3`) in H and plot the result.

`@nplot` can also search for any consecutive subsequence in H using a list (for example `context,reward->R`) which counts the number of times the compound stimulus `context,reward` got the response `R`.

As options `options`, `@nplot` supports the properties in Table 8. In addition, the properties in Table 9 are supported.

The property `cumulative` should be `on` (default) or `off`. This option controls whether the counting should be at each time-step (in which case

each value on the y-axis in the plot is either 0 or 1) or cumulatively (in which case the resulting plot is a non-decreasing function).

The property `match` should be `on` (default) or `off`. It controls whether the searching for `expr` in H should be exactly fulfilled or only inclusive, just like the property `xscale_match` described in section 7.1.1.

7.3 `@figure`

The `@figure` command creates a figure window.

Listing 23: Syntax for `@figure`

```
@figure title figure_options
```

where `title` is a string used as the title for the figure, and `figure_options` is a dictionary with the keywords to `matplotlib.figure.Figure` controlling the figure size, the figure patch facecolor and edgecolor, etc.[7]

7.4 `@subplot`

The `@subplot` command creates an axes in which to plot.

Listing 24: Syntax for `@subplot`

```
@subplot grid subplot_options
```

where `grid` is three nonzero digits specifying (i) the number of rows, (ii) the number of columns in a grid of axes, and (iii) in which grid cell to place the axes. For example, `211` produces a subaxes in a figure which represents the top plot (i.e. the first) in a 2 row by 1 column notional grid. The options `subplot_options` is a dictionary with the keywords to `matplotlib.pyplot.subplot` controlling, for example, the background color of the subplot.[8]

The `@subplot` command creates an axes in the figure created by the preceeding `@figure` command. If a `@subplot` command is not preceeded by a `@figure` command, a default figure window is created.

7.5 `@legend`

The `@legend` command places a legend on the current axes.

[7] See `https://matplotlib.org/api/_as_gen/matplotlib.figure.Figure.html` for the supported options.

[8] See `https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.subplot` for the supported options.

Listing 25: Syntax for `@legend`

```
@legend labels legend_options
```

where `labels` is a string or a tuple of strings for custom labels. If not specified, automatic labels will be used. The options `legend_options` is a dictionary with the keywords to `matplotlib.pyplot.legend` controlling the font size, the legend's background color, etc.[9]

7.6 Some examples of plotting commands

Listing 26 shows some examples of the use of the visualization commands `@figure`, `@subplot`, `@vplot`, `@wplot`, `@nplot`, and `@legend`. It is assumed to be preceeded by `@run run1` and `@run run2` that run two simulations.

Listing 26: Some examples of plotting commands

```
# Plot v(S,R) in a default axes in a default figure
@vplot S->R

# Plot v(S,R) as a red dashed line in the same axes as
# the above plot
@vplot S->R0  linecolor:red, linestyle:dashed

# Plot w(S) with dot-markers in a blue axis in a yellow
# figure with figure title "Figure Title"
@figure Figure Title
facecolor:yellow
@subplot 111 facecolor:blue
@wplot S marker:.

# Plot p(S,R) from simulation run1 together with p(S,R)
# from simulation run2, and add a custom legend
@figure
@pplot S->R
    runlabel:run1
    label:p(S,R) run1
@pplot S->R runlabel:run2
    label:p(S,R) run2
@legend

# Plot n(R0) and n(R) in the same axes
@figure
nplot R0
nplot R
```

[9] See `https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.legend` for the supported options.

```
# Plot n(R0) and n(R) in two subplots
@figure
@subplot 211
nplot R0
@subplot 212
nplot R
```

8 Exporting data

Each plotting command has a corresponding data export command, which exports the data to an external csv-file. In addition the `@hexport` command exports a history sequence of stimulus-response pairs. The export commands can be found in Table 10.

The syntax for the export commands can be found in Listing 27.

Listing 27: Syntax for `@vplot`, `@pplot`, `@wplot` and `@nplot`

```
@vexport (E,R) value_options
@pexport (E,R) value_options
@wexport  E    value_options
@nexport expr  value_options
@hexport value_options
```

As value-options `value_options`, the data export commands `@vexport`, `@wexport`, `@pexport`, and `@nexport` supports the same properties as the corresponding plot command (see Table 8 and Table 9). In addition, the properties in Table 11 are supported. The command `@hexport` only supports the parameters `runlabel` and `filename`.

8.1 Format of the csv-file

The data export commands exports the data as a csv-file with two or more columns. The first column contains step numbers (corresponding to the x-axis in the corresponding plot command). The second column onwards contains the data for the specified quantity for each subject (controlled by the `subject` parameter).

The `@hexport` command exports a csv-file with three or more columns. Column 1 contains step numbers. Columns 2 and 3 contains the stimulus and response, respectively, for subject 1. Column 4 and 5 contains the stimulus and response, respectively, for subject 2, etc. All subjects are included.

9 Changing individual parameters or phase lines

The scripting language supports editing individual parameters and phase lines. For example, after a simulation with a given set of parameter values, it is possible to change the value of one of them and run a simulation again. See Listing 28 for an example.

Listing 28: Changing an individual parameter

```
@parameters
{
'subjects'          : 1
'mechanism'         : 'Enquist',
'behaviors'         : ['R','R1'],
'stimulus_elements' : ['context','reward','US','CS','lever'],
'start_v'           : {'context':-1,'default':0},
'start_w'           : {'default':0},
'alpha_v'           : 1,
'alpha_w'           : 1,
'beta'              : 1,
'behavior_cost'     : {'R':1,'default':0},
'u'                 : {'reward':10, 'default': 0},
'omit_learning'     : ['US', 'CS']
}

@phase {'label':'fixed_time', 'end':'reward = 25'}
LEVER   'lever'   | 5: REWARD  | LEVER
REWARD  'reward'  | LEVER

@run {'label':'beta=1'}

@parameters
{
'beta':0.5
}

@run {'label':'beta=0.5'}
```

It is also possible to change an individual phase line without having to specify the entire phase again. See Listing 29.

Listing 29: Changing an individual parameter

```
@phase {'label':'fixed_ratio', 'end':'reward = 25'}
OFF 'lever'       | R=4: ON   | OFF
ON  'lever'       | R: REWARD | ON
REWARD  'reward'  | OFF

@run {'label','ratio=4'}
```

```
@phase {'label':'fixed_ratio'}
OFF 'lever'        | R=5: ON    | OFF

@run {'label','ratio=5'}
```

Parameter name	Value	Default	Description
`subjects`	A positive integer	1	The number of subjects
`mechanism`	`GA` (Genetically Guided Associative Learning), `SR` (Stimulus-Response learning), `ES` (Expected SARSA), `QL` (Q-learning), `AC` (Actor critic)		Which learning mechanism to use (see Table 3)
`behaviors`	A list of behavior names		The behavior repertoire (see Section 4.5.1)
`stimulus_elements`	A list of stimulus element names		The possible stimulus elements (see Section 4.5.1)
`start_v`	A number, or a list of `(S,B):val` and `default:val` where `S`∈`stimulus_elements`, `B`∈`behaviors` and `val` is a number	0	The initial v-values (see Section 4.5.2)
`alpha_v`	See `start_v`	1	α_v
`alpha_w`	A number, or a list of `S:val` and `default:val` where `S`∈`stimulus_elements` and `val` is a number	1	α_w
`beta`	A number	1	β
`behavior_cost`	A number, or a list of `B:val` and `default:val` where `S`∈`behaviors` and `val` is a number	0	The cost for each behavior
`u`	See `alpha_w`	0	The u-values
`response_requirements`	A list of `B:`S_1,S_2,... where `B`∈`behaviors` and S_i ∈`stimulus_elements`	No restrictions	The available stimulus elements for each behavior (see Section 4.5.3)
`bind_trials`	`on` or `off`	`off`	Whether ot not to bind learning between trials (see Section 4.5.4)

Table 4: The parameters.

Case	Description
`countrow()=N`	If this line has been visited `N` times *consecutively*.
`countrow(e)=N`	If event `e` has occured on this line `N` times (since this line was entered). The event `e` is either a stimulus element or a response.
`count(e)=N`	If event `e` has occured on this line `N` times (since the start of the phase or since it was last reset with `count_reset(e)`). The event `e` is either a line label, a stimulus element or a response.
`var=N`	If the variable `var` has the value `N`.
`R`	If the response to the stimulus on this line was `R`.

Table 5: The format of `condition` in a logic case in a phase line.

Case	Description
`lbl`	Go to the line with label `lbl`.
`lbl(`p`)`	Go to the line with label `lbl` with probability p.
`lbl1(`p_1`),lbl2(`p_2`),...,lblN(`p_N`)`	Go to `lbl1` with probability p_1, to `lbl2` w.p. p_2, etc.

Table 6: The format of `goto` in a logic case in a phase line.

Command name	Purpose
`@vplot`	Plots a v-variable against time-steps as a line plot
`@wplot`	Plots a w-variable against time-steps as a line plot
`@pplot`	Plots a probability (of a specific response to a specific stimulus) against time-steps as a line plot
`@nplot`	Plots the number of occurences of a specific stimulus, stimulus element, behavior or a sequence of them
`@figure`	Creates a figure window to hold axes objects
`@subplot`	Creates an axes object to hold the plots
`@legend`	Creates a legend for line labels

Table 7: The visualization commands

Parameter	Value	Default	Description
`runlabel`	A string	The label of the last `@run`	The `@run` label
`subject`	An integer (zero-based index) or `'average'` or `'all'`	`'average'`	Which subjects to include
`xscale`	`'all'` or a string or a tuple or a list	`'all'`	The steps at which to plot
`xscale_match`	`'exact'` or `'subset'`	`'subset'`	Use exact or subset matching for `steps`
`phase`	A string or a tuple of strings	All phases	Which phase(s) to include

Table 8: The value-options to `@vplot`, `@wplot`, `@pplot` and `@nplot`.

Parameter	Value	Default	Description
`cumulative`	`'on'` or `'off'`	`'on'`	Cumulative counting
`match`	`exact` or `subset`	`subset`	Use exact matching for `expr`

Table 9: The additional value-options to `@nplot`.

Command name	Purpose
`@vexport`	Exports data for a v-variable against time-steps
`@wexport`	Exports data for a w-variable against time-steps
`@pexport`	Exports probabilites (of a specific response to a specific stimulus) against time-steps
`@nexport`	Exports data for the number of occurences of a specific stimulus, stimulus element, behavior or a sequence of them
`@hexport`	Exports the stimulus-response pair for each step, together with the step numbers.

Table 10: The export commands.

Parameter	Value	Default	Description
`filename`	String		CSV-file name

Table 11: The additional value-options to the export commands.