Learning Simulator
User's Guide

Markus Jonsson

May 3, 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

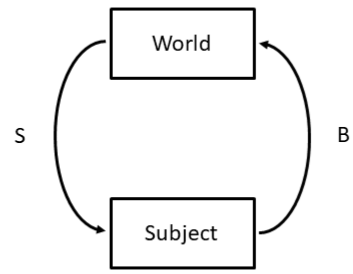

Figure 1: The world and the subject.

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: $S_1 \to B_1 \to S_2 \to B_2 \to S_3 \to ....$

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 is a compound of stimulus elements (E) 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

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

Mechanism	Memory states	Memory updates
Rescorla-Wagner[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]	$v_{S\to B}$	$\Delta v_{S\to B} = \alpha(r_{S'} + v_{S'\to B'} - v_{S\to B})$
Actor-critic[4]	$v_{S\to B}$, 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[5]	$v_{S\to B}$, w_S	$\Delta v_{S\to B} = \alpha_v(u_{S'} + w_{S'} - v_{S\to B})$ $\Delta w_S = \alpha_w(u_{S'} + w_{S'} - c_B - w_S)$

Table 3: Learning mechanisms and their memory updates.

[1] Bush and Mosteller [1951], Rescorla & Wagner [1972]

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

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

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

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

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 specify 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. It is also possible to open a text file using the **File**-menu. A line starting with # in a script is ignored. All leading and trailing spaces/tabs on each line are also ignored.

A Learning Simulator script consists of a number of sections. Each section starts with a keyword and each keyword starts with `@`. The keywords are `@comment`, `@parameters`, `@phase`, `@run`, `@figure`, `@subplot`, `@vplot`, `@wplot`, `@pplot`, `@nplot`, and `@legend`.

Below follows the description of each of the sections.

[2] Watkins [1989], Watkins & Dayan [1992]

[3] Rummery & Niranjan [1994]

[4] Witten [1977], Barto et al. [1983]

[5] Enquist et al. [2016]

4.1 `@comment`

The `@comment` section can contain any text. It may be used to describe the project, or for references, etc. It is usually places in the beginning of the script. Below follows an example.

Listing 2: An example of a `@comment` section

```
@comment
Study the effect of CS duration in the chaining model
Holland, P. C. (1977). Conditioned stimulus as a determinant
of the form of the Pavlovian conditioned response.
Journal of Experimental Psychology: Animal Behavior
Processes, 3(1), 77.

Project start date: 20170621
```

4.2 `@parameters`

The `@parameters` section sets the values of the parameters used in a simulation. They are specified as a Python dictionary whose keys are the parameter names. The supported parameter names and their values can be found in Table 4. Note that parameters without a default value are required.

Use the property `response_requirements` if not all behaviors are possible responses to each stimulus element. It is specified as a dictionary where each key is a behavior B (in `behaviors`) and the corresponding value is a list of the stimulus elements (a sublist of `stimulus_elements`) that B is a possible response to. If a behavior B is not present in `response_requirements`, then B is a possible response to each stimulus element in `stimulus_elements`.

Parameter name	Value	Default	Description
subject	A positive integer	1	The number of subjects
mechanism	'GA', 'Rescorla_Wagner', 'SARSA', 'Q_learning', 'Actor_critic'		The name of the learning mechanism
behaviors	A list of strings		The behavior repertoire
stimulus_elements	A list of strings		The possible stimulus elements
start_v	A number, or a dictionary where each key is 'default' or a tuple (E, B) where $E \in$stimulus_elements and $B \in$behaviors	0	The initial v-values
alpha_v	A real number	1	α_v
alpha_w	A real number	1	α_w
beta	A real number	1	β
behavior_cost	A number, or a dictionary where each key is 'default' or $B \in$behaviors	0	The cost for each behavior
u	A number, or a dictionary where each key is 'default' or $E \in$stimulus_elements	0	The u-values
omit_learning	A list of stimulus elements (a subset of stimulus_elements)	[ ]	The stimulus element(s) to omit when updating v and w
response_requirements	A dictionary where each key is a behavior and each value a sublist of stimulus_elements		The available stimulus elements for each behavior

Table 4: The parameters in a @parameters section.

An example of a @parameters section can be found in Listing 3.

Listing 3: An example of a @parameters section

```
@parameters
{
'subjects'           : 1
'mechanism'          : 'GA',
'behaviors'          : ['R0','R1','R2'],
'stimulus_elements'  : ['S1','S2','reward','new trial'],
'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},
'omit_learning'      : ['new trial']
}
```

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 an *end condition*.

A `@phase` section consists of a number pf *phase lines*. Each phase line consists of a label, a 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. The phase starts at the first phase line. The basic syntax of a `@phase` section is as follows:

Listing 4: The basic syntax of a `@phase` section

```
@phase {'label':phaselbl, 'end':condition}
lbl1   stimulus1   |   logic1
lbl2   stimulus2   |   logic2
lbl3   stimulus3   |   logic3
...
```

The label (`phaselbl` in the above listing) should be a string which provides a means of specifying which phases to include in a simulation (see the `@run` statement in Section 6). The end condition (`condition` in the above listing) is a string of 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, `'end':'reward'=20` ends the phase after 20 exposures to the stimulus `'reward'`.

Each stimuli (`stimulus1`, `stimulus2`, ... in the above listing) is specified either as a single stimulus element or a number of simultaneous stimulus elements. To specify a number of simultaneous elements, use a tuple of strings, for example `('E1','E2')`. To specify a single element, use a string (for example `'S'`) or a 1-tuple (`('S',)`).

The logical part (`logic1`, `logic2`, ... in the above listing) consists of one or more cases, separated by |. Each case must have one of the forms specified in Table 5.

The cases can be combined, separated by |, to form an if-else statement. For example, `'R1':lbl1` | `'R2':lbl2` | `lbl3` means (in psedo-code)

```
if the response was R1:
```

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.
`N: lbl`	If this line has been visited `N` times *consecutively*, go to `lbl`.
`'R': lbl`	If the response was `R`, go to `lbl`.
`'R': lbl(`p`)`	If the response was `R`, go to `lbl` with probability p.
`'R': lbl1(`p_1`),lbl2(`p_2`),...,lblN(`p_N`)`	If the response was `'R'`, go to `lbl1` with probability p_1, to `lbl2` w.p. p_2, etc.
`'R'=N: lbl`	If the response to the stimulus on this line has been `R` `N` times *consecutively*, go to `lbl`.

Table 5: The cases in the logical part of a phase line.

```
    go to lbl1
else if the response was R2:
    go to lbl2
else:
    go to lbl3
```

Note that the three logical parts in Listing 5 are equivalent.

Listing 5: Three equivalent logical parts

```
lbl1(1/3),lbl2(1/3),lbl3(1/3)
lbl1(1/3) | lbl2(1/2),lbl3(1/2)
lbl1(1/3) | lbl2(1/2) | lbl3
```

A few examples of `@phase` sections can be found in Listings 6 to 13.

Listing 6: 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 7: 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 8: 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 9: 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 10: 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 11: 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 12: 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 13: 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` statement runs a simulation, using the set of parameter values defined above the `@run` statement. It takes an optional dictionary as argument with `'label'` and `'phases'` as keys.

Listing 14: Syntax for `@run`

```
@run {'label':runlabel, 'phases':('phase1','phase2',...)}
@run {'label':runlabel, 'phases':'phase1'}
```

The phases to use in the simulation are specified in the value for the `'phases'` key in the dictionary. This can be either a string (for a single phase) or a tuple of strings (for several phases). If `'phases'` is not specified, all phases defined above the `@run`-statement will be used.

If `'label'` is not specified, the simulation will be given the automatic label `'run1'`, `'run2'` and so on.

7 Visualization commands

The commands for visualizing simulation data can be found in Table 6.

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 6: The visualization commands

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 15.

Listing 15: Syntax for `@vplot`, `@pplot` and `@wplot`

```
@vplot (E,R) value_options plot_options
@pplot (S,R) value_options plot_options
@pplot (E,R) value_options plot_options
@wplot  E    value_options plot_options
```

Here, `E` is a stimulus element (a string), `R` is a behavior (a string), and `S` a tuple of stimulus elements. The argument `value_options` is a dictionary with options described in section 7.1.1. The argument `plot_options` is a dictionary with the keywords to `matplotlib.lines.Line2D` controlling line style, line color, marker style, marker size, etc.[6]

Both `value_options` and `plot_options` are optional. If only one dictionary is specified, it is interpreted as `value_options`. To specify only `plot_options`, use an empty dictionary as `value_options`:

```
@vplot (E,R) {} plot_options
```

7.1.1 The value options

The supported value-options can be found in Table 7.

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
`steps`	`'all'` or a string or a tuple or a list	`'all'`	The steps at which to plot
`exact_steps`	`'on'` or `'off'`	`'off'`	Use exact matching for `steps`
`phase`	A string or a tuple of strings	All phases	Which phase(s) to include

Table 7: The value-options to `@vplot`, `@wplot`, `@pplot` and `@nplot`.

The option `subject` only has effect if the parameter `subjects` is > 1. When specifying a certain subject, use a zero-based index. For example, if the parameter `subjects` is 5, 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)

[6] See `https://matplotlib.org/api/_as_gen/matplotlib.lines.Line2D.html` for the supported plot options.

are indicated.

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

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 `exact_steps` only has effect when `steps` is not `'all'`, in other words when `steps` is a search pattern.

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

For example, if the history sequence is

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

and `steps` is E_1, the plot will be rendered at time steps $t = 0, 1, 2, 4, 6$ if `exact_steps` is `'off'`. If `exact_steps` is `'on'`, 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, either at each time step (specified with the `steps` option), or cumulatively, and plots the result.

The syntax of `@nplot` is

Listing 16: Syntax for `@nplot`

```
@nplot expr value_options plot_options
```

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

The argument `expr` is either a string, a tuple of strings, or a list being a consequtive subsequence of the history sequence H. In other words, it works similarly to the `steps` property described in section 7.1.1. As such, 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 plots 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 stimulus `('context','reward')` got the response `'R'`.

As value-options `value_options`, `@nplot` supports the properties in Table 7. In addition, the properties in Table 8 are supported.

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

Table 8: The additional value-options to `@nplot`.

The property `cumulative` should be `'on'` (default) or `'off'`. This options controls whether the counting should be at each time-step (i.e. 0 or 1) or cumulatively.

The property `exact_n` 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 `exact_steps` described in section 7.1.1

7.3 `@figure`

The `@figure` command creates a figure window.

Listing 17: 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] See `https://matplotlib.org/api/_as_gen/matplotlib.figure.Figure.html` for the supported options.

7.4 `@subplot`

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

Listing 18: 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.

Listing 19: 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 20 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 {'label':'run1'}` and `@run {'label':'run2'}` that run two simulations.

Listing 20: Some examples of plotting commands

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

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

```
# 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
@figure 'w(S)' {'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'}
@pplot ('S','R') {'runlabel','run2'}
@legend ('p(S,R) run1', 'p(S,R) run2')

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

# 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 9.

The syntax for the export commands can be found in Listing 21.

Listing 21: 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
```

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 9: The export commands.

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 7 and Table 8). In addition, the properties in Table 10 are supported. The command `@hexport` only supports the parameters `runlabel` and `filename`.

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

Table 10: The additional value-options to the export commands.

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 22 for an example.

Listing 22: 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 23.

Listing 23: 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'}
```