pyLSEx User Manual

version: 1.0

January 2017

pyLSEx was designed by Michael Lindner and Doug Saddy and programmed by Michael Lindner

l-s-ex@gmx.co.uk

University of Reading, United Kingdom

Center for Integrative Neuroscience and Neurodynamics

https://www.reading.ac.uk/cinn/cinn-home.aspx

Content

1. Installation

- Copy the files onto a folder on your hard drive. e.g. in C:\LSEx

2. Dependencies

pyLSEx is a Python based based toolbox. Therefore, for the usage of pyLSEx you need to have Python 2.7 installed. Additionally the following python packages need to be installed:

wx
datetime
pickle

3. License

4. Lindenmayer System Explorer

This tool is designed to generate and/or modify Lindenmayer systems (grammars).

First steps:

- To start the Lindenmayer System Explorer open a terminal (command window) and change to the folder where you copied the *pyLSEx.py* file. Start the pyLSEx.py by typing in: *python pyLSEX.py*
- From here you will have two choices (via two buttons):
  - Lindenmayer System Generator
    - Using the Generator you can generate predefined Lindenmayer systems (e.g. Fibonacci) or you can define your own rules to create grammar. See detailed description in section 5.
  - Lindenmayer System Modifier
    - Using the Modifier you can change and modify existing L-systems, by building rules to iteratively replace chains of n characters of the system with chains of m other ones. See detailed description in section 6.

Enjoy playing around with this tool!

5. Lindenmayer System Generator

This tool is designed to generate predefined Lindenmayer systems (e.g. Fibonacci). But also user defined rules can be specified to create new L-systems.

Usage:

Select L-system

You can select a predefined system (e.g. Fibonacci), specify new
rule or load user defined rules:

- predefined rules.
  - The rules for the predefined system will be presented in the fields under the drop-down menu after making your choice.
- User defined rules
  - You can also define a new grammar by selecting User defined. A window will open to specify the number of rules (replacements). A window, where you can specify the rules will then open. Here you can specify rules for replacing n characters with a specific amount of m characters. The n's of all replacements need to have the same length! You can save the defined rules by pressing the "Save rules" button. The rule will be saved in a file called user_defined_rules*_[date_time]*.txt .

Initial string:

The initial string for the L-system generation is by default: 0 for the predefined systems and for the user defined system it is the string (left hand side) of the first rule!
But the initial string can be changed individually.
IMPORTANT: The values in the "initial string" field need to be included within the rules you define!

Number of iterations:

With the number of iterations you can specify how often the rules should be used iteratively.

Output folder:

You need to specify an output folder by typing in a path or by using the browse button.

Output file prefix:

You can specify an output file prefix.

Output file type:

You can choose the type of output which should be stored. Either a pickle .dat file, a text file, or both are valid:

In the case of a:

- .dat file, three variables are stored: the grammar of each iteration, the rule and a vector with the length of each iteration.
- .txt file, three output files are stored. One with the grammar, one including the length of each iteration and one containing the rule.

Generate L-system:

After setting up all parameters, you simply need to press the generate L-system button.

If the n's of the replacement rules have a length bigger than 1, then you have three options for types of replacement (how the rules should be applied to the system):

a. segmentwise

In the segmentwise replacement all vocabulars need to have the same length. (e.g. 101, 111, 000, etc,). The L-system is then cut into segments of the same length and the replacement is then performed for each segment.

b. continuously

In the continuous replacement the algorithm goes through the whole grammar elementwise and replaces the n elements from the vocabulary with the m elements that you defined! Each element will be checked and can be used more than once for a replacement!

e.g. system: 00111011001001111001001111000 0

rule: 111-->2 ; 110-->3 ; 100-->4

results: 00231034004002234004002234000 0

c. continuously (skip last n)

This type of replacement follows the same protocol as the one above, but when identifying a match with your rule the next n-1 characters (length of the vocabular) of the system are skipped. Here, each element will only be used once for a replacement!

e.g. system: 00111011001001111001011110000

rule: 111-->2 ; 110-->3 ; 100-->4

results: 0020304230423000

The L-system will be generated and saved in the output files.

6. Lindenmayer System Modifier

This tool is designed to modify Lindenmayer systems (grammar). With this tool, rules can be defined to replace n with m number of elements in the system.

e.g. 111 with 1

10 with 2

110 with 10

1 with 1011 etc.

Usage:

Load L-System:

First you need to load a system by using the Load L-system button. You can easily load a pickle .dat output file from the Lindenmayer System Generator.

Define rules:

You can define any rules for replacement by pressing the "Define rules" button. After pressing the button a new window opens where you can specify the number of rules (replacements). Afterwards another window opens, where you can specify the rules

e.g. 101 ---> 1

010 ---> 0 etc.

You can save the defined rules by pressing the "Save rules" button. Then you can specify a name for the rule and it will be saved in a file called user_defined_rules*_[date_time]*.dat .

You have three options for types of replacement (how the rules should be applied to the system):

1. segmentwise

   In the segmentwise replacement all vocabulars (left side of the rules, which should be replaced) need to have the same length. (e.g. 101, 111, 000, etc,). The grammar is then cut into segments of the same length and the replacement is then performed for each segment.

2. continuously

   In the continuous replacement the algorithm goes through the whole grammar elementwise and replaces the n elements from the vocabulary with the m elements that you defined! Each element will be checked and can be used more than once for a replacement!

e.g. system: 00111011001001111001001111000 0

rule: 111-->2 ; 110-->3 ; 100-->4

results: 00231034004002234004002234000 0

3. continuously (skip last n)

   This type of replacement follows the same protocol as the one above, but when identifying a match with your rule the next n-1 characters (length of the vocabular) of the system are skipped. Here, each element will only be used once for a replacement!

   e.g. system: 00111011001001111001001111000 0

   rule: 111-->2 ; 110-->3 ; 100-->4

   results: 0020304230423000

Number of iterations:

With the number of iterations you can specify how often the rules should be used iteratively.

Output folder:

You need to specify an output folder by typing in a path or by using the browse button.

Output file prefix:

You can specify an output file prefix

Output file type:

You can choose the type of output which should be stored. Either a pickle .dat file, a text file, or both are valid:

In the case of a:

- .dat file, three variables are stored: the grammar of each iteration, the rule and a vector with the length of each iteration.
- .txt file, three output files are stored. One with the grammar, one including the length of each iteration and one containing the rule.

Modify L-system:

After setting up all parameters, you simply need to press the Modify L-system button.

The loaded L-system will be modified and saved in the output files.