Turle Defense\_SIIS\_python\_simulation

AUTHOR

Version 3

Thu May 5 2022

# Table of Contents

Table of contents

# Turle Defense\_SIIS\_python\_simulation

Python Simulation of the **SIIS** Model ‘Competing Memes Propagation on Networks: A Network Science Perspective’ Using random number generation

##### Author

Robert Bacigalupo

Tyler Ovenden

Auerman Atif

##### Version

3.0

##### Date

MAY 2022

**X**. Wei, N. C. Valler, B. A. Prakash, I. Neamtiu, M. Faloutsos, and C. Faloutsos, “Competing memes propagation on networks: A network science perspective,” IEEE Journal on Selected Areas in Communications, vol. 31, no. 6, pp. 1049–1060, 2013.

# Namespace Index

## Packages

Here are the packages with brief descriptions (if available):

**SIIS**  pagenum

# Hierarchical Index

## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

enum.Enum

SIIS.State pagenum

SIIS.Node pagenum

# Class Index

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

**SIIS.Node (Class to implement node object each node object has unique id, list of its neighbors on each edge, and its state )** pagenum

**SIIS.State (Enumeration for States )** pagenum

# File Index

## File List

Here is a list of all files with brief descriptions:

**C:/Users/Bobby the Kid's/Documents/turtledefense/TurtleDefense/PythonTrial/SIIR/SIIS.py**  pagenum

# Namespace Documentation

## SIIS Namespace Reference

### SIISClasses

class **Node**

*Class to implement node object each node object has unique id, list of its neighbors on each edge, and its state.*

class **State**

*Enumeration for States.*

### Functions

def **randomRem** ()

*randomRem picks a random node*

def **randomNeigh** (pick)

*randomNeigh*

def **max\_degree** (**edge**)

*max\_degree*

def **removeNode** (nodea)

*removeNode*

def **set\_simulation** ()

*set\_simulation*

def **count** ()

*count counts number of currently infected sets total\_M values to current values*

def **run\_sim** ()

***run\_sim()*** *carries out actual simulation uses for loop t inside time*

def **plot\_res** (pause, supress, spr)

*plot\_res*

### Variables

int **N** = 200

*Number of nodes.*

int **tm** = 1000

*Discrete time units.*

int **take\_sample** = 0

int **smp\_size** = 10

*sample size*

int **theta** = .10

*Margin off error to see if a Meme won in sampling, if less than theta no clear winner.*

int **beta1** = .05

int **beta2** = .05

int **delta1** = .04

int **delta2** = .04

int **M1\_Wins** = 0

*number of samples that M1 wins*

int **M2\_Wins** = 0

*number of samples M2 Wins*

int **No\_win** = 0

*number of samples with no clear winner*

list **allNodes** = []

*list of all nodes*

int **tot\_inf1** = 0

*total number of infections by meme 1*

int **old\_inf1** = 0

int **old\_inf2** = 0

int **tot\_inf2** = 0

*total number of infected by meme 2 before running sim for plotting*

int **total\_M1** = 0

*total number of nodes in state I1 (for plot)*

int **total\_M2** = 0

*total number of nodes in state I2 (for plot)*

int **total\_S** = 0

*total number of nodes in state S*

**A1** = np.random.randint(2, size=(**N**, **N**), dtype=np.int8)

*SETUP#################################################################################################################################################################################################.*

**A2** = np.random.randint(2, size=(**N**, **N**), dtype=np.int8)

tuple **S1** = (1 - **delta1**) \* np.identity(**N**, dtype=np.int8) + **beta1** \* **A1**

*create System Matrices*

tuple **S2** = (1 - **delta2**) \* np.identity(**N**, dtype=np.int8) + **beta2** \* **A2**

**eigenValues1** = eigenValues1[**idx**]

*get Eigen Values*

**eigenVectors1** = eigenVectors1[:,**idx**]

**eigenValues2** = eigenValues2[**idx**]

**eigenVectors2** = eigenVectors2[:,**idx**]

**idx** = eigenValues1.argsort()[::-1]

*sort list of found eigenvalues to find the largest one*

**node** = **Node**()

*Using our created* ***Node*** *data type, we fill a list with nodes of length N.*

**key**

*sort nodes by ID number*

**id**

**reverse**

**infected\_meme1** = random.sample(**allNodes**,10)

*create first list of infected nodes holds org sets to use for multiple lists*

**infected\_meme2** = random.sample(**allNodes**, 10)

int **edge** = 1

*Run Simulation without Supression.*

### Function Documentation

#### count:SIISSIIS:count def SIIS.count ()

count counts number of currently infected sets total\_M values to current values

Here is the caller graph for this function:

IMAGE

#### max\_degree:SIISSIIS:max\_degree def SIIS.max\_degree ( *edge*)

max\_degree

##### Parameters

|  |  |
| --- | --- |
| *edge* | What edge to delete node from removes node max degree or most connections |

Here is the call graph for this function:

IMAGE

#### plot\_res:SIISSIIS:plot\_res def SIIS.plot\_res ( *pause*, *supress*, *spr*)

plot\_res

plots histogram and time graph of simulation

##### Parameters

|  |  |
| --- | --- |
| *pause* | if true does not plot but sets up graph until ran again with pause = false |
| *supress* | if true plot supression methods as well |
| *spr* | hold name supression method graphed to name on graph |

#### randomNeigh:SIISSIIS:randomNeigh def SIIS.randomNeigh ( *pick*)

randomNeigh

##### Parameters

|  |  |
| --- | --- |
| *pick* | What edge to delete node from picks a random neighbor of a random node |

Here is the call graph for this function:

IMAGE

#### randomRem:SIISSIIS:randomRem def SIIS.randomRem ()

randomRem picks a random node

Here is the call graph for this function:

IMAGE

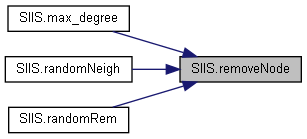
#### removeNode:SIISSIIS:removeNode def SIIS.removeNode ( *nodea*)

removeNode

##### Parameters

|  |  |
| --- | --- |
| *nodea* | The node to be deleted removes the node being passed |

Here is the caller graph for this function:

IMAGE

#### run\_sim:SIISSIIS:run\_sim def SIIS.run\_sim ()

**run\_sim()** carries out actual simulation uses for loop t inside time

Here is the call graph for this function:

IMAGE

#### set\_simulation:SIISSIIS:set\_simulation def SIIS.set\_simulation ()

set\_simulation

sets up infected lists for infection set infected counts to empty set array of stats to empty sets x axis as empty

Here is the call graph for this function:

IMAGE

Here is the caller graph for this function:

IMAGE

### Variable Documentation

#### A1:SIISSIIS:A1 tuple SIIS.A1 = np.random.randint(2, size=(N, N), dtype=np.int8)

SETUP#################################################################################################################################################################################################.

Make sure adj are symmetric and values are 2 or 1 By making it symmetric, we are making sure if **Node** 1 is connected to **Node** 2, then **Node** 2 is also connected to **Node** 1.

#### A2:SIISSIIS:A2 tuple SIIS.A2 = np.random.randint(2, size=(N, N), dtype=np.int8)

#### allNodes:SIISSIIS:allNodes SIIS.allNodes = []

list of all nodes

#### beta1:SIISSIIS:beta1 int SIIS.beta1 = .05

#### beta2:SIISSIIS:beta2 int SIIS.beta2 = .05

#### delta1:SIISSIIS:delta1 int SIIS.delta1 = .04

#### delta2:SIISSIIS:delta2 int SIIS.delta2 = .04

#### edge:SIISSIIS:edge int SIIS.edge = 1

Run Simulation without Supression.

MAIN LOOP################################################################################################################################################################################

Run with Suppression Methods

#### eigenValues1:SIISSIIS:eigenValues1 SIIS.eigenValues1 = eigenValues1[idx]

get Eigen Values

#### eigenValues2:SIISSIIS:eigenValues2 SIIS.eigenValues2 = eigenValues2[idx]

#### eigenVectors1:SIISSIIS:eigenVectors1 SIIS.eigenVectors1 = eigenVectors1[:,idx]

#### eigenVectors2:SIISSIIS:eigenVectors2 SIIS.eigenVectors2 = eigenVectors2[:,idx]

#### id:SIISSIIS:id SIIS.id

#### idx:SIISSIIS:idx SIIS.idx = eigenValues1.argsort()[::-1]

sort list of found eigenvalues to find the largest one

#### infected\_meme1:SIISSIIS:infected\_meme1 SIIS.infected\_meme1 = random.sample(allNodes,10)

create first list of infected nodes holds org sets to use for multiple lists

#### infected\_meme2:SIISSIIS:infected\_meme2 SIIS.infected\_meme2 = random.sample(allNodes, 10)

#### key:SIISSIIS:key SIIS.key

sort nodes by ID number

#### M1\_Wins:SIISSIIS:M1\_Wins SIIS.M1\_Wins = 0

number of samples that M1 wins

#### M2\_Wins:SIISSIIS:M2\_Wins SIIS.M2\_Wins = 0

number of samples M2 Wins

#### N:SIISSIIS:N SIIS.N = 200

Number of nodes.

#### No\_win:SIISSIIS:No\_win SIIS.No\_win = 0

number of samples with no clear winner

#### node:SIISSIIS:node SIIS.node = Node()

Using our created **Node** data type, we fill a list with nodes of length N.

#### old\_inf1:SIISSIIS:old\_inf1 int SIIS.old\_inf1 = 0

#### old\_inf2:SIISSIIS:old\_inf2 int SIIS.old\_inf2 = 0

#### reverse:SIISSIIS:reverse SIIS.reverse

#### S1:SIISSIIS:S1 tuple SIIS.S1 = (1 - delta1) \* np.identity(N, dtype=np.int8) + beta1 \* A1

create System Matrices

#### S2:SIISSIIS:S2 tuple SIIS.S2 = (1 - delta2) \* np.identity(N, dtype=np.int8) + beta2 \* A2

#### smp\_size:SIISSIIS:smp\_size SIIS.smp\_size = 10

sample size

#### take\_sample:SIISSIIS:take\_sample int SIIS.take\_sample = 0

#### theta:SIISSIIS:theta SIIS.theta = .10

Margin off error to see if a Meme won in sampling, if less than theta no clear winner.

#### tm:SIISSIIS:tm SIIS.tm = 1000

Discrete time units.

#### tot\_inf1:SIISSIIS:tot\_inf1 SIIS.tot\_inf1 = 0

total number of infections by meme 1

total number of infected by meme 1 before running sim for plotting

#### tot\_inf2:SIISSIIS:tot\_inf2 SIIS.tot\_inf2 = 0

total number of infected by meme 2 before running sim for plotting

total number of infections by meme 2

#### total\_M1:SIISSIIS:total\_M1 SIIS.total\_M1 = 0

total number of nodes in state I1 (for plot)

#### total\_M2:SIISSIIS:total\_M2 SIIS.total\_M2 = 0

total number of nodes in state I2 (for plot)

#### total\_S:SIISSIIS:total\_S SIIS.total\_S = 0

total number of nodes in state S

# Class Documentation

## SIIS.Node Class Reference

SIIS.Node

Class to implement node object each node object has unique id, list of its neighbors on each edge, and its state.

### Public Member Functions

def **\_\_init\_\_** (self)

*init*

def **addNeighbors** (self)

*addNeighbors adds neighbors to each node method to add neighbors to node, call after Adj matrices are made*

def **attack** (self)

*attack*

def **recover** (self)

### Public Attributes

**id**

**e1\_Neighbors**

**e2\_Neighbors**

**state**

### Static Public Attributes

int **count** = 0

*class variable count of total amount of nodes created, used to create unique* ***Node*** *ids*

### Detailed Description

Class to implement node object each node object has unique id, list of its neighbors on each edge, and its state.

### Constructor & Destructor Documentation

#### \_\_init\_\_:SIIS.NodeSIIS.Node:\_\_init\_\_ def SIIS.Node.\_\_init\_\_ ( *self*)

*init*

##### Parameters

|  |  |
| --- | --- |
| *self* | The object pointer increases count after giving **Node** id |

### Member Function Documentation

#### addNeighbors:SIIS.NodeSIIS.Node:addNeighbors def SIIS.Node.addNeighbors ( *self*)

addNeighbors adds neighbors to each node method to add neighbors to node, call after Adj matrices are made

##### Warning

: Make sure to sort Nodes before using this method

#### attack:SIIS.NodeSIIS.Node:attack def SIIS.Node.attack ( *self*)

attack

simualtes attacks on single node at single time t checks each infected neighbor on its corresponding edge using random number generation, random.random() which outputs a number between 0 and 1 it counts the number of attack from neighbors infected by each meme if the node is in state I1 or I2 then the node does not go through method if C1 > C2 the node becomes infected with meme 1 if C2 > C1 the node becomes infected with meme 2 else the node stays in state S

#### recover:SIIS.NodeSIIS.Node:recover def SIIS.Node.recover ( *self*)

### Member Data Documentation

#### count:SIIS.NodeSIIS.Node:count SIIS.Node.count = 0[static]

class variable count of total amount of nodes created, used to create unique **Node** ids

#### e1\_Neighbors:SIIS.NodeSIIS.Node:e1\_Neighbors SIIS.Node.e1\_Neighbors

#### e2\_Neighbors:SIIS.NodeSIIS.Node:e2\_Neighbors SIIS.Node.e2\_Neighbors

#### id:SIIS.NodeSIIS.Node:id SIIS.Node.id

#### state:SIIS.NodeSIIS.Node:state SIIS.Node.state

#### The documentation for this class was generated from the following file:

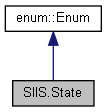
C:/Users/Bobby the Kid's/Documents/turtledefense/TurtleDefense/PythonTrial/SIIR/**SIIS.py**

## SIIS.State Class Reference

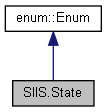
SIIS.State

Enumeration for States.

Inheritance diagram for SIIS.State:

IMAGE

Collaboration diagram for SIIS.State:

IMAGE

### Static Public Attributes

int **S** = 0

int **I1** = 1

int **I2** = 2

### Detailed Description

Enumeration for States.

### Member Data Documentation

#### I1:SIIS.StateSIIS.State:I1 int SIIS.State.I1 = 1[static]

#### I2:SIIS.StateSIIS.State:I2 int SIIS.State.I2 = 2[static]

#### S:SIIS.StateSIIS.State:S int SIIS.State.S = 0[static]

#### The documentation for this class was generated from the following file:

C:/Users/Bobby the Kid's/Documents/turtledefense/TurtleDefense/PythonTrial/SIIR/**SIIS.py**

# File Documentation

## C:/Users/Bobby the Kid's/Documents/turtledefense/TurtleDefense/PythonTrial/SIIR/SIIS.py File Reference

### C:/Users/Bobby the Kid's/Documents/turtledefense/TurtleDefense/PythonTrial/SIIR/SIIS.pyClasses

class **SIIS.State**

*Enumeration for States.*

class **SIIS.Node**

*Class to implement node object each node object has unique id, list of its neighbors on each edge, and its state.*

### Namespaces

namespace **SIIS**

### Functions

def **SIIS.randomRem** ()

*randomRem picks a random node*

def **SIIS.randomNeigh** (pick)

*randomNeigh*

def **SIIS.max\_degree** (edge)

*max\_degree*

def **SIIS.removeNode** (nodea)

*removeNode*

def **SIIS.set\_simulation** ()

*set\_simulation*

def **SIIS.count** ()

*count counts number of currently infected sets total\_M values to current values*

def **SIIS.run\_sim** ()

***run\_sim()*** *carries out actual simulation uses for loop t inside time*

def **SIIS.plot\_res** (pause, supress, spr)

*plot\_res*

### Variables

int **SIIS.N** = 200

*Number of nodes.*

int **SIIS.tm** = 1000

*Discrete time units.*

int **SIIS.take\_sample** = 0

int **SIIS.smp\_size** = 10

*sample size*

int **SIIS.theta** = .10

*Margin off error to see if a Meme won in sampling, if less than theta no clear winner.*

int **SIIS.beta1** = .05

int **SIIS.beta2** = .05

int **SIIS.delta1** = .04

int **SIIS.delta2** = .04

int **SIIS.M1\_Wins** = 0

*number of samples that M1 wins*

int **SIIS.M2\_Wins** = 0

*number of samples M2 Wins*

int **SIIS.No\_win** = 0

*number of samples with no clear winner*

list **SIIS.allNodes** = []

*list of all nodes*

int **SIIS.tot\_inf1** = 0

*total number of infections by meme 1*

int **SIIS.old\_inf1** = 0

int **SIIS.old\_inf2** = 0

int **SIIS.tot\_inf2** = 0

*total number of infected by meme 2 before running sim for plotting*

int **SIIS.total\_M1** = 0

*total number of nodes in state I1 (for plot)*

int **SIIS.total\_M2** = 0

*total number of nodes in state I2 (for plot)*

int **SIIS.total\_S** = 0

*total number of nodes in state S*

**SIIS.A1** = np.random.randint(2, size=(N, N), dtype=np.int8)

*SETUP#################################################################################################################################################################################################.*

**SIIS.A2** = np.random.randint(2, size=(N, N), dtype=np.int8)

tuple **SIIS.S1** = (1 - delta1) \* np.identity(N, dtype=np.int8) + beta1 \* A1

*create System Matrices*

tuple **SIIS.S2** = (1 - delta2) \* np.identity(N, dtype=np.int8) + beta2 \* A2

**SIIS.eigenValues1** = eigenValues1[idx]

*get Eigen Values*

**SIIS.eigenVectors1** = eigenVectors1[:,idx]

**SIIS.eigenValues2** = eigenValues2[idx]

**SIIS.eigenVectors2** = eigenVectors2[:,idx]

**SIIS.idx** = eigenValues1.argsort()[::-1]

*sort list of found eigenvalues to find the largest one*

**SIIS.node** = Node()

*Using our created* ***Node*** *data type, we fill a list with nodes of length N.*

**SIIS.key**

*sort nodes by ID number*

**SIIS.id**

**SIIS.reverse**

**SIIS.infected\_meme1** = random.sample(allNodes,10)

*create first list of infected nodes holds org sets to use for multiple lists*

**SIIS.infected\_meme2** = random.sample(allNodes, 10)

int **SIIS.edge** = 1

*Run Simulation without Supression.*

# Index

INDEX