Sample rst2pdf doc

version 1

Carlos Delgado

October 07, 2020

Contents

Welcome to LAMFRIA’s documentation! **1**
- Description 1
  - Box Counting algorithm 1
    - BoxCounting algorithm: 1
    - Example 1
  - Fixed Size Box Counting Algorithm 1
    - Box Counting Fixed Size module 1
    - Example 2
  - CBBAlgorithm package 2
    - Submodules 2
    - CBBAlgorithm.CBBAlgorithm module 2
    - Module contents 2
  - Genetic strategy 2
    - Genetic 2
    - Example 4
  - SandBox Algorithm 5
    - SBAlgorithm 5
    - Example 5
  - Simulated Annealing strategy 5
    - SimulatedAnnealing.SimulatedAnnealing module 5
    - Example 7
  - robustness package 7
    - Submodules 7
    - robustness.robustness module 7
    - Module contents 7
  - utils package 7
    - Submodules 7
    - utils.utils module 7
    - Module contents 8
- Requeriments 8

Indices and tables **8**

Index **9**

Python Module Index **11**

Welcome to LAMFRIA's documentation!

Description

LAMFRIA is a tool for multifractal and robustness analisys. It also includes two artificial intelligence strategies for calculate fractal dimensions.

Contents:

Box Counting algorithm

This package provides multifractal analysis with Box Counting Algorithm proposed in journal article: Fractal and multifractal properties of a family of fractal networks DOI: 10.1088/1742-5468/2014/02/P02020

BoxCounting algorithm:

`BCAlgorithm.BCAlgorithm.`**`BCAlgorithm`**(`g`, `minq`, `maxq`, `percentNodesT`, `centerNodes=array([], dtype=float64)`)

Calculate fractal dimension with BoxCounting method
Inputs are parameters to configure algorithm behaviour.

Parameters:

- **g** (*Snap PUN Graph.*) – Network.
- **minq** – Minimum value of q
- **minq** – Maximum value of q
- **percentNodesT** (*Integer*) – Number of combinations of center nodes. This value is a percent of the total nodes
- **CenterNodes** (*Numpy 1D Array*) – Calculated center. If this is null, then the centers are calculated

Returns: logR: Numpy arraylogarithm of r/d Indexzero: Integerposition of q=0 in Tq and Dq Tq: Numpy arraymass exponents Dq: Numpy arrayfractal dimensions lnMrq: Numpy 2D arraylogarithm of number of nodes in boxes by radio

Example

```
import sys
import lib.snap as snap
import BCAlgorithm.BCAlgorithm as BCAlgorithm
import numpy

minq = -10
maxq = 10
percentNodesT = 2 #200% of nodes
Rnd = snap.TRnd(1,0)
graph = snap.GenPrefAttach(10000, 10,Rnd)  #ScaleFree with 10 edges per node

logR, Indexzero,Tq, Dq, lnMrq = BCAlgorithm.BCAlgorithm(graph,minq,maxq,percentNodesT)
```

Fixed Size Box Counting Algorithm

This package provides multifractal analysis with Box Counting Fixed Size Algorithm proposed in journal article: Multifractal analysis of complex networks DOI: 10.1088/1674-1056/21/8/080504

Box Counting Fixed Size module

`FSBCAlgorithm.FSBCAlgorithm.`**`FSBCAlgorithm`**(`g`, `minq`, `maxq`, `percentNodesT`, `centerNodes=array([], dtype=float64)`)

Calculate fractal dimension with BoxCounting fixed size method
Inputs are parameters to configure algorithm behaviour.

Parameters:

- **g** (*Snap PUN Graph.*) – Network.
- **minq** – Minimum value of q
- **minq** – Maximum value of q
- **percentNodesT** – Number of combinations of center nodes. This value is a percent of the total nodes
- **percentNodesT** – Center of boxes. All centers have to different and the lenght array must be equal to number of nodes

Returns: logR: Numpy arraylogarithm of r/d Indexzero: Integerposition of q=0 in Tq and Dq Tq: Numpy arraymass exponents Dq: Numpy arrayfractal dimensions lnMrq: Numpy 2D arraylogarithm of number of nodes in boxes by radio

Example

```
import sys
import lib.snap as snap
import FSBCAlgorithm.FSBCAlgorithm as FSBCAlgorithm
import numpy

minq = -10
maxq = 10
percentNodesT = 2 #200% of nodes
Rnd = snap.TRnd(1,0)
graph = snap.GenPrefAttach(10000, 10,Rnd)  #ScaleFree with 10 edges per node

logR, Indexzero,Tq, Dq, lnMrq = FSBCAlgorithm.FSBCAlgorithm(graph,minq,maxq,percentNodesT)
```

CBBAlgorithm package

Submodules

CBBAlgorithm.CBBAlgorithm module

`CBBAlgorithm.CBBAlgorithm.`**`CBBFractality`**(`graph`)

`CBBAlgorithm.CBBAlgorithm.`**`calculateLb`**(`boxes`)

Module contents

Genetic strategy

Genetic algorithm for multifractal analysis

Genetic

`Genetic.Genetic.`**`Genetic`**(`g`, `minq`, `maxq`, `sizePopulation`, `iterations`, `percentCrossOver`, `percentMutation`, `degreeOfBoring`, `typeAlgorithm`)

The genetic algorithm

Parameters:

- **graph** (*Snap PUN Graph.*) – Network.
- **iterations** (*Integer*) – Number of max iterations
- **sizePopulation** (*Integer*) – Size of population
- **percentCrossOver** (*Double*) – Percent (0 to 1) of individual select to cross
- **percentMutation** (*Double*) – (0 to 1) of probability to apply mutation to an individual
- **degreeOfBoring** (*Integer*) – Number of iterations of boring
- **typeAlgorithm** (*String*) – Method for calculate fractal dimensions: 'SB' for Sandbox, 'BC' for BoxCounting, 'FSBC' for fixed size box counting

Returns: logR: Numpy arraylogarithm of r/d Indexzero: Integerposition of q=0 in Tq and Dq Tq: Numpy arraymass exponents Dq: Numpy arrayfractal dimensions lnMrq: Numpy 2D arraylogarithm of number of nodes in boxes by radio fitNessAverage: Numpy arrayAverage fitness across iterations fitNessMax: Numpy arrayMaximum fitness across iterations fitNessMin: Numpy array:Minimal fitness across iterations

`Genetic.Genetic.`**`calculateCenters`**`(graph, numNodes, iterations, sizePopulation, radius, distances, percentCrossOver, percentMutation, listDegree, maxDegree, degreeOfBoring)`

Calculate centers with a random size between 20% and 90% of number of nodes
The genetic algorithm:

1. Generate a poblation of ramdom nodes as centers of the boxes
2. Evaluate each set of nodes with fitness funtion
3. Categorize poblation according fitness
4. Select two indivudues into population, then create a new individual
5. Remove worst individuals
6. Repeat 2 to 5 ultil a number of iterations or a boring degree

Parameters:

- **graph** (*Snap PUN Graph.*) – Network.
- **numNodes** (*Integer*) – Number of nodes in the network.
- **sizePopulation** (*Integer*) – Size of population
- **radius** (*Integer*) – Diameter of the network
- **distances** (*Numpy 2D array of integers*) – Distance to other nodes
- **percentCrossOver** (*Double*) – Percent (0 to 1) of individual select to cross
- **percentMutation** (*Double*) – (0 to 1) of probability to apply mutation to an individual
- **listDegree** (*Numpy 1D Array*) – List of degree all nodes
- **maxDegree** (*Integer*) – Max degree in the network
- **degreeOfBoring** (*Integer*) – Number of iterations of boring

Returns: best: Numpy arrayBest individual, select centers of boxes Indexzero: Integerposition of q=0 in Tq and Dq Tq: Numpy arraymass exponents Dq: Numpy arrayfractal dimensions lnMrq: Numpy 2D arraylogarithm of number of nodes in boxes by radio fitNessAverage: Numpy arrayAverage fitness across iterations fitNessMax: Numpy arrayMaximum fitness across iterations fitNessMin: Numpy array:Minimal fitness across iterations

`Genetic.Genetic.`**`calculateCentersFixedSize`**`(graph, numNodes, iterations, sizePopulation, radius, distances, percentCrossOver, percentMutation, listDegree, maxDegree, sizeChromosome, degreeOfBoring)`

Calculate centers with a specified size
The genetic algorithm:

1. Generate a poblation of ramdom nodes as centers of the boxes
2. Evaluate each set of nodes with fitness funtion

3. Categorize poblation according fitness
4. Select two indivudues into population, then create a new individual
5. Remove worst individuals
6. Repeat 2 to 5 ultil a number of iterations or a boring degree

Parameters:
- **graph** (*Snap PUN Graph.*) – Network.
- **numNodes** (*Integer*) – Number of nodes in the network.
- **sizePopulation** (*Integer*) – Size of population
- **radius** (*Integer*) – Diameter of the network
- **distances** (*Numpy 2D array of integers*) – Distance to other nodes
- **percentCrossOver** (*Double*) – Percent (0 to 1) of individual select to cross
- **Percent** (*Double*) – (0 to 1) of probability to apply mutation to an individual
- **listDegree** (*Numpy 1D Array*) – List of degree all nodes
- **maxDegree** (*Integer*) – Max degree in the network
- **sizeChromosome** (*Integer*) – Number of centers of boxes
- **degreeOfBoring** (*Integer*) – Number of iterations of boring

Returns: best: Numpy arrayBest individual, select centers of boxes

`Genetic.Genetic.`**`calculateFitness`**`(graph, chromosome, radius, distances, listDegree, maxDegree)`

Calculate fitness from a select node centers in a network

Fitness is the average between distances of the centers and the average the degrees , the centers can be of different size

Parameters:
- **graph** (*Snap PUN Graph.*) – Network.
- **chromosome** (*Numpy array of integers*) – Centers
- **radius** (*Integer*) – Diameter of the network
- **distances** (*Numpy 2D array of integers*) – Distance between all nodes
- **listDegree** (*Numpy 1D Array*) – List of degree all nodes
- **maxDegree** (*Integer*) – Max degree in the network

Returns: Fitness of the centers

Type: Double

Example

```
import sys
import lib.snap as snap
import Genetic.Genetic as Genetic
import numpy

minq = -10
maxq = 10
sizePopulation = 200
percentCrossOver = 0.4
percentMutation = 0.05
degreeOfBoring = 20
Rnd = snap.TRnd(1,0)
graph = snap.GenPrefAttach(10000, 10,Rnd)  #ScaleFree with 10 edges per node

logR, Indexzero,Tq, Dq, lnMrq,fitNessAverage,fitNessMax,fitNessMin = Genetic.Genetic(graph,m
```

SandBox Algorithm

This package provides multifractal analysis with SandBox Algorithm proposed in journal article: Determination of multifractal dimensions of complex networks by means of the sandbox algorithm DOI: 10.1063/1.4907557

SBAlgorithm

`SBAlgorithm.SBAlgorithm.`**`SBAlgorithm`**(`g`, `minq`, `maxq`, `percentSandBox`, `repetitions`, `centerNodes=array([], dtype=float64)`)

Calculate fractal dimension with SandBox method

Inputs are parameters to configure algorithm behaviour.

Parameters:

- **g** (*Snap PUN Graph.*) – Network.
- **minq** – Minimum value of q
- **minq** – Maximum value of q
- **percentSandBox** (*Double*) – Number of combinations of center nodes. This value is a percent of the total nodes
- **repetitions** (*Integer*) – Number of repetitions of algorithm
- **CenterNodes** (*Numpy 1D Array*) – Calculated center. If this is null, then the centers are calculated

Returns: logR: Numpy arraylogarithm of r/d Indexzero: Integerposition of q=0 in Tq and Dq Tq: Numpy arraymass exponents Dq: Numpy arrayfractal dimensions lnMrq: Numpy 2D arraylogarithm of number of nodes in boxes by radio

Example

```
import sys
import lib.snap as snap
import SBAlgorithm.SBAlgorithm as SBAlgorithm
import numpy

minq = -10
maxq = 10
percentOfSandBoxes = 0.6
repetitionsSB = 50
Rnd = snap.TRnd(1,0)
graph = snap.GenPrefAttach(10000, 10,Rnd)  #ScaleFree with 10 edges per node

logRB, IndexzeroB,TqB, DqB, lnMrqB = SBAlgorithm.SBAlgorithm(graph,minq,maxq,percentOfSandBo
```

Simulated Annealing strategy

Simulated annealing algorithm for multifractal analysis

SimulatedAnnealing.SimulatedAnnealing module

`SimulatedAnnealing.SimulatedAnnealing.`**`SA`**(`g`, `minq`, `maxq`, `percentNodes`, `sizePopulation`, `Kmax`, `typeAlgorithm`)

The simulated annealing algorithm

Parameters:
- **graph** (*Snap PUN Graph.*) – Network.
- **sizePopulation** (*Integer*) – Size of population
- **Kmax** (*Integer*) – Initial temperature
- **typeAlgorithm** (*String*) – Method for calculate fractal dimensions: 'SB' for Sandbox, 'BC' for BoxCounting, 'FSBC' for fixed size box counting

Returns: logR: Numpy arraylogarithm of r/d Indexzero: Integerposition of q=0 in Tq and Dq Tq: Numpy arraymass exponents Dq: Numpy arrayfractal dimensions lnMrq: Numpy 2D arraylogarithm of number of nodes in boxes by radio

`SimulatedAnnealing.SimulatedAnnealing.`**`calculateCenters`**(`graph`, `numNodes`, `percentNodes`, `Kmax`, `d`, `distances`, `listID`, `listDegree`, `totalRemoved=0`)

Calculate centers with a specified size

The genetic algorithm:

1. Generate a poblation of ramdom nodes as centers of the boxes
2. Select a neighbor state
3. Select this state according its fitness and global temperature

Parameters:
- **graph** (*Snap PUN Graph.*) – Network.
- **numNodes** (*Integer*) – Number of nodes in the network.
- **percentNodes** (*Integer*) – Size of population
- **Kmax** (*Integer*) – System temperature
- **lisID** (*Numpy 2D array of integers*) – ID of nodes
- **listDegree** (*Numpy 2D array of integers*) – List of degree all nodes
- **totalRemoved** (*Integer*) – Number of nodes in solution, only if you want apply robusness analysis

Returns: currentState: Numpy arraycurrent individual, select centers of boxes

`SimulatedAnnealing.SimulatedAnnealing.`**`calculateFitness`**(`g`, `element`, `radius`, `distances`, `listID`, `listDegree`)

Calculate fitness from a select node centers in a network

Fitness is the average between distances of the centers and the average the degrees , the centers can be of different size

Parameters:
- **graph** (*Snap PUN Graph.*) – Network.
- **chromosome** (*Numpy array of integers*) – Centers
- **radius** (*Integer*) – Diameter of the network
- **distances** (*Numpy 2D array of integers*) – Distance between all nodes
- **listDegree** (*Numpy 1D Array*) – List of degree all nodes
- **maxDegree** (*Integer*) – Max degree in the network

Returns: Fitness of the centers

Type: Double

`SimulatedAnnealing.SimulatedAnnealing.`**`createNeighbors`**(`node`, `numNodes`, `distances`)

Calculate fitness from a select node centers in a network

Fitness is the average between distances of the centers and the average the degrees , the centers can be of different size

Parameters:
- **node** (*Integer.*) – ID of node in the network.
- **numNodes** (*Integer*) – Number of nodes
- **distances** (*Numpy 2D array of integers*) – Distance between all nodes

Returns: neighbors. Numpy ID array

Type: Double

Example

```
import sys
import lib.snap as snap
import SimulatedAnnealing.SimulatedAnnealing as SimulatedAnnealing
import numpy

minq = -10
maxq = 10
sizePopulation = 200
Kmax = 1500
Rnd = snap.TRnd(1,0)
graph = snap.GenPrefAttach(10000, 10,Rnd)  #ScaleFree with 10 edges per node

logRD, IndexzeroD,TqD, DqD, lnMrqD = SimulatedAnnealing.SA(graph,minq,maxq,percentOfSandBoxe
```

robustness package

Submodules

robustness.robustness module

robustness.robustness.**robustness_analysis** (graph, typeRemoval, minq, maxq, percentSandBox, repetitions, temperature=0, sizePopulation=0, iterationsGenetic=0, percentCrossOver=0, percentMutation=0, degreeOfBoring=0, percentOfNodes=0.1, initialPercent=0.1, finalPercent=1.0, iteracionPercent=0.1, nameFile='none')

Module contents

utils package

Submodules

utils.utils module

utils.utils.**copyGraph** (graph)

utils.utils.**getAdjacenceMatriz** (distances, numNodes)

utils.utils.**getAveragePathLength** (graph)

utils.utils.**getDistancesMatrix** (graph, numNodes, listID)

utils.utils.**getOrderedClosenessCentrality** (graph, N)

utils.utils.**getSizeOfGiantComponent** (graph)

utils.utils.**linealRegresssion** (x, y)

utils.utils.**removeNodes** (graph, typeRemoval, p, numberNodesToRemove, ClosenessCentrality, listID, nodesToRemove=array([], dtype=float64))

Module contents

Requeriments

- numpy >= 1.12.1
- snap >= 0.5
- matplotlib >= 2.0
- python = 2.7

Indices and tables

- **`genindex`**
- **`modindex`**
- **`search`**

Index

B

BCAlgorithm() (in module BCAlgorithm.BCAlgorithm)

BCAlgorithm.BCAlgorithm (module)

C

calculateCenters() (in module Genetic.Genetic)

(in module SimulatedAnnealing.SimulatedAnnealing)

calculateCentersFixedSize() (in module Genetic.Genetic)

calculateFitness() (in module Genetic.Genetic)

(in module SimulatedAnnealing.SimulatedAnnealing)

calculateLb() (in module CBBAlgorithm.CBBAlgorithm)

CBBAlgorithm (module)

CBBAlgorithm.CBBAlgorithm (module)

CBBFractality() (in module CBBAlgorithm.CBBAlgorithm)

copyGraph() (in module utils.utils)

createNeighbors() (in module SimulatedAnnealing.SimulatedAnnealing)

F

FSBCAlgorithm() (in module FSBCAlgorithm.FSBCAlgorithm)

FSBCAlgorithm.FSBCAlgorithm (module)

G

Genetic() (in module Genetic.Genetic)

Genetic.Genetic (module)

getAdjacenceMatriz() (in module utils.utils)

getAveragePathLength() (in module utils.utils)

getDistancesMatrix() (in module utils.utils)

getOrderedClosenessCentrality() (in module utils.utils)

getSizeOfGiantComponent() (in module utils.utils)

L

linealRegresssion() (in module utils.utils)

R

removeNodes() (in module utils.utils)

robustness (module)

robustness.robustness (module)

robustness_analysis() (in module robustness.robustness)

S

SA() (in module SimulatedAnnealing.SimulatedAnnealing)

SBAlgorithm() (in module SBAlgorithm.SBAlgorithm)

SBAlgorithm.SBAlgorithm (module)

SimulatedAnnealing.SimulatedAnnealing (module)

U

utils (module)

utils.utils (module)

Python Module Index

b

BCAlgorithm

BCAlgorithm.BCAlgorithm

c

CBBAlgorithm

CBBAlgorithm.CBBAlgorithm

f

FSBCAlgorithm

FSBCAlgorithm.FSBCAlgorithm

g

Genetic

Genetic.Genetic

r

robustness

robustness.robustness

s

SBAlgorithm

SBAlgorithm.SBAlgorithm

SimulatedAnnealing

SimulatedAnnealing.SimulatedAnnealing

u

utils

utils.utils