giddy Documentation

Release 2.2.0

pysal developers

Jun 20, 2019

CONTENTS:

Giddy is an open-source python library for the analysis of dynamics of longitudinal spatial data. Originating from the spatial dynamics module in PySAL (Python Spatial Analysis Library), it is under active development for the inclusion of many newly proposed analytics that consider the role of space in the evolution of distributions over time and has several new features including inter- and intra-regional decomposition of mobility association and local measures of exchange mobility in addition to space-time LISA and spatial markov methods.

CHAPTER

ONE

INSTALLATION

From version 2.2.0, giddy supports python 3.6 and 3.7 only. Please make sure that you are operating in a python 3 environment.

1.1 Installing released version

giddy is available on the Python Package Index. Therefore, you can either install directly with *pip* from the command line:

```
pip install -U giddy
```

or download the source distribution (.tar.gz) and decompress it to your selected destination. Open a command shell and navigate to the decompressed folder. Type:

```
pip install .
```

You may also install the latest stable giddy via conda-forge channel by running:

```
$ conda install --channel conda-forge giddy
```

1.2 Installing development version

Potentially, you might want to use the newest features in the development version of giddy on github - pysal/giddy while have not been incorporated in the Pypi released version. You can achieve that by installing pysal/giddy by running the following from a command shell:

```
pip install git+https://github.com/pysal/giddy.git
```

You can also fork the pysal/giddy repo and create a local clone of your fork. By making changes to your local clone and submitting a pull request to pysal/giddy, you can contribute to the giddy development.

CHAPTER

TWO

API REFERENCE

2.1 Markov Methods

giddy.markov.Markov(class_ids[, classes])	Classic Markov transition matrices.
giddy.markov.Spatial_Markov(y, w[, k, m, ...])	Markov transitions conditioned on the value of the spatial lag.
giddy.markov.LISA_Markov(y, w[, ...])	Markov for Local Indicators of Spatial Association
giddy.markov.FullRank_Markov(y)	Full Rank Markov in which ranks are considered as Markov states rather than quantiles or other discretized classes.
giddy.markov.GeoRank_Markov(y)	Geographic Rank Markov.
giddy.markov.kullback(F)	Kullback information based test of Markov Homogeneity.
giddy.markov.prais(pmat)	Prais conditional mobility measure.
giddy.markov.homogeneity(transition_matrices)	Test for homogeneity of Markov transition probabilities across regimes.
giddy.markov.sojourn_time(p)	Calculate sojourn time based on a given transition probability matrix.
giddy.ergodic.steady_state(P)	Calculates the steady state probability vector for a regular Markov transition matrix P.
giddy.ergodic.fmpt(P)	Calculates the matrix of first mean passage times for an ergodic transition probability matrix.
giddy.ergodic.var_fmpt(P)	Variances of first mean passage times for an ergodic transition probability matrix.

2.1.1 giddy.markov.Markov

class giddy.markov.**Markov**(*class_ids*, *classes=None*)

Classic Markov transition matrices.

Parameters

class_ids [array] (n, t), one row per observation, one column recording the state of each observation, with as many columns as time periods.

classes [array] (k, 1), all different classes (bins) of the matrix.

Examples

```
>>> import numpy as np
>>> from giddy.markov import Markov
>>> c = [['b','a','c'],['c','c','a'],['c','b','c']]
>>> c.extend([['a','a','b'], ['a','b','c']])
>>> c = np.array(c)
>>> m = Markov(c)
>>> m.classes.tolist()
['a', 'b', 'c']
>>> m.p
array([[0.25      , 0.5       , 0.25      ],
       [0.33333333, 0.        , 0.66666667],
       [0.33333333, 0.33333333, 0.33333333]])
>>> m.steady_state
array([0.30769231, 0.28846154, 0.40384615])
```

US nominal per capita income 48 states 81 years 1929-2009

```
>>> import libpysal
>>> import mapclassify as mc
>>> f = libpysal.io.open(libpysal.examples.get_path("usjoin.csv"))
>>> pci = np.array([f.by_col[str(y)] for y in range(1929,2010)])
```

set classes to quintiles for each year

```
>>> q5 = np.array([mc.Quantiles(y).yb for y in pci]).transpose()
>>> m = Markov(q5)
>>> m.transitions
array([[729.,  71.,   1.,   0.,   0.],
       [ 72., 567.,  80.,   3.,   0.],
       [  0.,  81., 631.,  86.,   2.],
       [  0.,   3.,  86., 573.,  56.],
       [  0.,   0.,   1.,  57., 741.]])
>>> m.p
array([[0.91011236, 0.0886392 , 0.00124844, 0.        , 0.        ],
       [0.09972299, 0.78531856, 0.11080332, 0.00415512, 0.        ],
       [0.        , 0.10125   , 0.78875   , 0.1075    , 0.0025    ],
       [0.        , 0.00417827, 0.11977716, 0.79805014, 0.07799443],
       [0.        , 0.        , 0.00125156, 0.07133917, 0.92740926]])
>>> m.steady_state
array([0.20774716, 0.18725774, 0.20740537, 0.18821787, 0.20937187])
```

Relative incomes

```
>>> pci = pci.transpose()
>>> rpci = pci/(pci.mean(axis=0))
>>> rq = mc.Quantiles(rpci.flatten()).yb.reshape(pci.shape)
>>> mq = Markov(rq)
>>> mq.transitions
array([[707.,  58.,   7.,   1.,   0.],
       [ 50., 629.,  80.,   1.,   1.],
       [  4.,  79., 610.,  73.,   2.],
       [  0.,   7.,  72., 650.,  37.],
       [  0.,   0.,   0.,  48., 724.]])
>>> mq.steady_state
array([0.17957376, 0.21631443, 0.21499942, 0.21134662, 0.17776576])
```

Attributes

p [array] (k, k), transition probability matrix.

steady_state [array] (k,), ergodic distribution.

transitions [array] (k, k), count of transitions between each state i and j.

__init__(*self*, *class_ids*, *classes=None*)
Initialize self. See help(type(self)) for accurate signature.

2.1.2 giddy.markov.Spatial_Markov

class giddy.markov.**Spatial_Markov**(*y*, *w*, *k=4*, *m=4*, *permutations=0*, *fixed=True*, *discrete=False*, *cutoffs=None*, *lag_cutoffs=None*, *variable_name=None*)
Markov transitions conditioned on the value of the spatial lag.

Parameters

y [array] (n, t), one row per observation, one column per state of each observation, with as many columns as time periods.

w [W] spatial weights object.

k [integer, optional] number of classes (quantiles) for input time series y. Default is 4. If discrete=True, k is determined endogenously.

m [integer, optional] number of classes (quantiles) for the spatial lags of regional time series. Default is 4. If discrete=True, m is determined endogenously.

permutations [int, optional] number of permutations for use in randomization based inference (the default is 0).

fixed [bool, optional] If true, discretization are taken over the entire n*t pooled series and cutoffs can be user-defined. If cutoffs and lag_cutoffs are not given, quantiles are used. If false, quantiles are taken each time period over n. Default is True.

discrete [bool, optional] If true, categorical spatial lags which are most common categories of neighboring observations serve as the conditioning and fixed is ignored; if false, weighted averages of neighboring observations are used. Default is false.

cutoffs [array, optional] users can specify the discretization cutoffs for continuous time series. Default is None, meaning that quantiles will be used for the discretization.

lag_cutoffs [array, optional] users can specify the discretization cutoffs for the spatial lags of continuous time series. Default is None, meaning that quantiles will be used for the discretization.

variable_name [string] name of variable.

Notes

Based on [Rey01].

The shtest and chi2 tests should be used with caution as they are based on classic theory assuming random transitions. The x2 based test is preferable since it simulates the randomness under the null. It is an experimental test requiring further analysis.

Examples

```
>>> import libpysal
>>> from giddy.markov import Spatial_Markov
>>> import numpy as np
>>> f = libpysal.io.open(libpysal.examples.get_path("usjoin.csv"))
>>> pci = np.array([f.by_col[str(y)] for y in range(1929,2010)])
>>> pci = pci.transpose()
>>> rpci = pci/(pci.mean(axis=0))
>>> w = libpysal.io.open(libpysal.examples.get_path("states48.gal")).read()
>>> w.transform = 'r'
```

Now we create a *Spatial_Markov* instance for the continuous relative per capita income time series for 48 US lower states 1929-2009. The current implementation allows users to classify the continuous incomes in a more flexible way.

(1) Global quintiles to discretize the income data (k=5), and global quintiles to discretize the spatial lags of incomes (m=5).

```
>>> sm = Spatial_Markov(rpci, w, fixed=True, k=5, m=5, variable_name='rpci')
```

We can examine the cutoffs for the incomes and cutoffs for the spatial lags

```
>>> sm.cutoffs
array([0.83999133, 0.94707545, 1.03242697, 1.14911154])
>>> sm.lag_cutoffs
array([0.88973386, 0.95891917, 1.01469758, 1.1183566 ])
```

Obviously, they are slightly different.

We now look at the estimated spatially lag conditioned transition probability matrices.

```
>>> for p in sm.P:
...     print(p)
[[0.96341463 0.0304878  0.00609756 0.         0.        ]
 [0.06040268 0.83221477 0.10738255 0.         0.        ]
 [0.         0.14       0.74       0.12       0.        ]
 [0.         0.03571429 0.32142857 0.57142857 0.07142857]
 [0.         0.         0.         0.16666667 0.83333333]]
[[0.79831933 0.16806723 0.03361345 0.         0.        ]
 [0.0754717  0.88207547 0.04245283 0.         0.        ]
 [0.00537634 0.06989247 0.8655914  0.05913978 0.        ]
 [0.         0.         0.06372549 0.90196078 0.03431373]
 [0.         0.         0.         0.19444444 0.80555556]]
[[0.84693878 0.15306122 0.         0.         0.        ]
 [0.08133971 0.78947368 0.1291866  0.         0.        ]
 [0.00518135 0.0984456  0.79274611 0.0984456  0.00518135]
 [0.         0.         0.09411765 0.87058824 0.03529412]
 [0.         0.         0.         0.10204082 0.89795918]]
[[0.8852459  0.09836066 0.         0.01639344 0.        ]
 [0.03875969 0.81395349 0.13953488 0.         0.00775194]
 [0.0049505  0.09405941 0.77722772 0.11881188 0.0049505 ]
 [0.         0.02339181 0.12865497 0.75438596 0.09356725]
 [0.         0.         0.         0.09661836 0.90338164]]
[[0.33333333 0.66666667 0.         0.         0.        ]
 [0.0483871  0.77419355 0.16129032 0.01612903 0.        ]
 [0.01149425 0.16091954 0.74712644 0.08045977 0.        ]
```

(continues on next page)

(continued from previous page)

```
 [0.         0.01036269 0.06217617 0.89637306 0.03108808]
 [0.         0.         0.         0.02352941 0.97647059]]
```

The probability of a poor state remaining poor is 0.963 if their neighbors are in the 1st quintile and 0.798 if their neighbors are in the 2nd quintile. The probability of a rich economy remaining rich is 0.976 if their neighbors are in the 5th quintile, but if their neighbors are in the 4th quintile this drops to 0.903.

The global transition probability matrix is estimated:

```
>>> print(sm.p)
[[0.91461837 0.07503234 0.00905563 0.00129366 0.        ]
 [0.06570302 0.82654402 0.10512484 0.00131406 0.00131406]
 [0.00520833 0.10286458 0.79427083 0.09505208 0.00260417]
 [0.         0.00913838 0.09399478 0.84856397 0.04830287]
 [0.         0.         0.         0.06217617 0.93782383]]
```

The Q and likelihood ratio statistics are both significant indicating the dynamics are not homogeneous across the lag classes:

```
>>> "%.3f"%sm.LR
'170.659'
>>> "%.3f"%sm.Q
'200.624'
>>> "%.3f"%sm.LR_p_value
'0.000'
>>> "%.3f"%sm.Q_p_value
'0.000'
>>> sm.dof_hom
60
```

The long run distribution for states with poor (rich) neighbors has 0.435 (0.018) of the values in the first quintile, 0.263 (0.200) in the second quintile, 0.204 (0.190) in the third, 0.0684 (0.255) in the fourth and 0.029 (0.337) in the fifth quintile.

```
>>> sm.S
array([[0.43509425, 0.2635327 , 0.20363044, 0.06841983, 0.02932278],
       [0.13391287, 0.33993305, 0.25153036, 0.23343016, 0.04119356],
       [0.12124869, 0.21137444, 0.2635101 , 0.29013417, 0.1137326 ],
       [0.0776413 , 0.19748806, 0.25352636, 0.22480415, 0.24654013],
       [0.01776781, 0.19964349, 0.19009833, 0.25524697, 0.3372434 ]])
```

States with incomes in the first quintile with neighbors in the first quintile return to the first quartile after 2.298 years, after leaving the first quintile. They enter the fourth quintile after 80.810 years after leaving the first quintile, on average. Poor states within neighbors in the fourth quintile return to the first quintile, on average, after 12.88 years, and would enter the fourth quintile after 28.473 years.

```
>>> for f in sm.F:
...     print(f)
...
[[  2.29835259  28.95614035  46.14285714  80.80952381 279.42857143]
 [ 33.86549708   3.79459555  22.57142857  57.23809524 255.85714286]
 [ 43.60233918   9.73684211   4.91085714  34.66666667 233.28571429]
 [ 46.62865497  12.76315789   6.25714286  14.61564626 198.61904762]
 [ 52.62865497  18.76315789  12.25714286   6.          34.1031746 ]]
[[  7.46754205   9.70574606  25.76785714  74.53116883 194.23446197]
 [ 27.76691978   2.94175577  24.97142857  73.73474026 193.4380334 ]
```

(continues on next page)

(continued from previous page)

```
 [ 53.57477715  28.48447637   3.97566318  48.76331169 168.46660482]
 [ 72.03631562  46.94601483  18.46153846   4.28393653 119.70329314]
 [ 77.17917276  52.08887197  23.6043956    5.14285714  24.27564033]]
[[  8.24751154   6.53333333  18.38765432  40.70864198 112.76732026]
 [ 47.35040872   4.73094099  11.85432099  34.17530864 106.23398693]
 [ 69.42288828  24.76666667   3.794921    22.32098765  94.37966594]
 [ 83.72288828  39.06666667  14.3          3.44668119  76.36702977]
 [ 93.52288828  48.86666667  24.1          9.8          8.79255406]]
[[ 12.87974382  13.34847151  19.83446328  28.47257282  55.82395142]
 [ 99.46114206   5.06359731  10.54545198  23.05133495  49.68944423]
 [117.76777159  23.03735526   3.94436301  15.0843986   43.57927247]
 [127.89752089  32.4393006   14.56853107   4.44831643  31.63099455]
 [138.24752089  42.7893006   24.91853107  10.35         4.05613474]]
[[ 56.2815534    1.5         10.57236842  27.02173913 110.54347826]
 [ 82.9223301    5.00892857   9.07236842  25.52173913 109.04347826]
 [ 97.17718447  19.53125      5.26043557  21.42391304 104.94565217]
 [127.1407767   48.74107143  33.29605263   3.91777427  83.52173913]
 [169.6407767   91.24107143  75.79605263  42.5          2.96521739]]
```

(2) Global quintiles to discretize the income data (k=5), and global quartiles to discretize the spatial lags of incomes (m=4).

```
>>> sm = Spatial_Markov(rpci, w, fixed=True, k=5, m=4, variable_name='rpci')
```

We can also examine the cutoffs for the incomes and cutoffs for the spatial lags:

```
>>> sm.cutoffs
array([0.83999133, 0.94707545, 1.03242697, 1.14911154])
>>> sm.lag_cutoffs
array([0.91440247, 0.98583079, 1.08698351])
```

We now look at the estimated spatially lag conditioned transition probability matrices.

```
>>> for p in sm.P:
...     print(p)
[[0.95708955 0.03544776 0.00746269 0.         0.        ]
 [0.05825243 0.83980583 0.10194175 0.         0.        ]
 [0.         0.1294964  0.76258993 0.10791367 0.        ]
 [0.         0.01538462 0.18461538 0.72307692 0.07692308]
 [0.         0.         0.         0.14285714 0.85714286]]
[[0.7421875  0.234375   0.0234375  0.         0.        ]
 [0.08550186 0.85130112 0.06319703 0.         0.        ]
 [0.00865801 0.06926407 0.86147186 0.05627706 0.004329  ]
 [0.         0.         0.05363985 0.92337165 0.02298851]
 [0.         0.         0.         0.13432836 0.86567164]]
[[0.95145631 0.04854369 0.         0.         0.        ]
 [0.06       0.79       0.145      0.         0.005     ]
 [0.00358423 0.10394265 0.7921147  0.09677419 0.00358423]
 [0.         0.01630435 0.13586957 0.75543478 0.0923913 ]
 [0.         0.         0.         0.10204082 0.89795918]]
[[0.16666667 0.66666667 0.         0.16666667 0.        ]
 [0.03488372 0.80232558 0.15116279 0.01162791 0.        ]
 [0.00840336 0.13445378 0.70588235 0.1512605  0.        ]
 [0.         0.01171875 0.08203125 0.87109375 0.03515625]
 [0.         0.         0.         0.03434343 0.96565657]]
```

We now obtain 4 5*5 spatial lag conditioned transition probability matrices instead of 5 as in case (1).

The Q and likelihood ratio statistics are still both significant.

```
>>> "%.3f"%sm.LR
'172.105'
>>> "%.3f"%sm.Q
'321.128'
>>> "%.3f"%sm.LR_p_value
'0.000'
>>> "%.3f"%sm.Q_p_value
'0.000'
>>> sm.dof_hom
45
```

(3) We can also set the cutoffs for relative incomes and their spatial lags manually. For example, we want the defining cutoffs to be [0.8, 0.9, 1, 1.2], meaning that relative incomes: 2.1 smaller than 0.8 : class 0 2.2 between 0.8 and 0.9: class 1 2.3 between 0.9 and 1.0 : class 2 2.4 between 1.0 and 1.2: class 3 2.5 larger than 1.2: class 4

```
>>> cc = np.array([0.8, 0.9, 1, 1.2])
>>> sm = Spatial_Markov(rpci, w, cutoffs=cc, lag_cutoffs=cc, variable_name='rpci')
>>> sm.cutoffs
array([0.8, 0.9, 1. , 1.2])
>>> sm.k
5
>>> sm.lag_cutoffs
array([0.8, 0.9, 1. , 1.2])
>>> sm.m
5
>>> for p in sm.P:
...     print(p)
[[0.96703297 0.03296703 0.         0.         0.        ]
 [0.10638298 0.68085106 0.21276596 0.         0.        ]
 [0.         0.14285714 0.7755102  0.08163265 0.        ]
 [0.         0.         0.5        0.5        0.        ]
 [0.         0.         0.         0.         0.        ]]
[[0.88636364 0.10606061 0.00757576 0.         0.        ]
 [0.04402516 0.89308176 0.06289308 0.         0.        ]
 [0.         0.05882353 0.8627451  0.07843137 0.        ]
 [0.         0.         0.13846154 0.86153846 0.        ]
 [0.         0.         0.         0.         1.        ]]
[[0.78082192 0.17808219 0.02739726 0.01369863 0.        ]
 [0.03488372 0.90406977 0.05813953 0.00290698 0.        ]
 [0.         0.05919003 0.84735202 0.09034268 0.00311526]
 [0.         0.         0.05811623 0.92985972 0.01202405]
 [0.         0.         0.         0.14285714 0.85714286]]
[[0.82692308 0.15384615 0.         0.01923077 0.        ]
 [0.0703125  0.7890625  0.125      0.015625   0.        ]
 [0.00295858 0.06213018 0.82248521 0.10946746 0.00295858]
 [0.         0.00185529 0.07606679 0.88497217 0.03710575]
 [0.         0.         0.         0.07803468 0.92196532]]
[[0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.        ]
 [0.         0.06666667 0.9        0.03333333 0.        ]
 [0.         0.         0.05660377 0.90566038 0.03773585]
 [0.         0.         0.         0.03932584 0.96067416]]
```

(4) Spatial_Markov also accept discrete time series and calculate categorical spatial lags on which several transition probability matrices are conditioned. Let's still use the US state income time series to demonstrate. We first discretize them into categories and then pass them to Spatial_Markov.

```
>>> import mapclassify as mc
>>> y = mc.Quantiles(rpci.flatten(), k=5).yb.reshape(rpci.shape)
>>> np.random.seed(5)
>>> sm = Spatial_Markov(y, w, discrete=True, variable_name='discretized rpci')
>>> sm.k
5
>>> sm.m
5
>>> for p in sm.P:
...     print(p)
[[0.94787645 0.04440154 0.00772201 0.         0.        ]
 [0.08333333 0.81060606 0.10606061 0.         0.        ]
 [0.         0.12765957 0.79787234 0.07446809 0.        ]
 [0.         0.02777778 0.22222222 0.66666667 0.08333333]
 [0.         0.         0.         0.33333333 0.66666667]]
[[0.888      0.096      0.016      0.         0.        ]
 [0.06049822 0.84341637 0.09608541 0.         0.        ]
 [0.00666667 0.10666667 0.81333333 0.07333333 0.        ]
 [0.         0.         0.08527132 0.86821705 0.04651163]
 [0.         0.         0.         0.10204082 0.89795918]]
[[0.65217391 0.32608696 0.02173913 0.         0.        ]
 [0.07446809 0.80851064 0.11170213 0.         0.00531915]
 [0.01071429 0.1        0.76428571 0.11785714 0.00714286]
 [0.         0.00552486 0.09392265 0.86187845 0.03867403]
 [0.         0.         0.         0.13157895 0.86842105]]
[[0.91935484 0.06451613 0.         0.01612903 0.        ]
 [0.06796117 0.90291262 0.02912621 0.         0.        ]
 [0.         0.05755396 0.87769784 0.0647482  0.        ]
 [0.         0.02150538 0.10752688 0.80107527 0.06989247]
 [0.         0.         0.         0.08064516 0.91935484]]
[[0.81818182 0.18181818 0.         0.         0.        ]
 [0.01754386 0.70175439 0.26315789 0.01754386 0.        ]
 [0.         0.14285714 0.73333333 0.12380952 0.        ]
 [0.         0.0042735  0.06837607 0.89316239 0.03418803]
 [0.         0.         0.         0.03891051 0.96108949]]
```

Attributes

class_ids [array] (n, t), discretized series if y is continuous. Otherwise it is identical to y.

classes [array] (k, 1), all different classes (bins).

lclass_ids [array] (n, t), spatial lag series.

lclasses [array] (k, 1), all different classes (bins) for spatial lags.

p [array] (k, k), transition probability matrix for a-spatial Markov.

s [array] (k, 1), ergodic distribution for a-spatial Markov.

transitions [array] (k, k), counts of transitions between each state i and j for a-spatial Markov.

T [array] (k, k, k), counts of transitions for each conditional Markov. T[0] is the matrix of transitions for observations with lags in the 0th quantile; T[k-1] is the transitions for the observations with lags in the k-1th.

P [array] (k, k, k), transition probability matrix for spatial Markov first dimension is the conditioned on the lag.

S [array] (k, k), steady state distributions for spatial Markov. Each row is a conditional steady_state.

F [array] (k, k, k),first mean passage times. First dimension is conditioned on the lag.

shtest [list] (k elements), each element of the list is a tuple for a multinomial difference test between the steady state distribution from a conditional distribution versus the overall steady state distribution: first element of the tuple is the chi2 value, second its p-value and the third the degrees of freedom.

chi2 [list] (k elements), each element of the list is a tuple for a chi-squared test of the difference between the conditional transition matrix against the overall transition matrix: first element of the tuple is the chi2 value, second its p-value and the third the degrees of freedom.

x2 [float] sum of the chi2 values for each of the conditional tests. Has an asymptotic chi2 distribution with k(k-1)(k-1) degrees of freedom. Under the null that transition probabilities are spatially homogeneous. (see chi2 above)

x2_dof [int] degrees of freedom for homogeneity test.

x2_pvalue [float] pvalue for homogeneity test based on analytic. distribution

x2_rpvalue [float] (if permutations>0) pseudo p-value for x2 based on random spatial permutations of the rows of the original transitions.

x2_realizations [array] (permutations,1), the values of x2 for the random permutations.

Q [float] Chi-square test of homogeneity across lag classes based on [BB03].

Q_p_value [float] p-value for Q.

LR [float] Likelihood ratio statistic for homogeneity across lag classes based on [BB03].

LR_p_value [float] p-value for LR.

dof_hom [int] degrees of freedom for LR and Q, corrected for 0 cells.

Methods

summary(self[, file_name])	A summary method to call the Markov homogeneity test to test for temporally lagged spatial dependence.

giddy.markov.Spatial_Markov

Spatial_Markov.**summary**(*self*, *file_name=None*)

A summary method to call the Markov homogeneity test to test for temporally lagged spatial dependence.

To learn more about the properties of the tests, refer to [RKW16] and [KR18].

__init__(*self*, *y*, *w*, *k=4*, *m=4*, *permutations=0*, *fixed=True*, *discrete=False*, *cutoffs=None*, *lag_cutoffs=None*, *variable_name=None*)

Initialize self. See help(type(self)) for accurate signature.

2.1.3 giddy.markov.LISA_Markov

class giddy.markov.**LISA_Markov**(*y*, *w*, *permutations=0*, *significance_level=0.05*, *geoda_quads=False*)

Markov for Local Indicators of Spatial Association

Parameters

y [array] (n, t), n cross-sectional units observed over t time periods.

w [W] spatial weights object.

permutations [int, optional] number of permutations used to determine LISA significance (the default is 0).

significance_level [float, optional] significance level (two-sided) for filtering significant LISA endpoints in a transition (the default is 0.05).

geoda_quads [bool] If True use GeoDa scheme: HH=1, LL=2, LH=3, HL=4. If False use PySAL Scheme: HH=1, LH=2, LL=3, HL=4. (the default is False).

Examples

```
>>> import libpysal
>>> import numpy as np
>>> from giddy.markov import LISA_Markov
>>> f = libpysal.io.open(libpysal.examples.get_path("usjoin.csv"))
>>> years = list(range(1929, 2010))
>>> pci = np.array([f.by_col[str(y)] for y in years]).transpose()
>>> w = libpysal.io.open(libpysal.examples.get_path("states48.gal")).read()
>>> lm = LISA_Markov(pci,w)
>>> lm.classes
array([1, 2, 3, 4])
>>> lm.steady_state
array([0.28561505, 0.14190226, 0.40493672, 0.16754598])
>>> lm.transitions
array([[1.087e+03, 4.400e+01, 4.000e+00, 3.400e+01],
       [4.100e+01, 4.700e+02, 3.600e+01, 1.000e+00],
       [5.000e+00, 3.400e+01, 1.422e+03, 3.900e+01],
       [3.000e+01, 1.000e+00, 4.000e+01, 5.520e+02]])
>>> lm.p
array([[0.92985458, 0.03763901, 0.00342173, 0.02908469],
       [0.07481752, 0.85766423, 0.06569343, 0.00182482],
       [0.00333333, 0.02266667, 0.948     , 0.026     ],
       [0.04815409, 0.00160514, 0.06420546, 0.88603531]])
>>> lm.move_types[0,:3]
array([11, 11, 11])
>>> lm.move_types[0,-3:]
array([11, 11, 11])
```

Now consider only moves with one, or both, of the LISA end points being significant

```
>>> np.random.seed(10)
>>> lm_random = LISA_Markov(pci, w, permutations=99)
>>> lm_random.significant_moves[0, :3]
array([11, 11, 11])
>>> lm_random.significant_moves[0,-3:]
array([59, 43, 27])
```

Any value less than 49 indicates at least one of the LISA end points was significant. So for example, the first spatial unit experienced a transition of type 11 (LL, LL) during the first three and last tree intervals (according to lm.move_types), however, the last three of these transitions involved insignificant LISAS in both the start and ending year of each transition.

Test whether the moves of y are independent of the moves of wy

```
>>> "Chi2: %8.3f, p: %5.2f, dof: %d" % lm.chi_2
'Chi2: 1058.208, p:  0.00, dof: 9'
```

Actual transitions of LISAs

```
>>> lm.transitions
array([[1.087e+03, 4.400e+01, 4.000e+00, 3.400e+01],
       [4.100e+01, 4.700e+02, 3.600e+01, 1.000e+00],
       [5.000e+00, 3.400e+01, 1.422e+03, 3.900e+01],
       [3.000e+01, 1.000e+00, 4.000e+01, 5.520e+02]])
```

Expected transitions of LISAs under the null y and wy are moving independently of one another

```
>>> lm.expected_t
array([[1.12328098e+03, 1.15377356e+01, 3.47522158e-01, 3.38337644e+01],
       [3.50272664e+00, 5.28473882e+02, 1.59178880e+01, 1.05503814e-01],
       [1.53878082e-01, 2.32163556e+01, 1.46690710e+03, 9.72266513e+00],
       [9.60775143e+00, 9.86856346e-02, 6.23537392e+00, 6.07058189e+02]])
```

If the LISA classes are to be defined according to GeoDa, the *geoda_quad* option has to be set to true

```
>>> lm.q[0:5,0]
array([3, 2, 3, 1, 4])
>>> lm = LISA_Markov(pci,w, geoda_quads=True)
>>> lm.q[0:5,0]
array([2, 3, 2, 1, 4])
```

Attributes

chi_2 [tuple] (3 elements) (chi square test statistic, p-value, degrees of freedom) for test that dynamics of y are independent of dynamics of wy.

classes [array] (4, 1) 1=HH, 2=LH, 3=LL, 4=HL (own, lag) 1=HH, 2=LL, 3=LH, 4=HL (own, lag) (if geoda_quads=True)

expected_t [array] (4, 4), expected number of transitions under the null that dynamics of y are independent of dynamics of wy.

move_types [matrix] (n, t-1), integer values indicating which type of LISA transition occurred (q1 is quadrant in period 1, q2 is quadrant in period 2).

q1	q2	move_type
1	1	1
1	2	2
1	3	3
1	4	4
2	1	5
2	2	6
2	3	7
2	4	8
3	1	9
3	2	10
3	3	11
3	4	12
4	1	13
4	2	14
4	3	15
4	4	16

p [array] (k, k), transition probability matrix.

p_values [matrix] (n, t), LISA p-values for each end point (if permutations > 0).

significant_moves [matrix] (n, t-1), integer values indicating the type and significance of a LISA transition. st = 1 if significant in period t, else st=0 (if permutations > 0).

(s1,s2)	move_type
(1,1)	[1, 16]
(1,0)	[17, 32]
(0,1)	[33, 48]
(0,0)	[49, 64]

q1	q2	s1	s2	move_type
1	1	1	1	1
1	2	1	1	2
1	3	1	1	3
1	4	1	1	4
2	1	1	1	5
2	2	1	1	6
2	3	1	1	7
2	4	1	1	8
3	1	1	1	9
3	2	1	1	10
3	3	1	1	11
3	4	1	1	12
4	1	1	1	13
4	2	1	1	14
4	3	1	1	15
4	4	1	1	16
1	1	1	0	17
1	2	1	0	18
.	.	.	.	.
.	.	.	.	.
4	3	1	0	31
4	4	1	0	32
1	1	0	1	33
1	2	0	1	34
.	.	.	.	.
.	.	.	.	.
4	3	0	1	47
4	4	0	1	48
1	1	0	0	49
1	2	0	0	50
.	.	.	.	.
.	.	.	.	.
4	3	0	0	63
4	4	0	0	64

steady_state [array] (k,), ergodic distribution.

transitions [array] (4, 4), count of transitions between each state i and j.

spillover [array] Detect spillover locations for diffusion in LISA Markov.

Methods

spillover(self[, quadrant, neighbors_on])	Detect spillover locations for diffusion in LISA Markov.

giddy.markov.LISA_Markov

LISA_Markov.**spillover**(*self*, *quadrant=1*, *neighbors_on=False*)

Detect spillover locations for diffusion in LISA Markov.

Parameters

quadrant [int] which quadrant in the scatterplot should form the core of a cluster.

neighbors_on [binary] If false, then only the 1st order neighbors of a core location are included in the cluster. If true, neighbors of cluster core 1st order neighbors are included in the cluster.

Returns

results [dictionary] two keys - values pairs: 'components' - array (n, t) values are integer ids (starting at 1) indicating which component/cluster observation i in period t belonged to. 'spillover' - array (n, t-1) binary values indicating if the location was a spill-over location that became a new member of a previously existing cluster.

Examples

```
>>> import libpysal
>>> from giddy.markov import LISA_Markov
>>> f = libpysal.io.open(libpysal.examples.get_path("usjoin.csv"))
>>> years = list(range(1929, 2010))
>>> pci = np.array([f.by_col[str(y)] for y in years]).transpose()
>>> w = libpysal.io.open(libpysal.examples.get_path("states48.gal")).read()
>>> np.random.seed(10)
>>> lm_random = LISA_Markov(pci, w, permutations=99)
>>> r = lm_random.spillover()
>>> (r['components'][:, 12] > 0).sum()
17
>>> (r['components'][:, 13]>0).sum()
23
>>> (r['spill_over'][:,12]>0).sum()
6
```

Including neighbors of core neighbors >>> rn = lm_random.spillover(neighbors_on=True) >>> (rn['components'][:, 12] > 0).sum() 26 >>> (rn["components"][:, 13] > 0).sum() 34 >>> (rn["spill_over"][:, 12] > 0).sum() 8

__init__(*self*, *y*, *w*, *permutations=0*, *significance_level=0.05*, *geoda_quads=False*)

Initialize self. See help(type(self)) for accurate signature.

2.1.4 giddy.markov.FullRank_Markov

class giddy.markov.**FullRank_Markov**(*y*)

Full Rank Markov in which ranks are considered as Markov states rather than quantiles or other discretized classes. This is one way to avoid issues associated with discretization.

Parameters

y [array] (n, t) with t>>n, one row per observation (n total), one column recording the value of each observation, with as many columns as time periods.

Notes

Refer to [Rey14b] Equation (11) for details. Ties are resolved by assigning distinct ranks, corresponding to the order that the values occur in each cross section.

Examples

US nominal per capita income 48 states 81 years 1929-2009

```
>>> from giddy.markov import FullRank_Markov
>>> import libpysal as ps
>>> import numpy as np
>>> f = ps.io.open(ps.examples.get_path("usjoin.csv"))
>>> pci = np.array([f.by_col[str(y)] for y in range(1929,2010)]).transpose()
>>> m = FullRank_Markov(pci)
>>> m.ranks
array([[45, 45, 44, ..., 41, 40, 39],
       [24, 25, 25, ..., 36, 38, 41],
       [46, 47, 45, ..., 43, 43, 43],
       ...,
       [34, 34, 34, ..., 47, 46, 42],
       [17, 17, 22, ..., 25, 26, 25],
       [16, 18, 19, ...,  6,  6,  7]])
>>> m.transitions
array([[66.,  5.,  5., ...,  0.,  0.,  0.],
       [ 8., 51.,  9., ...,  0.,  0.,  0.],
       [ 2., 13., 44., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ..., 40., 17.,  0.],
       [ 0.,  0.,  0., ..., 15., 54.,  2.],
       [ 0.,  0.,  0., ...,  2.,  1., 77.]])
>>> m.p[0, :5]
array([0.825 , 0.0625, 0.0625, 0.025 , 0.025 ])
>>> m.fmpt[0, :5]
array([48.        , 87.96280048, 68.1089084 , 58.83306575, 41.77250827])
>>> m.sojourn_time[:5]
array([5.71428571, 2.75862069, 2.22222222, 1.77777778, 1.66666667])
```

Attributes

ranks [array] ranks of the original y array (by columns): higher values rank higher, e.g. the largest value in a column ranks 1.

p [array] (n, n), transition probability matrix for Full Rank Markov.

steady_state [array] (n,), ergodic distribution.

transitions [array] (n, n), count of transitions between each rank i and j

fmpt [array] (n, n), first mean passage times.

sojourn_time [array] (n,), sojourn times.

__init__ (*self*, *y*)
Initialize self. See help(type(self)) for accurate signature.

2.1.5 giddy.markov.GeoRank_Markov

class giddy.markov.**GeoRank_Markov**(*y*)
Geographic Rank Markov. Geographic units are considered as Markov states.

Parameters

y [array] (n, t) with t>>n, one row per observation (n total), one column recording the value of each observation, with as many columns as time periods.

Notes

Refer to [Rey14b] Equation (13)-(16) for details. Ties are resolved by assigning distinct ranks, corresponding to the order that the values occur in each cross section.

Examples

US nominal per capita income 48 states 81 years 1929-2009

```
>>> from giddy.markov import GeoRank_Markov
>>> import libpysal as ps
>>> import numpy as np
>>> f = ps.io.open(ps.examples.get_path("usjoin.csv"))
>>> pci = np.array([f.by_col[str(y)] for y in range(1929,2010)]).transpose()
>>> m = GeoRank_Markov(pci)
>>> m.transitions
array([[38.,  0.,  8., ...,  0.,  0.,  0.],
       [ 0., 15.,  0., ...,  0.,  1.,  0.],
       [ 6.,  0., 44., ...,  5.,  0.,  0.],
       ...,
       [ 2.,  0.,  5., ..., 34.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0., 18.,  2.],
       [ 0.,  0.,  0., ...,  0.,  3., 14.]])
>>> m.p
array([[0.475 , 0.    , 0.1   , ..., 0.    , 0.    , 0.    ],
       [0.    , 0.1875, 0.    , ..., 0.    , 0.0125, 0.    ],
       [0.075 , 0.    , 0.55  , ..., 0.0625, 0.    , 0.    ],
       ...,
       [0.025 , 0.    , 0.0625, ..., 0.425 , 0.    , 0.    ],
       [0.    , 0.    , 0.    , ..., 0.    , 0.225 , 0.025 ],
       [0.    , 0.    , 0.    , ..., 0.    , 0.0375, 0.175 ]])
>>> m.fmpt
array([[ 48.        ,  63.35532038,  92.75274652, ...,  82.47515731,
         71.01114491,  68.65737127],
       [108.25928005,  48.        , 127.99032986, ...,  92.03098299,
         63.36652935,  61.82733039],
```

(continues on next page)

(continued from previous page)

```
       [ 76.96801786,  64.7713783 ,  48.        , ...,  73.84595169,
         72.24682723,  69.77497173],
       ...,
       [ 93.3107474 ,  62.47670463, 105.80634118, ...,  48.        ,
         69.30121319,  67.08838421],
       [113.65278078,  61.1987031 , 133.57991745, ...,  96.0103924 ,
         48.        ,  56.74165107],
       [114.71894813,  63.4019776 , 134.73381719, ...,  97.287895  ,
         61.45565054,  48.        ]])
>>> m.sojourn_time
array([ 1.9047619 ,  1.23076923,  2.22222222,  1.73913043,  1.15942029,
        3.80952381,  1.70212766,  1.25      ,  1.31147541,  1.11111111,
        1.73913043,  1.37931034,  1.17647059,  1.21212121,  1.33333333,
        1.37931034,  1.09589041,  2.10526316,  2.        ,  1.45454545,
        1.26984127, 26.66666667,  1.19402985,  1.23076923,  1.09589041,
        1.56862745,  1.26984127,  2.42424242,  1.50943396,  2.        ,
        1.29032258,  1.09589041,  1.6       ,  1.42857143,  1.25      ,
        1.45454545,  1.29032258,  1.6       ,  1.17647059,  1.56862745,
        1.25      ,  1.37931034,  1.45454545,  1.42857143,  1.29032258,
        1.73913043,  1.29032258,  1.21212121])
```

Attributes

p [array] (n, n), transition probability matrix for geographic rank Markov.

steady_state [array] (n,), ergodic distribution.

transitions [array] (n, n), count of rank transitions between each geographic unit i and j.

fmpt [array] (n, n), first mean passage times.

sojourn_time [array] (n,), sojourn times.

__init__(*self*, *y*)

Initialize self. See help(type(self)) for accurate signature.

2.1.6 giddy.markov.kullback

giddy.markov.**kullback**(*F*)

Kullback information based test of Markov Homogeneity.

Parameters

F [array] (s, r, r), values are transitions (not probabilities) for s strata, r initial states, r terminal states.

Returns

Results [dictionary] (key - value)

Conditional homogeneity - (float) test statistic for homogeneity of transition probabilities across strata.

Conditional homogeneity pvalue - (float) p-value for test statistic.

Conditional homogeneity dof - (int) degrees of freedom = r(s-1)(r-1).

Notes

Based on [KKK62]. Example below is taken from Table 9.2 .

Examples

```
>>> import numpy as np
>>> from giddy.markov import kullback
>>> s1 = np.array([
...         [ 22, 11, 24,  2,  2,  7],
...         [ 5, 23, 15,  3, 42,  6],
...         [ 4, 21, 190, 25, 20, 34],
...         [0, 2, 14, 56, 14, 28],
...         [32, 15, 20, 10, 56, 14],
...         [5, 22, 31, 18, 13, 134]
...     ])
>>> s2 = np.array([
...     [3, 6, 9, 3, 0, 8],
...     [1, 9, 3, 12, 27, 5],
...     [2, 9, 208, 32, 5, 18],
...     [0, 14, 32, 108, 40, 40],
...     [22, 14, 9, 26, 224, 14],
...     [1, 5, 13, 53, 13, 116]
...     ])
>>>
>>> F = np.array([s1, s2])
>>> res = kullback(F)
>>> "%8.3f"%res['Conditional homogeneity']
' 160.961'
>>> "%d"%res['Conditional homogeneity dof']
'30'
>>> "%3.1f"%res['Conditional homogeneity pvalue']
'0.0'
```

2.1.7 giddy.markov.prais

giddy.markov.**prais**(*pmat*)

Prais conditional mobility measure.

Parameters

pmat [matrix] (k, k), Markov probability transition matrix.

Returns

pr [matrix] (1, k), conditional mobility measures for each of the k classes.

Notes

Prais' conditional mobility measure for a class is defined as:

$$pr_i = 1 - p_{i,i}$$

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from giddy.markov import Markov,prais
>>> f = libpysal.io.open(libpysal.examples.get_path("usjoin.csv"))
>>> pci = np.array([f.by_col[str(y)] for y in range(1929,2010)])
>>> q5 = np.array([mc.Quantiles(y).yb for y in pci]).transpose()
>>> m = Markov(q5)
>>> m.transitions
array([[729.,  71.,   1.,   0.,   0.],
       [ 72., 567.,  80.,   3.,   0.],
       [  0.,  81., 631.,  86.,   2.],
       [  0.,   3.,  86., 573.,  56.],
       [  0.,   0.,   1.,  57., 741.]])
>>> m.p
array([[0.91011236, 0.0886392 , 0.00124844, 0.        , 0.        ],
       [0.09972299, 0.78531856, 0.11080332, 0.00415512, 0.        ],
       [0.        , 0.10125   , 0.78875   , 0.1075    , 0.0025    ],
       [0.        , 0.00417827, 0.11977716, 0.79805014, 0.07799443],
       [0.        , 0.        , 0.00125156, 0.07133917, 0.92740926]])
>>> prais(m.p)
array([0.08988764, 0.21468144, 0.21125   , 0.20194986, 0.07259074])
```

2.1.8 giddy.markov.homogeneity

giddy.markov.**homogeneity**(*transition_matrices*, *regime_names=[]*, *class_names=[]*, *title='Markov Homogeneity Test'*)

Test for homogeneity of Markov transition probabilities across regimes.

Parameters

transition_matrices [list] of transition matrices for regimes, all matrices must have same size (r, c). r is the number of rows in the transition matrix and c is the number of columns in the transition matrix.

regime_names [sequence] Labels for the regimes.

class_names [sequence] Labels for the classes/states of the Markov chain.

title [string] name of test.

Returns

: implicit an instance of Homogeneity_Results.

2.1.9 giddy.markov.sojourn_time

giddy.markov.**sojourn_time**(*p*)

Calculate sojourn time based on a given transition probability matrix.

Parameters

p [array] (k, k), a Markov transition probability matrix.

Returns

: array (k,), sojourn times. Each element is the expected time a Markov chain spends in each states before leaving that state.

Notes

Refer to [Ibe09] for more details on sojourn times for Markov chains.

Examples

```
>>> from giddy.markov import sojourn_time
>>> import numpy as np
>>> p = np.array([[.5, .25, .25], [.5, 0, .5], [.25, .25, .5]])
>>> sojourn_time(p)
array([2., 1., 2.])
```

2.1.10 giddy.ergodic.steady_state

giddy.ergodic.**steady_state**(*P*)

Calculates the steady state probability vector for a regular Markov transition matrix P.

Parameters

P [array] (k, k), an ergodic Markov transition probability matrix.

Returns

: array (k,), steady state distribution.

Examples

Taken from [KS67]. Land of Oz example where the states are Rain, Nice and Snow, so there is 25 percent chance that if it rained in Oz today, it will snow tomorrow, while if it snowed today in Oz there is a 50 percent chance of snow again tomorrow and a 25 percent chance of a nice day (nice, like when the witch with the monkeys is melting).

```
>>> import numpy as np
>>> from giddy.ergodic import steady_state
>>> p=np.array([[.5, .25, .25],[.5,0,.5],[.25,.25,.5]])
>>> steady_state(p)
array([0.4, 0.2, 0.4])
```

Thus, the long run distribution for Oz is to have 40 percent of the days classified as Rain, 20 percent as Nice, and 40 percent as Snow (states are mutually exclusive).

2.1.11 giddy.ergodic.fmpt

giddy.ergodic.**fmpt**(*P*)

Calculates the matrix of first mean passage times for an ergodic transition probability matrix.

Parameters

P [array] (k, k), an ergodic Markov transition probability matrix.

Returns

M [array] (k, k), elements are the expected value for the number of intervals required for a chain starting in state i to first enter state j. If i=j then this is the recurrence time.

Notes

Uses formulation (and examples on p. 218) in [KS67].

Examples

```
>>> import numpy as np
>>> from giddy.ergodic import fmpt
>>> p=np.array([[.5, .25, .25],[.5,0,.5],[.25,.25,.5]])
>>> fm=fmpt(p)
>>> fm
array([[2.5       , 4.        , 3.33333333],
       [2.66666667, 5.        , 2.66666667],
       [3.33333333, 4.        , 2.5       ]])
```

Thus, if it is raining today in Oz we can expect a nice day to come along in another 4 days, on average, and snow to hit in 3.33 days. We can expect another rainy day in 2.5 days. If it is nice today in Oz, we would experience a change in the weather (either rain or snow) in 2.67 days from today. (That wicked witch can only die once so I reckon that is the ultimate absorbing state).

2.1.12 giddy.ergodic.var_fmpt

giddy.ergodic.**var_fmpt**(*P*)

Variances of first mean passage times for an ergodic transition probability matrix.

Parameters

P [array] (k, k), an ergodic Markov transition probability matrix.

Returns

: array (k, k), elements are the variances for the number of intervals required for a chain starting in state i to first enter state j.

Notes

Uses formulation (and examples on p. 83) in [KS67].

Examples

```
>>> import numpy as np
>>> from giddy.ergodic import var_fmpt
>>> p=np.array([[.5, .25, .25],[.5,0,.5],[.25,.25,.5]])
>>> vfm=var_fmpt(p)
>>> vfm
array([[ 5.58333333, 12.        ,  6.88888889],
       [ 6.22222222, 12.        ,  6.22222222],
       [ 6.88888889, 12.        ,  5.58333333]])
```

2.2 Directional LISA

giddy.directional.Rose(Y, w[, k])	Rose diagram based inference for directional LISAs.

2.2.1 giddy.directional.Rose

class giddy.directional.**Rose**(*Y*, *w*, *k=8*)

Rose diagram based inference for directional LISAs.

For n units with LISA values at two points in time, the Rose class provides the LISA vectors, their visualization, and computationally based inference.

Parameters

Y [array (n,2)] Columns correspond to end-point time periods to calculate LISA vectors for n object.

w [PySAL W] Spatial weights object.

k [int] Number of circular sectors in rose diagram.

Attributes

cuts [(k, 1) ndarray] Radian cuts for rose diagram (circular histogram).

counts: (k, 1) ndarray Number of vectors contained in each sector.

r [(n, 1) ndarray] Vector lengths.

theta [(n,1) ndarray] Signed radians for observed LISA vectors.

If self.permute is called the following attributes are available:

alternative [string] Form of the specified alternative hypothesis ['two-sided'(default) | 'positive' | 'negative']

counts_perm [(permutations, k) ndarray] Counts obtained for each sector for every permutation

expected_perm [(k, 1) ndarray] Average number of counts for each sector taken over all permutations.

p [(k, 1) ndarray] Psuedo p-values for the observed sector counts under the specified alternative.

larger_perm [(k, 1) ndarray] Number of times realized counts are as large as observed sector count.

smaller_perm [(k, 1) ndarray] Number of times realized counts are as small as observed sector count.

Methods

permute(self[, permutations, alternative])	Generate ransom spatial permutations for inference on LISA vectors.
plot(self[, attribute, ax])	Plot the rose diagram.
plot_origin(self)	Plot vectors of positional transition of LISA values starting from the same origin.
plot_vectors(self[, arrows])	Plot vectors of positional transition of LISA values within quadrant in scatterplot in a polar plot.

giddy.directional.Rose

Rose.**permute**(*self*, *permutations=99*, *alternative='two.sided'*)

Generate ransom spatial permutations for inference on LISA vectors.

Parameters

permutations [int, optional] Number of random permutations of observations.

alternative [string, optional] Type of alternative to form in generating p-values. Options are: *two-sided* which tests for difference between observed counts and those obtained from the permutation distribution; *positive* which tests the alternative that the focal unit and its lag move in the same direction over time; *negative* which tests that the focal unit and its lag move in opposite directions over the interval.

giddy.directional.Rose

Rose.**plot**(*self*, *attribute=None*, *ax=None*, ***kwargs*)

Plot the rose diagram.

Parameters

attribute [(n,) ndarray, optional] Variable to specify colors of the colorbars.

ax [Matplotlib Axes instance, optional] If given, the figure will be created inside this axis. Default =None. Note, this axis should have a polar projection.

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the plot. Note: 'c' and 'color' cannot be passed when attribute is not None

Returns

fig [Matplotlib Figure instance] Moran scatterplot figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

giddy.directional.Rose

Rose.**plot_origin**(*self*)

Plot vectors of positional transition of LISA values starting from the same origin.

giddy.directional.Rose

Rose.**plot_vectors**(*self*, *arrows=True*)

Plot vectors of positional transition of LISA values within quadrant in scatterplot in a polar plot.

Parameters

ax [Matplotlib Axes instance, optional] If given, the figure will be created inside this axis. Default =None.

arrows [boolean, optional] If True show arrowheads of vectors. Default =True

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the plot. Note: 'c' and 'color' cannot be passed when attribute is not None

Returns

fig [Matplotlib Figure instance] Moran scatterplot figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

__init__ (*self*, *Y*, *w*, *k=8*)

Calculation of rose diagram for local indicators of spatial association.

Parameters

Y [(n, 2) ndarray] Variable observed on n spatial units over 2 time periods

w [W] Spatial weights object.

k [int] number of circular sectors in rose diagram (the default is 8).

Notes

Based on [RMA11].

Examples

Constructing data for illustration of directional LISA analytics. Data is for the 48 lower US states over the period 1969-2009 and includes per capita income normalized to the national average.

Load comma delimited data file in and convert to a numpy array

```
>>> import libpysal
>>> from giddy.directional import Rose
>>> import matplotlib.pyplot as plt
>>> file_path = libpysal.examples.get_path("spi_download.csv")
>>> f=open(file_path,'r')
>>> lines=f.readlines()
>>> f.close()
>>> lines=[line.strip().split(",") for line in lines]
>>> names=[line[2] for line in lines[1:-5]]
>>> data=np.array([list(map(int,line[3:])) for line in lines[1:-5]])
```

Bottom of the file has regional data which we don't need for this example so we will subset only those records that match a state name

```
>>> sids=list(range(60))
>>> out=['"United States 3/"',
...      '"Alaska 3/"',
...      '"District of Columbia"',
...      '"Hawaii 3/"',
...      '"New England"',
...      '"Mideast"',
...      '"Great Lakes"',
...      '"Plains"',
...      '"Southeast"',
...      '"Southwest"',
...      '"Rocky Mountain"',
...      '"Far West 3/"']
>>> snames=[name for name in names if name not in out]
>>> sids=[names.index(name) for name in snames]
>>> states=data[sids,:]
>>> us=data[0]
>>> years=np.arange(1969,2009)
```

Now we convert state incomes to express them relative to the national average

```
>>> rel=states/(us*1.)
```

Create our contiguity matrix from an external GAL file and row standardize the resulting weights

```
>>> gal=libpysal.io.open(libpysal.examples.get_path('states48.gal'))
>>> w=gal.read()
>>> w.transform='r'
```

Take the first and last year of our income data as the interval to do the directional directional analysis

```
>>> Y=rel[:,[0,-1]]
```

Set the random seed generator which is used in the permutation based inference for the rose diagram so that we can replicate our example results

```
>>> np.random.seed(100)
```

Call the rose function to construct the directional histogram for the dynamic LISA statistics. We will use four circular sectors for our histogram

```
>>> r4=Rose(Y,w,k=4)
```

What are the cut-offs for our histogram - in radians

```
>>> r4.cuts
array([0.        , 1.57079633, 3.14159265, 4.71238898, 6.28318531])
```

How many vectors fell in each sector

```
>>> r4.counts
array([32,  5,  9,  2])
```

We can test whether these counts are different than what would be expected if there was no association between the movement of the focal unit and its spatial lag.

To do so we call the *permute* method of the object

```
>>> r4.permute()
```

and then inspect the *p* attibute:

```
>>> r4.p
array([0.04, 0.  , 0.02, 0.  ])
```

Repeat the exercise but now for 8 rather than 4 sectors

```
>>> r8 = Rose(Y, w, k=8)
>>> r8.counts
array([19, 13,  3,  2,  7,  2,  1,  1])
>>> r8.permute()
>>> r8.p
array([0.86, 0.08, 0.16, 0.  , 0.02, 0.2 , 0.56, 0.  ])
```

The default is a two-sided alternative. There is an option for a directional alternative reflecting positive co-movement of the focal series with its spatial lag. In this case the number of vectors in quadrants I and III should be much larger than expected, while the counts of vectors falling in quadrants II and IV should be much lower than expected.

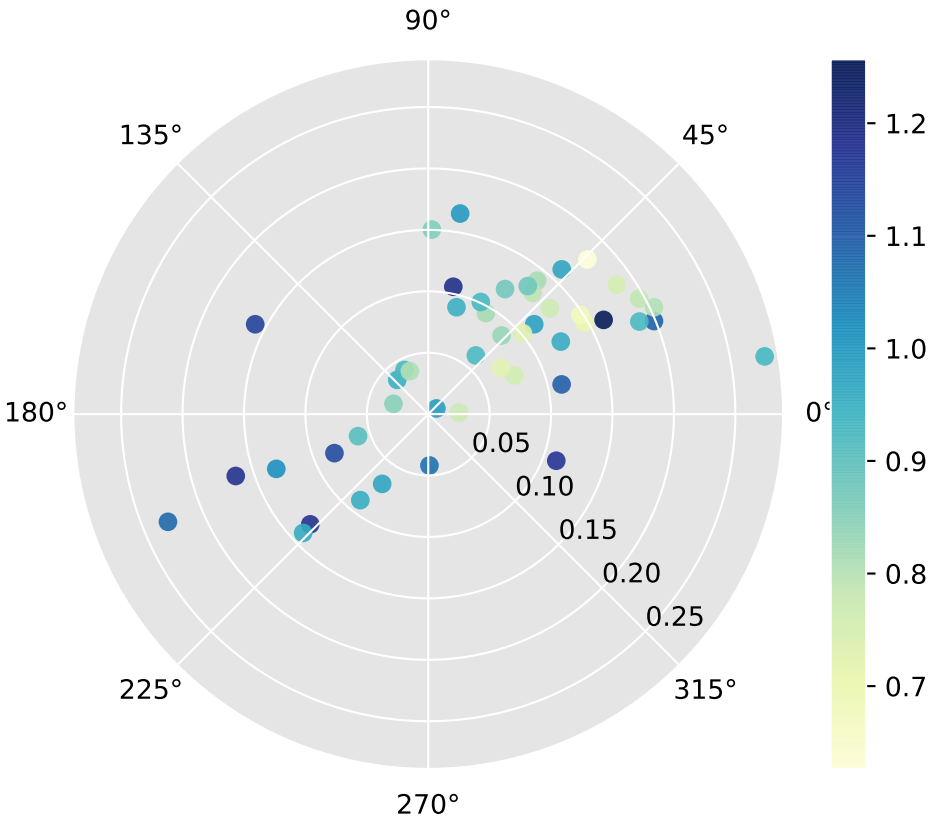
```
>>> r8.permute(alternative='positive')
>>> r8.p
array([0.51, 0.04, 0.28, 0.02, 0.01, 0.14, 0.57, 0.03])
```

Finally, there is a second directional alternative for examining the hypothesis that the focal unit and its lag move in opposite directions.

```
>>> r8.permute(alternative='negative')
>>> r8.p
array([0.69, 0.99, 0.92, 1.  , 1.  , 0.97, 0.74, 1.  ])
```

We can call the plot method to visualize directional LISAs as a rose diagram conditional on the starting relative income:

```
>>> fig1, _ = r8.plot(attribute=Y[:,0])
>>> plt.show(fig1)
```



2.3 Economic Mobility Indices

giddy.mobility.markov_mobility(p[, measure, ini])	Markov-based mobility index.

2.3.1 giddy.mobility.markov_mobility

giddy.mobility.**markov_mobility**(*p*, *measure='P'*, *ini=None*)

Markov-based mobility index.

Parameters

p [array] (k, k), Markov transition probability matrix.

measure [string] If measure= "P", $M_P = \frac{m-\sum_{i=1}^{m} P_{ii}}{m-1}$; if measure = "D", $M_D = 1 - |\det(P)|$, where $\det(P)$ is the determinant of P; if measure = "L2", $M_{L2} = 1 - |\lambda_2|$, where λ_2 is the second largest eigenvalue of P; if measure = "B1", $M_{B1} = \frac{m-m\sum_{i=1}^{m} \pi_i P_{ii}}{m-1}$, where π is the initial income distribution; if measure == "B2", $M_{B2} = \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=1}^{m} \pi_i P_{ij} |i-j|$, where π is the initial income distribution.

ini [array] (k,), initial distribution. Need to be specified if measure = "B1" or "B2". If not, the initial distribution would be treated as a uniform distribution.

Returns

mobi [float] Mobility value.

Notes

The mobility indices are based on [FSZ04].

Examples

```
>>> import numpy as np
>>> import libpysal
>>> import mapclassify as mc
>>> from giddy.markov import Markov
>>> from giddy.mobility import markov_mobility
>>> f = libpysal.io.open(libpysal.examples.get_path("usjoin.csv"))
>>> pci = np.array([f.by_col[str(y)] for y in range(1929,2010)])
>>> q5 = np.array([mc.Quantiles(y).yb for y in pci]).transpose()
>>> m = Markov(q5)
>>> m.p
array([[0.91011236, 0.0886392 , 0.00124844, 0.        , 0.        ],
       [0.09972299, 0.78531856, 0.11080332, 0.00415512, 0.        ],
       [0.        , 0.10125   , 0.78875   , 0.1075    , 0.0025    ],
       [0.        , 0.00417827, 0.11977716, 0.79805014, 0.07799443],
       [0.        , 0.        , 0.00125156, 0.07133917, 0.92740926]])
```

(1) Estimate Shorrock1 mobility index:

```
>>> mobi_1 = markov_mobility(m.p, measure="P")
>>> print("{:.5f}".format(mobi_1))
0.19759
```

(2) Estimate Shorrock2 mobility index:

```
>>> mobi_2 = markov_mobility(m.p, measure="D")
>>> print("{:.5f}".format(mobi_2))
0.60685
```

(3) Estimate Sommers and Conlisk mobility index:

```
>>> mobi_3 = markov_mobility(m.p, measure="L2")
>>> print("{:.5f}".format(mobi_3))
0.03978
```

(4) Estimate Bartholomew1 mobility index (note that the initial distribution should be given):

```
>>> ini = np.array([0.1,0.2,0.2,0.4,0.1])
>>> mobi_4 = markov_mobility(m.p, measure = "B1", ini=ini)
>>> print("{:.5f}".format(mobi_4))
0.22777
```

(5) Estimate Bartholomew2 mobility index (note that the initial distribution should be given):

```
>>> ini = np.array([0.1,0.2,0.2,0.4,0.1])
>>> mobi_5 = markov_mobility(m.p, measure = "B2", ini=ini)
>>> print("{:.5f}".format(mobi_5))
0.04637
```

2.4 Exchange Mobility Methods

giddy.rank.Theta(y, regime[, permutations])	Regime mobility measure.
giddy.rank.Tau(x, y)	Kendall's Tau is based on a comparison of the number of pairs of n observations that have concordant ranks between two variables.
giddy.rank.SpatialTau(x, y, w[, permutations])	Spatial version of Kendall's rank correlation statistic.
giddy.rank.Tau_Local(x, y)	Local version of the classic Tau.
giddy.rank.Tau_Local_Neighbor(x, y, w[, ...])	Neighbor set LIMA.
giddy.rank.Tau_Local_Neighborhood(x, y, w[, ...])	Neighborhood set LIMA.
giddy.rank.Tau_Regional(x, y, regime[, ...])	Inter and intraregional decomposition of the classic Tau.

2.4.1 giddy.rank.Theta

class giddy.rank.**Theta**(*y*, *regime*, *permutations=999*)

Regime mobility measure. [Rey04]

For sequence of time periods Theta measures the extent to which rank changes for a variable measured over n locations are in the same direction within mutually exclusive and exhaustive partitions (regimes) of the n locations.

Theta is defined as the sum of the absolute sum of rank changes within the regimes over the sum of all absolute rank changes.

Parameters

y [array] (n, k) with k>=2, successive columns of y are later moments in time (years, months, etc).

regime [array] (n,), values corresponding to which regime each observation belongs to.

permutations [int] number of random spatial permutations to generate for computationally based inference.

Examples

```
>>> import libpysal as ps
>>> from giddy.rank import Theta
>>> import numpy as np
>>> f=ps.io.open(ps.examples.get_path("mexico.csv"))
>>> vnames=["pcgdp%d"%dec for dec in range(1940,2010,10)]
>>> y=np.transpose(np.array([f.by_col[v] for v in vnames]))
>>> regime=np.array(f.by_col['esquivel99'])
>>> np.random.seed(10)
>>> t=Theta(y,regime,999)
>>> t.theta
array([[0.41538462, 0.28070175, 0.61363636, 0.62222222, 0.33333333,
        0.47222222]])
>>> t.pvalue_left
array([0.307, 0.077, 0.823, 0.552, 0.045, 0.735])
>>> t.total
array([130., 114.,  88.,  90.,  90.,  72.])
>>> t.max_total
512
```

Attributes

ranks [array] ranks of the original y array (by columns).

regimes [array] the original regimes array.

total [array] (k-1,), the total number of rank changes for each of the k periods.

max_total [int] the theoretical maximum number of rank changes for n observations.

theta [array] (k-1,), the theta statistic for each of the k-1 intervals.

permutations [int] the number of permutations.

pvalue_left [float] p-value for test that observed theta is significantly lower than its expectation under complete spatial randomness.

pvalue_right [float] p-value for test that observed theta is significantly greater than its expectation under complete spatial randomness.

__init__(*self*, *y*, *regime*, *permutations=999*)
Initialize self. See help(type(self)) for accurate signature.

2.4.2 giddy.rank.Tau

class giddy.rank.**Tau**(*x*, *y*)
Kendall's Tau is based on a comparison of the number of pairs of n observations that have concordant ranks between two variables.

Parameters

x [array] (n,), first variable.

y [array] (n,), second variable.

Notes

Modification of algorithm suggested by [Chr05].PySAL/giddy implementation uses a list based representation of a binary tree for the accumulation of the concordance measures. Ties are handled by this implementation (in other words, if there are ties in either x, or y, or both, the calculation returns Tau_b, if no ties classic Tau is returned.)

Examples

```
>>> from scipy.stats import kendalltau
>>> from giddy.rank import Tau
>>> x1 = [12, 2, 1, 12, 2]
>>> x2 = [1, 4, 7, 1, 0]
>>> kt = Tau(x1,x2)
>>> kt.tau
-0.47140452079103173
>>> kt.tau_p
0.24821309157521476
>>> tau, p = kendalltau(x1,x2)
>>> tau
-0.4714045207910316
>>> p
0.2827454599327748
```

Attributes

tau [float] The classic Tau statistic.

tau_p [float] asymptotic p-value.

__init__ (*self*, *x*, *y*)
Initialize self. See help(type(self)) for accurate signature.

2.4.3 giddy.rank.SpatialTau

class giddy.rank.**SpatialTau**(*x*, *y*, *w*, *permutations=0*)
Spatial version of Kendall's rank correlation statistic.

Kendall's Tau is based on a comparison of the number of pairs of n observations that have concordant ranks between two variables. The spatial Tau decomposes these pairs into those that are spatial neighbors and those that are not, and examines whether the rank correlation is different between the two sets relative to what would be expected under spatial randomness.

Parameters

x [array] (n,), first variable.

y [array] (n,), second variable.

w [W] spatial weights object.

permutations [int] number of random spatial permutations for computationally based inference.

Notes

Algorithm has two stages. The first calculates classic Tau using a list based implementation of the algorithm from [Chr05]. Second stage calculates concordance measures for neighboring pairs of locations using a modification of the algorithm from [PTVF07]. See [Rey14a] for details.

Examples

```
>>> import libpysal as ps
>>> import numpy as np
>>> from giddy.rank import SpatialTau
>>> f=ps.io.open(ps.examples.get_path("mexico.csv"))
>>> vnames=["pcgdp%d"%dec for dec in range(1940,2010,10)]
>>> y=np.transpose(np.array([f.by_col[v] for v in vnames]))
>>> regime=np.array(f.by_col['esquivel99'])
>>> w=ps.weights.block_weights(regime)
>>> np.random.seed(12345)
>>> res=[SpatialTau(y[:,i],y[:,i+1],w,99) for i in range(6)]
>>> for r in res:
...     ev = r.taus.mean()
...     "%8.3f %8.3f %8.3f"%(r.tau_spatial, ev, r.tau_spatial_psim)
...
'   0.397    0.659    0.010'
'   0.492    0.706    0.010'
'   0.651    0.772    0.020'
'   0.714    0.752    0.210'
'   0.683    0.705    0.270'
'   0.810    0.819    0.280'
```

Attributes

tau [float] The classic Tau statistic.

tau_spatial [float] Value of Tau for pairs that are spatial neighbors.

taus [array] (permtuations, 1), values of simulated tau_spatial values under random spatial permutations in both periods. (Same permutation used for start and ending period).

pairs_spatial [int] Number of spatial pairs.

concordant [float] Number of concordant pairs.

concordant_spatial [float] Number of concordant pairs that are spatial neighbors.

extraX [float] Number of extra X pairs.

extraY [float] Number of extra Y pairs.

discordant [float] Number of discordant pairs.

discordant_spatial [float] Number of discordant pairs that are spatial neighbors.

taus [float] spatial tau values for permuted samples (if permutations>0).

tau_spatial_psim [float] one-sided pseudo p-value for observed tau_spatial under the null of spatial randomness of rank exchanges (if permutations>0).

__init__ (*self*, *x*, *y*, *w*, *permutations=0*)
Initialize self. See help(type(self)) for accurate signature.

2.4.4 giddy.rank.Tau_Local

class giddy.rank.**Tau_Local**(*x*, *y*)

Local version of the classic Tau.

Decomposition of the classic Tau into local components.

Parameters

x [array] (n,), first variable.

y [array] (n,), second variable.

Notes

The equation for calculating local concordance statistic can be found in [Rey16] Equation (9).

Examples

```
>>> import libpysal as ps
>>> import numpy as np
>>> from giddy.rank import Tau_Local,Tau
>>> np.random.seed(10)
>>> f = ps.io.open(ps.examples.get_path("mexico.csv"))
>>> vnames = ["pcgdp%d"%dec for dec in range(1940, 2010, 10)]
>>> y = np.transpose(np.array([f.by_col[v] for v in vnames]))
>>> r = y / y.mean(axis=0)
>>> tau_local = Tau_Local(r[:,0],r[:,1])
>>> tau_local.tau_local
array([-0.03225806,  0.93548387,  0.80645161,  0.74193548,  0.93548387,
        0.74193548,  0.67741935,  0.41935484,  1.        ,  0.5483871 ,
        0.74193548,  0.93548387,  0.67741935,  0.74193548,  0.80645161,
        0.74193548,  0.5483871 ,  0.67741935,  0.74193548,  0.74193548,
        0.5483871 , -0.16129032,  0.93548387,  0.61290323,  0.67741935,
        0.48387097,  0.93548387,  0.61290323,  0.74193548,  0.41935484,
        0.61290323,  0.61290323])
>>> tau_local.tau
0.6612903225806451
>>> tau_classic = Tau(r[:,0],r[:,1])
>>> tau_classic.tau
0.6612903225806451
```

Attributes

n [int] number of observations.

tau [float] The classic Tau statistic.

tau_local [array] (n,), local concordance (local version of the classic tau).

S [array] (n ,n), concordance matrix, s_{i,j}=1 if observation i and j are concordant, s_{i,j}=-1 if observation i and j are discordant, and s_{i,j}=0 otherwise.

__init__(*self*, *x*, *y*)

Initialize self. See help(type(self)) for accurate signature.

2.4.5 giddy.rank.Tau_Local_Neighbor

class giddy.rank.**Tau_Local_Neighbor**(*x*, *y*, *w*, *permutations=0*)

Neighbor set LIMA.

Local concordance relationships between a focal unit and its neighbors. A decomposition of local Tau into neighbor and non-neighbor components.

Parameters

x [array] (n,), first variable.

y [array] (n,), second variable.

w [W] spatial weights object.

permutations [int] number of random spatial permutations for computationally based inference.

Notes

The equation for calculating neighbor set LIMA statistic can be found in [Rey16] Equation (16).

Examples

```
>>> import libpysal as ps
>>> import numpy as np
>>> from giddy.rank import Tau_Local_Neighbor, SpatialTau
>>> np.random.seed(10)
>>> f = ps.io.open(ps.examples.get_path("mexico.csv"))
>>> vnames = ["pcgdp%d"%dec for dec in range(1940, 2010, 10)]
>>> y = np.transpose(np.array([f.by_col[v] for v in vnames]))
>>> r = y / y.mean(axis=0)
>>> regime = np.array(f.by_col['esquivel99'])
>>> w = ps.weights.block_weights(regime)
>>> res = Tau_Local_Neighbor(r[:,0], r[:,1], w, permutations=999)
>>> res.tau_ln
array([-0.2       ,  1.        ,  1.        ,  1.        ,  0.33333333,
        0.6       ,  0.6       , -0.5       ,  1.        ,  1.        ,
        0.2       ,  0.33333333,  0.33333333,  0.5       ,  1.        ,
        1.        ,  1.        ,  0.        ,  0.6       , -0.33333333,
       -0.33333333, -0.6       ,  1.        ,  0.2       ,  0.        ,
        0.2       ,  1.        ,  0.6       ,  0.33333333,  0.5       ,
        0.5       , -0.2       ])
>>> res.tau_ln_weights
array([0.03968254, 0.03968254, 0.03174603, 0.03174603, 0.02380952,
       0.03968254, 0.03968254, 0.03174603, 0.00793651, 0.03968254,
       0.03968254, 0.02380952, 0.02380952, 0.03174603, 0.00793651,
       0.02380952, 0.02380952, 0.03174603, 0.03968254, 0.02380952,
       0.02380952, 0.03968254, 0.03174603, 0.03968254, 0.03174603,
       0.03968254, 0.03174603, 0.03968254, 0.02380952, 0.03174603,
       0.03174603, 0.03968254])
>>> res.tau_ln_pvalues
array([0.541, 0.852, 0.668, 0.568, 0.11 , 0.539, 0.609, 0.058, 1.   ,
       0.255, 0.125, 0.087, 0.393, 0.433, 0.908, 0.657, 0.447, 0.128,
       0.531, 0.033, 0.12 , 0.271, 0.868, 0.234, 0.124, 0.387, 0.859,
       0.697, 0.349, 0.664, 0.596, 0.041])
```

(continues on next page)

(continued from previous page)

```
>>> res.sign
array([-1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, -1, -1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1])
>>> (res.tau_ln * res.tau_ln_weights).sum() #global spatial tau
0.39682539682539675
>>> res1 = SpatialTau(r[:,0],r[:,1],w,permutations=999)
>>> res1.tau_spatial
0.3968253968253968
```

Attributes

n [int] number of observations.

tau_local [array] (n,), local concordance (local version of the classic tau).

S [array] (n ,n), concordance matrix, s_{i,j}=1 if observation i and j are concordant, s_{i, j}=-1 if observation i and j are discordant, and s_{i,j}=0 otherwise.

tau_ln [array] (n,), observed neighbor set LIMA values.

tau_ln_weights [array] (n,), weights for neighbor set LIMA at each location. GIMA is the weighted average of neighbor set LIMA.

tau_ln_sim [array] (n, permutations), neighbor set LIMA values for permuted samples (if permutations>0).

tau_ln_pvalues [array] (n,), one-sided pseudo p-values for observed neighbor set LIMA values under the null that concordance relationship between the focal state and itsn eighbors is not different from what could be expected from randomly distributed rank changes.

sign [array] (n,), values indicate concordant or disconcordant: 1 concordant, -1 disconcordant

__init__(*self*, *x*, *y*, *w*, *permutations=0*)
Initialize self. See help(type(self)) for accurate signature.

2.4.6 giddy.rank.Tau_Local_Neighborhood

class giddy.rank.**Tau_Local_Neighborhood**(*x*, *y*, *w*, *permutations=0*)
Neighborhood set LIMA.

An extension of neighbor set LIMA. Consider local concordance relationships for a subset of states, defined as the focal state and its neighbors.

Parameters

x [array] (n,), first variable.

y [array] (n,), second variable.

w [W] spatial weights object.

permutations [int] number of random spatial permutations for computationally based inference.

Notes

The equation for calculating neighborhood set LIMA statistic can be found in [Rey16] Equation (22).

Examples

```
>>> import libpysal as ps
>>> from giddy.rank import Tau_Local_Neighborhood
>>> import numpy as np
>>> np.random.seed(10)
>>> f = ps.io.open(ps.examples.get_path("mexico.csv"))
>>> vnames = ["pcgdp%d"%dec for dec in range(1940, 2010, 10)]
>>> y = np.transpose(np.array([f.by_col[v] for v in vnames]))
>>> r = y / y.mean(axis=0)
>>> regime = np.array(f.by_col['esquivel99'])
>>> w = ps.weights.block_weights(regime)
>>> res = Tau_Local_Neighborhood(r[:,0],r[:,1],w,permutations=999)
>>> res.tau_lnhood
array([0.06666667, 0.6       , 0.2       , 0.8       , 0.33333333,
       0.6       , 0.6       , 0.2       , 1.        , 0.06666667,
       0.06666667, 0.33333333, 0.33333333, 0.2       , 1.        ,
       0.33333333, 0.33333333, 0.2       , 0.6       , 0.33333333,
       0.33333333, 0.06666667, 0.8       , 0.06666667, 0.2       ,
       0.6       , 0.8       , 0.6       , 0.33333333, 0.8       ,
       0.8       , 0.06666667])
>>> res.tau_lnhood_pvalues
array([0.106, 0.33 , 0.107, 0.535, 0.137, 0.414, 0.432, 0.169, 1.   ,
       0.03 , 0.019, 0.146, 0.249, 0.1  , 0.908, 0.225, 0.311, 0.125,
       0.399, 0.215, 0.334, 0.115, 0.669, 0.045, 0.11 , 0.525, 0.655,
       0.466, 0.236, 0.413, 0.504, 0.038])
>>> res.sign
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Attributes

n [int] number of observations.

tau_local [array] (n,), local concordance (local version of the classic tau).

S [array] (n ,n), concordance matrix, s_{i,j}=1 if observation i and j are concordant, s_{i, j}=-1 if observation i and j are discordant, and s_{i,j}=0 otherwise.

tau_lnhood [array] (n,), observed neighborhood set LIMA values.

tau_lnhood_sim [array] (n, permutations), neighborhood set LIMA values for permuted samples (if permutations>0).

tau_lnhood_pvalues [array] (n, 1), one-sided pseudo p-values for observed neighborhood set LIMA values under the null that the concordance relationships for a subset of states, defined as the focal state and its neighbors, is different from what would be expected from randomly distributed rank changes.

sign [array] (n,), values indicate concordant or disconcordant: 1 concordant, -1 disconcordant

__init__(*self*, *x*, *y*, *w*, *permutations=0*)
Initialize self. See help(type(self)) for accurate signature.

2.4.7 giddy.rank.Tau_Regional

class giddy.rank.**Tau_Regional**(*x*, *y*, *regime*, *permutations=0*)
Inter and intraregional decomposition of the classic Tau.

Parameters

x [array] (n,), first variable.

y [array] (n,), second variable.

regimes [array] (n,), ids of which regime an observation belongs to.

permutations [int] number of random spatial permutations for computationally based inference.

Notes

The equation for calculating inter and intraregional Tau statistic can be found in [Rey16] Equation (27).

Examples

```
>>> import libpysal as ps
>>> import numpy as np
>>> from giddy.rank import Tau_Regional
>>> np.random.seed(10)
>>> f = ps.io.open(ps.examples.get_path("mexico.csv"))
>>> vnames = ["pcgdp%d"%dec for dec in range(1940, 2010, 10)]
>>> y = np.transpose(np.array([f.by_col[v] for v in vnames]))
>>> r = y / y.mean(axis=0)
>>> regime = np.array(f.by_col['esquivel99'])
>>> res = Tau_Regional(y[:,0],y[:,-1],regime,permutations=999)
>>> res.tau_reg
array([[1.        , 0.25      , 0.5       , 0.6       , 0.83333333,
        0.6       , 1.        ],
       [0.25      , 0.33333333, 0.5       , 0.3       , 0.91666667,
        0.4       , 0.75      ],
       [0.5       , 0.5       , 0.6       , 0.4       , 0.38888889,
        0.53333333, 0.83333333],
       [0.6       , 0.3       , 0.4       , 0.2       , 0.4       ,
        0.28      , 0.8       ],
       [0.83333333, 0.91666667, 0.38888889, 0.4       , 0.6       ,
        0.73333333, 1.        ],
       [0.6       , 0.4       , 0.53333333, 0.28      , 0.73333333,
        0.8       , 0.8       ],
       [1.        , 0.75      , 0.83333333, 0.8       , 1.        ,
        0.8       , 0.33333333]])
>>> res.tau_reg_pvalues
array([[0.782, 0.227, 0.464, 0.638, 0.294, 0.627, 0.201],
       [0.227, 0.352, 0.391, 0.14 , 0.048, 0.252, 0.327],
       [0.464, 0.391, 0.587, 0.198, 0.107, 0.423, 0.124],
       [0.638, 0.14 , 0.198, 0.141, 0.184, 0.089, 0.217],
       [0.294, 0.048, 0.107, 0.184, 0.583, 0.25 , 0.005],
       [0.627, 0.252, 0.423, 0.089, 0.25 , 0.38 , 0.227],
       [0.201, 0.327, 0.124, 0.217, 0.005, 0.227, 0.322]])
```

Attributes

n [int] number of observations.

S [array] (n ,n), concordance matrix, s_{i,j}=1 if observation i and j are concordant, s_{i, j}=-1 if observation i and j are discordant, and s_{i,j}=0 otherwise.

tau_reg [array] (k, k), observed concordance matrix with diagonal elements measuring concordance between units within a regime and the off-diagonal elements denoting concordance between observations from a specific pair of different regimes.

tau_reg_sim [array] (permutations, k, k), concordance matrices for permuted samples (if permutations>0).

tau_reg_pvalues [array] (k, k), one-sided pseudo p-values for observed concordance matrix under the null that income mobility were random in its spatial distribution.

__init__(*self*, *x*, *y*, *regime*, *permutations=0*)
Initialize self. See help(type(self)) for accurate signature.

CHAPTER

THREE

REFERENCES

BIBLIOGRAPHY

[BB03] Frank Bickenbach and Eckhardt Bode. Evaluating the Markov property in studies of economic convergence. *International Regional Science Review*, 26(3):363–392, 2003. URL: https://doi.org/10.1177/0160017603253789, doi:10.1177/0160017603253789.

[Chr05] David Christensen. Fast algorithms for the calculation of kendall's . *Computational Statistics*, 20(1):51–62, Mar 2005. URL: https://doi.org/10.1007/BF02736122", doi:10.1007/BF02736122.

[FSZ04] John P. Formby, W. James Smith, and Buhong Zheng. Mobility measurement, transition matrices and statistical inference. *Journal of Econometrics*, 120(1):181–205, 2004. URL: http://www.sciencedirect.com/science/article/pii/S0304407603002112, doi:https://doi.org/10.1016/S0304-4076(03)00211-2.

[Ibe09] Oliver Ibe. *Markov processes for stochastic modeling*. Elsevier Academic Press, Amsterdam, 2009.

[KR18] Wei Kang and Sergio J. Rey. Conditional and joint tests for spatial effects in discrete markov chain models of regional income distribution dynamics. *The Annals of Regional Science*, 61(1):73–93, Jul 2018. URL: https://doi.org/10.1007/s00168-017-0859-9, doi:10.1007/s00168-017-0859-9.

[KS67] John G. Kemeny and James Laurie Snell. *Finite markov chains*. Van Nostrand, 1967.

[KKK62] S. Kullback, M. Kupperman, and H. H. Ku. Tests for contingency tables and Markov chains. *Technometrics*, 4(4):573–608, 1962. URL: http://www.jstor.org/stable/1266291, doi:10.2307/1266291.

[PTVF07] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes: the art of scientific computing*. Cambridge Univ Pr, Cambridge, 3rd edition, 2007.

[Rey01] Sergio J. Rey. Spatial empirics for economic growth and convergence. *Geographical Analysis*, 33(3):195–214, 2001. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.2001.tb00444.x, doi:10.1111/j.1538-4632.2001.tb00444.x.

[Rey04] Sergio J. Rey. Spatial dependence in the evolution of regional income distributions. In A. Getis, J. Múr, and H. Zoeller, editors, *Spatial econometrics and spatial statistics*, pages 193–213. Palgrave, Hampshire, 2004.

[Rey14a] Sergio J. Rey. Fast algorithms for a space-time concordance measure. *Computational Statistics*, 29(3-4):799–811, 2014. URL: https://doi.org/10.1007/s00180-013-0461-2, doi:10.1007/s00180-013-0461-2.

[Rey14b] Sergio J. Rey. Rank-based Markov chains for regional income distribution dynamics. *Journal of Geographical Systems*, 16(2):115–137, 2014.

[Rey16] Sergio J. Rey. Space–time patterns of rank concordance: local indicators of mobility association with application to spatial income inequality dynamics. *Annals of the American Association of Geographers*, 106(4):788–803, 2016. URL: https://doi.org/10.1080/24694452.2016.1151336, doi:10.1080/24694452.2016.1151336.

[RKW16] Sergio J. Rey, Wei Kang, and Levi Wolf. The properties of tests for spatial effects in discrete Markov chain models of regional income distribution dynamics. *Journal of Geographical Systems*, 18(4):377–398, 2016. URL: http://dx.doi.org/10.1007/s10109-016-0234-x, doi:10.1007/s10109-016-0234-x.

[RMA11] Sergio J. Rey, Alan T. Murray, and Luc Anselin. Visualizing regional income distribution dynamics. *Letters in Spatial and Resource Sciences*, 4(1):81–90, 2011. URL: https://doi.org/10.1007/s12076-010-0048-2, doi:10.1007/s12076-010-0048-2.

INDEX

Symbols

F

G

H

K

L

M

P

R

S

T

V