giddy Documentation

Release 2.2.2

pysal developers

Nov 20, 2019

CONTENTS:

1 Citing *giddy* **3**

1.1 Installation . . . 3
1.1.1 Installing released version . . . 3
1.1.2 Installing development version . . . 4
1.2 Tutorial . . . 4
1.2.1 Spatially Explicit Markov Methods . . . 4
1.2.2 Full Rank Markov and Geographic Rank Markov . . . 21
1.2.3 Measures of Income Mobility . . . 29
1.2.4 Directional Analysis of Dynamic LISAs . . . 32
1.2.5 Rank based Methods . . . 37
1.3 API reference . . . 52
1.3.1 Markov Methods . . . 52
1.3.2 Directional LISA . . . 71
1.3.3 Economic Mobility Indices . . . 76
1.3.4 Exchange Mobility Methods . . . 78
1.3.5 Alignment-based Sequence Methods . . . 87
1.4 References . . . 89

Bibliography **91**

Index **93**

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

CITING *GIDDY*

If you use PySAL-giddy in a scientific publication, we would appreciate using the following citation:

Bibtex entry:

```
@misc{wei_kang_2019_3351744,
  author       = {Wei Kang and
                  Sergio Rey and
                  Philip Stephens and
                  Nicholas Malizia and
                  Levi John Wolf and
                  Stefanie Lumnitz and
                  James Gaboardi and
                  jlaura and
                  Charles Schmidt and
                  eli knaap and
                  Andy Eschbacher},
  title        = {pysal/giddy: giddy 2.2.1},
  month        = jul,
  year         = 2019,
  doi          = {10.5281/zenodo.3351744},
  url          = {https://doi.org/10.5281/zenodo.3351744}
}
```

1.1 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.

1.1.1 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
```

1.1.2 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.

1.2 Tutorial

The following section was generated from doc/notebooks/MarkovBasedMethods.ipynb .....

1.2.1 Spatially Explicit Markov Methods

Author: Serge Rey sjsrey@gmail.com, Wei Kang weikang9009@gmail.com

Introduction

This notebook introduces Discrete Markov Chains (DMC) model and its two variants which explicitly incorporate spatial effects. We will demonstrate the usage of these methods by an empirical study for understanding *regional income dynamics in the US*. The dataset is the per capita incomes observed annually from 1929 to 2009 for the lower 48 US states.

- *Classic Markov*
- *Spatial Markov*
- *LISA Markov*

Note that a full execution of this notebook requires **pandas**, **matplotlib** and light-weight geovisualization package pysal-**splot**.

Classic Markov

```
giddy.markov.Markov(self, class_ids, classes=None)
```

We start with a look at a simple example of classic DMC methods implemented in PySAL-giddy. A Markov chain may be in one of k different states/classes at any point in time. These states are exhaustive and mutually exclusive. If one had a time series of remote sensing images used to develop land use classifications, then the states could be defined as the specific land use classes and interest would center on the transitions in and out of different classes for each pixel.

For example, suppose there are 5 pixels, each of which takes on one of 3 states (a,b,c) at 3 consecutive periods:

[1]:
```
import numpy as np
c = np.array([['b','a','c'],['c','c','a'],['c','b','c'],['a','a','b'],['a','b','c']])
```

So the first pixel was in state 'b' in period 1, state 'a' in period 2, and state 'c' in period 3. Each pixel's trajectory (row) owns Markov property, meaning that which state a pixel takes on today is only dependent on its immediate past.

Let's suppose that all the 5 pixels are governed by the same transition dynamics rule. That is, each trajectory is a realization of a Discrete Markov Chain process. We could pool all the 5 trajectories from which to estimate a transition probability matrix. To do that, we utlize the **Markov** class in **giddy**:

[2]:
```
import giddy
m = giddy.markov.Markov(c)
```

In this way, we create a **Markov** instance - m. Its attribute $classes$ gives 3 unique classes these pixels can take on, which are 'a','b' and 'c'.

[3]:
```
print(m.classes)
```
```
['a' 'b' 'c']
```

[4]:
```
print(len(m.classes))
```
```
3
```

In addition to extracting the unique states as an attribute, our **Markov** instance will also have the attribute *trnasitions* which is a transition matrix counting the number of transitions from one state to another. Since there are 3 unique states, we will have a $(3, 3)$ transtion matrix:

[5]:
```
print(m.transitions)
```
```
[[1. 2. 1.]
 [1. 0. 2.]
 [1. 1. 1.]]
```

The above transition matrix indicates that of the four pixels that began a transition interval in state 'a', 1 remained in that state, 2 transitioned to state 'b' and 1 transitioned to state 'c'. Another attribute p gives the transtion probability matrix which is the transition dynamics rule ubiquitous to all the 5 pixels across the 3 periods. The maximum likehood estimator for each element $p_{i,j}$ is shown below where $n_{i,j}$ is the number of transitions from state i to state j and k is the number of states (here $k = 3$):

$$\hat{p}_{i,j} = \frac{n_{i,j}}{\sum_{q=1}^{k} n_{i,q}}$$

[6]:
```
print(m.p)
```
```
[[0.25       0.5        0.25      ]
 [0.33333333 0.         0.66666667]
 [0.33333333 0.33333333 0.33333333]]
```

This means that if any of the 5 pixels was in state 'c', the probability of staying at 'c' or transitioning to any other states ('a', 'b') in the next period is the same (0.333). If a pixel was in state 'b', there is a high possibility that it would take on state 'c' in the next period because $\hat{p}_{2,3} = 0.667$.

[7]:
```
m.steady_state  # steady state distribution
```
[7]:
```
array([0.30769231, 0.28846154, 0.40384615])
```

This simple example illustrates the basic creation of a Markov instance, but the small sample size makes it unrealistic for the more advanced features of this approach. For a larger example, we will look at an application of Markov methods to understanding regional income dynamics in the US. Here we will load in data on per capita incomes observed annually from 1929 to 2010 for the lower 48 US states:

Regional income dynamics in the US

Firstly, we load in data on per capita incomes observed annually from 1929 to 2009 for the lower 48 US states. We use the example dataset in **libpysal** which was downloaded from US Bureau of Economic Analysis.

[8]:
```
import libpysal
f = libpysal.io.open(libpysal.examples.get_path("usjoin.csv"))
pci = np.array([f.by_col[str(y)] for y in range(1929,2010)])
print(pci.shape)
```

```
(81, 48)
```

The first row of the array is the per capita incomes for the 48 US states for the year 1929:

[9]:
```
print(pci[0, :])
```

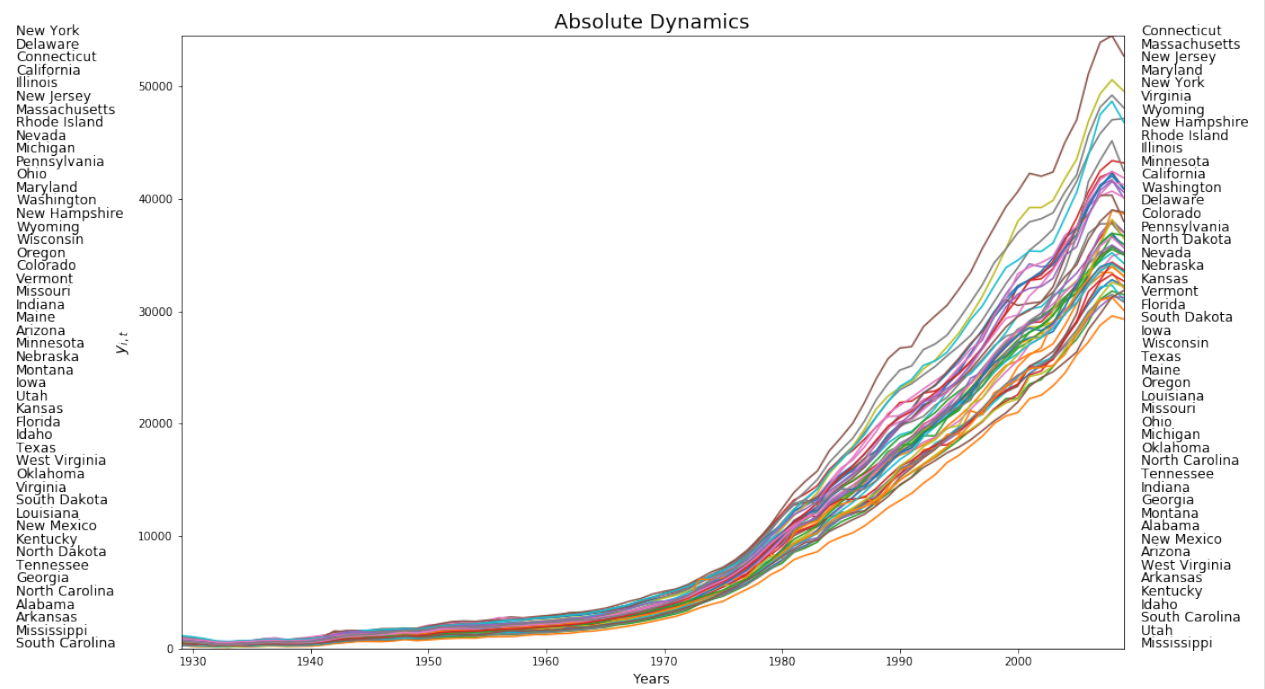
```
[ 323  600  310  991  634 1024 1032  518  347  507  948  607  581  532
  393  414  601  768  906  790  599  286  621  592  596  868  686  918
  410 1152  332  382  771  455  668  772  874  271  426  378  479  551
  634  434  741  460  673  675]
```

In order to apply the classic Markov approach to this series, we first have to discretize the distribution by defining our classes. There are many ways to do this including quantiles classification scheme, equal interval classification scheme, Fisher Jenks classification scheme, etc. For a list of classification methods, please refer to the pysal package **mapclassify**.

Here we will use the quintiles for each annual income distribution to define the classes. It should be noted that using quintiles for the pooled income distribution to define the classes will result in a different interpretation of the income dynamics. Quintiles for each annual income distribution (the former) will reveal more of relative income dynamics while those for the pooled income distribution (the latter) will provide insights in absolute dynamics.

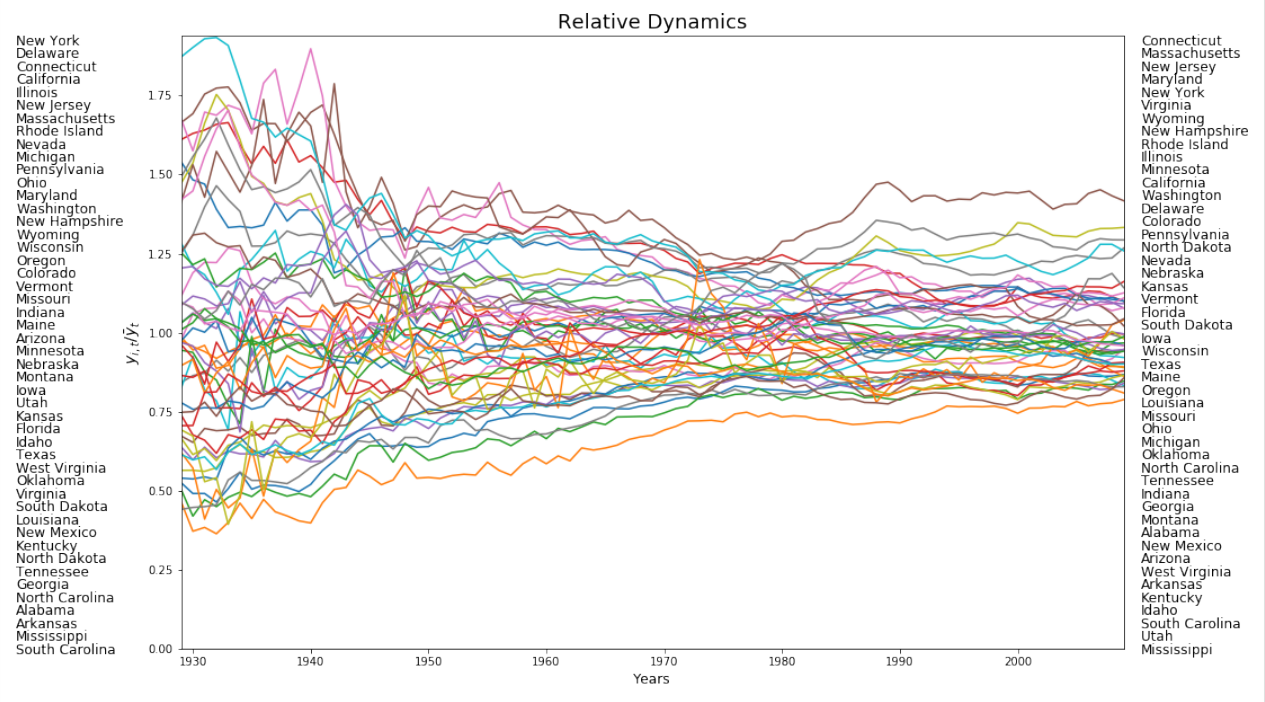
[10]:
```
import matplotlib.pyplot as plt
%matplotlib inline
years = range(1929,2010)
names = np.array(f.by_col("Name"))
order1929 = np.argsort(pci[0,:])
order2009 = np.argsort(pci[-1,:])
names1929 = names[order1929[::-1]]
names2009 = names[order2009[::-1]]
first_last = np.vstack((names1929,names2009))
from pylab import rcParams
rcParams['figure.figsize'] = 15,10
plt.plot(years,pci)
for i in range(48):
    plt.text(1915,54530-(i*1159), first_last[0][i],fontsize=12)
    plt.text(2010.5,54530-(i*1159), first_last[1][i],fontsize=12)
plt.xlim((years[0], years[-1]))
plt.ylim((0, 54530))
plt.ylabel(r"$y_{i,t}$",fontsize=14)
plt.xlabel('Years',fontsize=12)
plt.title('Absolute Dynamics',fontsize=18)
```

[10]:
```
Text(0.5,1,'Absolute Dynamics')
```



[11]:
```
years = range(1929,2010)
rpci= (pci.T / pci.mean(axis=1)).T
names = np.array(f.by_col("Name"))
order1929 = np.argsort(rpci[0,:])
order2009 = np.argsort(rpci[-1,:])
names1929 = names[order1929[::-1]]
names2009 = names[order2009[::-1]]
first_last = np.vstack((names1929,names2009))
from pylab import rcParams
rcParams['figure.figsize'] = 15,10
plt.plot(years,rpci)
for i in range(48):
    plt.text(1915,1.91-(i*0.041), first_last[0][i],fontsize=12)
    plt.text(2010.5,1.91-(i*0.041), first_last[1][i],fontsize=12)
plt.xlim((years[0], years[-1]))
plt.ylim((0, 1.94))
plt.ylabel(r"$y_{i,t}/\bar{y}_t$",fontsize=14)
plt.xlabel('Years',fontsize=12)
plt.title('Relative Dynamics',fontsize=18)
```

[11]:
```
Text(0.5,1,'Relative Dynamics')
```



[12]:
```
import mapclassify as mc
q5 = np.array([mc.Quantiles(y,k=5).yb for y in pci]).transpose()
print(q5[:, 0])
```

```
[0 2 0 4 2 4 4 1 0 1 4 2 2 1 0 1 2 3 4 4 2 0 2 2 2 4 3 4 0 4 0 0 3 1 3 3 4
 0 1 0 1 2 2 1 3 1 3 3]
```

```
/Users/weikang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:␣
↪FutureWarning: Using a non-tuple sequence for multidimensional indexing is␣
↪deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be␣
↪interpreted as an array index, `arr[np.array(seq)]`, which will result either in an␣
↪error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

[13]:
```
print(f.by_col("Name"))
```

```
['Alabama', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connecticut', 'Delaware
↪', 'Florida', 'Georgia', 'Idaho', 'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky
↪', 'Louisiana', 'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
↪'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire', 'New␣
↪Jersey', 'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohio',
↪'Oklahoma', 'Oregon', 'Pennsylvania', 'Rhode Island', 'South Carolina', 'South␣
↪Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington', 'West␣
↪Virginia', 'Wisconsin', 'Wyoming']
```

A number of things need to be noted here. First, we are relying on the classification methods in **mapclassify** for defining our quintiles. The class *Quantiles* uses quintiles ($k = 5$) as the default and will create an instance of this class that has multiple attributes, the one we are extracting in the first line is yb - the class id for each observation. The second thing to note is the transpose operator which gets our resulting array $q5$ in the proper structure required for use of Markov. Thus we see that the first spatial unit (Alabama with an income of 323) fell in the first quintile in 1929, while the last unit (Wyoming with an income of 675) fell in the fourth quintile.

So now we have a time series for each state of its quintile membership. For example, Colorado's quintile time series is:

[14]:
```
print(q5[4, :])
```

```
[2 3 2 2 3 2 2 3 2 2 2 2 2 2 2 3 2 3 2 3 2 3 3 3 2 2 3 3 3 3 3 3 3 3 3
 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4
 3 3 3 4 3 3 3]
```

indicating that it has occupied the 3rd, 4th and 5th quintiles in the distribution at the first 3 periods. To summarize the transition dynamics for all units, we instantiate a Markov object:

[15]:
```
m5 = giddy.markov.Markov(q5)
```

The number of transitions between any two quintile classes could be counted:

[16]:
```
print(m5.transitions)
```

```
[[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.]]
```

By assuming the first-order Markov property, time homogeneity, spatial homogeneity and spatial independence, a transition probability matrix could be estimated which holds for all the 48 US states across 1929-2010:

[17]:
```
print(m5.p)
```

```
[[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]]
```

The fact that each of the 5 diagonal elements is larger than 0.78 indicates a high stability of US regional income dynamics system.

Another very important feature of DMC model is the steady state distribution π (also called limiting distribution) defined as $\pi p = \pi$. The attribute $steady\_state$ gives π as follows:

[18]:
```
print(m5.steady_state)
```

```
[0.20774716 0.18725774 0.20740537 0.18821787 0.20937187]
```

If the distribution at t is a steady state distribution as shown above, then any distribution afterwards is the same distribution.

With the transition probability matrix in hand, we can estimate the first mean passage time which is the average number of steps to go from a state/class to another state for the first time:

[19]:
```
print(giddy.ergodic.fmpt(m5.p))
```

```
[[  4.81354357  11.50292712  29.60921231  53.38594954 103.59816743]
 [ 42.04774505   5.34023324  18.74455332  42.50023268  92.71316899]
 [ 69.25849753  27.21075248   4.82147603  25.27184624  75.43305672]
 [ 84.90689329  42.85914824  17.18082642   5.31299186  51.60953369]
 [ 98.41295543  56.36521038  30.66046735  14.21158356   4.77619083]]
```

Thus, for a state with income in the first quintile, it takes on average 11.5 years for it to first enter the second quintile, 29.6 to get to the third quintile, 53.4 years to enter the fourth, and 103.6 years to reach the richest quintile.

Regional context and Moran's Is

Thus far we have treated all the spatial units as independent to estimate the transition probabilities. This hides an implicit assumption: the movement of a spatial unit in the income distribution is independent of the movement of its neighbors or the position of the neighbors in the distribution. But what if spatial context matters??

We could plot the choropleth maps of per capita incomes of US states to get a first impression of the spatial distribution.

[20]:
```
import geopandas as gpd
import pandas as pd
```

[21]:
```
geo_table = gpd.read_file(libpysal.examples.get_path('us48.shp'))
income_table = pd.read_csv(libpysal.examples.get_path("usjoin.csv"))
complete_table = geo_table.merge(income_table,left_on='STATE_NAME',right_on='Name')
complete_table.head()
```

[21]:
```
     AREA  PERIMETER  STATE_  STATE_ID    STATE_NAME STATE_FIPS_x SUB_REGION  \
0  20.750     34.956       1         1    Washington           53    Pacific
1  45.132     34.527       2         2       Montana           30        Mtn
2   9.571     18.899       3         3         Maine           23      N Eng
3  21.874     21.353       4         4  North Dakota           38    W N Cen
4  22.598     22.746       5         5  South Dakota           46    W N Cen

  STATE_ABBR                                           geometry          Name  \
0         WA  (POLYGON ((-122.400749206543 48.22539520263672...    Washington
1         MT  POLYGON ((-111.4746322631836 44.70223999023438...       Montana
2         ME  (POLYGON ((-69.77778625488281 44.0740737915039...         Maine
3         ND  POLYGON ((-98.73005676269531 45.93829727172852...  North Dakota
4         SD  POLYGON ((-102.7879333496094 42.99532318115234...  South Dakota

   ...   2000   2001   2002   2003   2004   2005   2006   2007   2008   2009
0  ...  31528  32053  32206  32934  34984  35738  38477  40782  41588  40619
1  ...  22569  24342  24699  25963  27517  28987  30942  32625  33293  32699
2  ...  25623  27068  27731  28727  30201  30721  32340  33620  34906  35268
3  ...  25068  26118  26770  29109  29676  31644  32856  35882  39009  38672
4  ...  26115  27531  27727  30072  31765  32726  33320  35998  38188  36499

[5 rows x 92 columns]
```

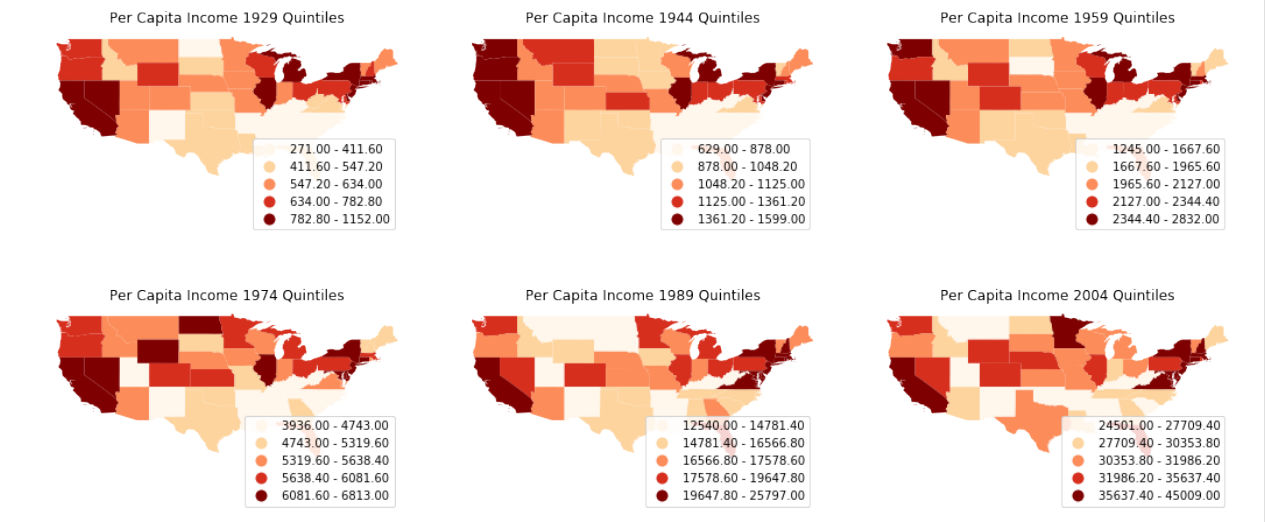
[22]:
```
index_year = range(1929,2010,15)
fig, axes = plt.subplots(nrows=2, ncols=3,figsize = (15,7))
for i in range(2):
    for j in range(3):
        ax = axes[i,j]
        complete_table.plot(ax=ax, column=str(index_year[i*3+j]), cmap='OrRd', scheme=
↪'quantiles', legend=True)
        ax.set_title('Per Capita Income %s Quintiles'%str(index_year[i*3+j]))
        ax.axis('off')
        leg = ax.get_legend()
        leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
plt.tight_layout()
```

```
/Users/weikang/anaconda3/lib/python3.6/site-packages/pysal/__init__.py:65:␣
↪VisibleDeprecationWarning: PySAL's API will be changed on 2018-12-31. The last␣
↪release made with this API is version 1.14.4. A preview of the next API version is␣
↪provided in the `pysal` 2.0 prelease candidate. The API changes and a guide on how␣
↪to change imports is provided at https://pysal.org/about
  ), VisibleDeprecationWarning)
```

(continues on next page)

(continued from previous page)

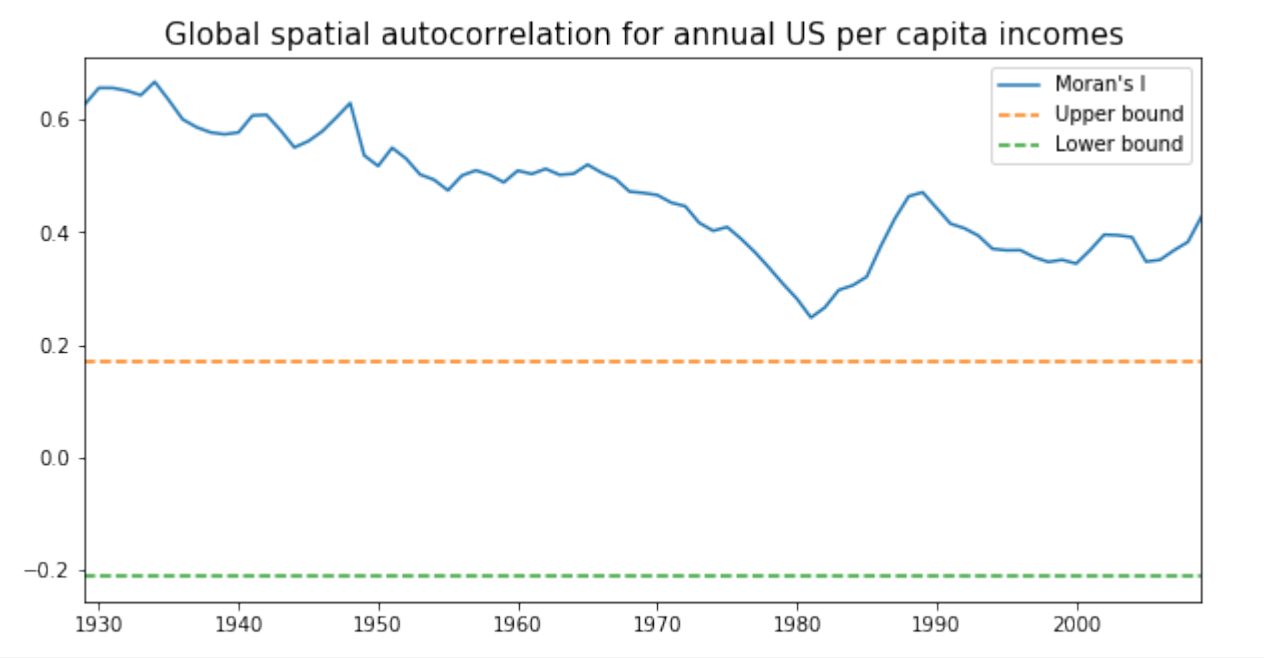
```
/Users/weikang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:␣
↪FutureWarning: Using a non-tuple sequence for multidimensional indexing is␣
↪deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be␣
↪interpreted as an array index, `arr[np.array(seq)]`, which will result either in an␣
↪error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



It is quite obvious that the per capita incomes are not randomly distributed: we could spot clusters in the mid-south, south-east and north-east. Let's proceed to calculate Moran's I, a widely used measure of global spatial autocorrelation, to aid the visual interpretation.

[23]:
```
from esda.moran import Moran
import matplotlib.pyplot as plt
%matplotlib inline
w = libpysal.io.open(libpysal.examples.get_path("states48.gal")).read()
w.transform = 'R'
mits = [Moran(cs, w) for cs in pci]
res = np.array([(mi.I, mi.EI, mi.seI_norm, mi.sim[974]) for mi in mits])
years = np.arange(1929,2010)
fig, ax = plt.subplots(nrows=1, ncols=1,figsize = (10,5) )
ax.plot(years, res[:,0], label='Moran\'s I')
#plot(years, res[:,1], label='E[I]')
ax.plot(years, res[:,1]+1.96*res[:,2], label='Upper bound',linestyle='dashed')
ax.plot(years, res[:,1]-1.96*res[:,2], label='Lower bound',linestyle='dashed')
ax.set_title("Global spatial autocorrelation for annual US per capita incomes",
↪fontdict={'fontsize':15})
ax.set_xlim([1929,2009])
ax.legend()
```

[23]:
```
<matplotlib.legend.Legend at 0x1a27ef4390>
```



From the above figure we could observe that Moran's I value was always positive and significant for each year across 1929-2009. In other words, US regional income series are not independent of each other and regional context could be important in shaping the regional income dynamics. However, the classic Markov approach is silent on this issue. We turn to the spatially explict Markov methods - **Spatial Markov** and **LISA Markov** - for an explicit incorporation of space in understanding US regional income distribution dynamics.

Spatial Markov

```
giddy.markov.Spatial_Markov(y, w, k=4, m=4, permutations=0, fixed=True,␣
↪discrete=False, cutoffs=None, lag_cutoffs=None, variable_name=None)
```

Spatial Markov is an extension to class Markov allowing for a more comprehensive analysis of the spatial dimensions of the transitional dynamics (Rey, 2001). Here, whether the transition probabilities are dependent on regional context is investigated and quantified. Rather than estimating one transition probability matrix, spatial Markov requires estimation of k transition probability matrices, each of which is conditional on the regional context at the preceding period. The regional context is usually formalized by spatial lag - the weighted average income level of neighbors:

$$z_{r,t} = \sum_{s=1}^{n} w_{r,s} y_{s,t}$$

where W is the spatial weight matrix and $w_{r,s}$ represents the weight that spatial unit s contributes to the local context of spatial unit r at time period t.

Similar to the construction of a **Markov** instance, we could create a **Spatial Markov** instance by utilizing the $Spatial\_Markov$ class in **giddy**. The only difference between the adoption of $Markov$ and $Spatial\_Markov$ class is that the latter accepts the original continuous income data while the former requires a pre-classification/discretization. In other words, here we do not need to apply the classification methods in **map-classify** as we did earlier. In fact, the **Spatial Markov** class nested the quantile classification methods and all we need to do is set the desired number of classes k when creating the $Spatial\_Markov$ instance. Here, we set $k = 5$ (quintile classes) as before.

Different from before, quintiles are defined for the pooled relative incomes (by standardizing by each period by the mean). This is achieved by setting the parameter $fixed$ as *True*.

[24]:
```
giddy.markov.Spatial_Markov?
```

[25]:
```
sm = giddy.markov.Spatial_Markov(rpci.T, w, fixed = True, k = 5,m=5) # spatial_markov␣
↪instance o
```

```
/Users/weikang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:␣
↪FutureWarning: Using a non-tuple sequence for multidimensional indexing is␣
↪deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be␣
↪interpreted as an array index, `arr[np.array(seq)]`, which will result either in an␣
↪error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

We can next examine the global transition probability matrix for relative incomes.

[26]:
```
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 Spatial Markov allows us to compare the global transition dynamics to those conditioned on regional context. More specifically, the transition dynamics are split across economies who have spatial lags in different quintiles at the preceding year. In our example we have 5 classes, so 5 different conditioned transition probability matrices are estimated - P(LAG0), P(LAG1), P(LAG2), P(LAG3), and P(LAG4).

[27]:
```
sm.summary()
```

```
------------------------------------------------------------
                    Spatial Markov Test
------------------------------------------------------------
Number of classes: 5
Number of transitions: 3840
Number of regimes: 5
Regime names: LAG0, LAG1, LAG2, LAG3, LAG4
------------------------------------------------------------
   Test                    LR                Chi-2
  Stat.                170.659              200.624
    DOF                     60                  60
p-value                  0.000               0.000
------------------------------------------------------------
P(H0)           C0         C1         C2         C3         C4
     C0      0.915      0.075      0.009      0.001      0.000
     C1      0.066      0.827      0.105      0.001      0.001
     C2      0.005      0.103      0.794      0.095      0.003
     C3      0.000      0.009      0.094      0.849      0.048
     C4      0.000      0.000      0.000      0.062      0.938
------------------------------------------------------------
P(LAG0)         C0         C1         C2         C3         C4
     C0      0.963      0.030      0.006      0.000      0.000
     C1      0.060      0.832      0.107      0.000      0.000
     C2      0.000      0.140      0.740      0.120      0.000
     C3      0.000      0.036      0.321      0.571      0.071
     C4      0.000      0.000      0.000      0.167      0.833
------------------------------------------------------------
P(LAG1)         C0         C1         C2         C3         C4
     C0      0.798      0.168      0.034      0.000      0.000
```

(continues on next page)

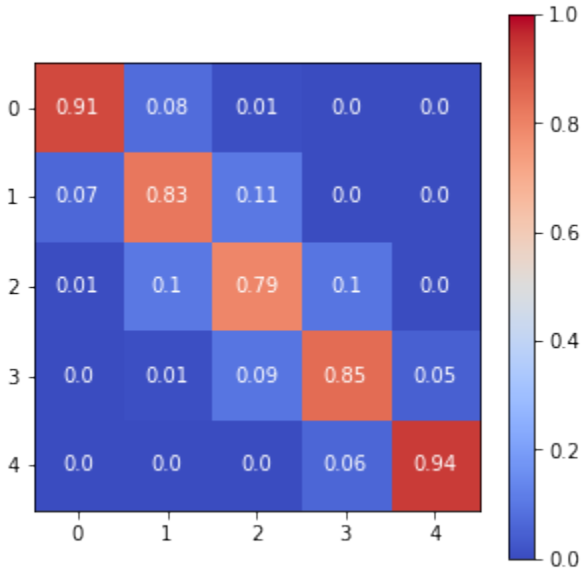
(continued from previous page)

```
     C1      0.075      0.882      0.042      0.000      0.000
     C2      0.005      0.070      0.866      0.059      0.000
     C3      0.000      0.000      0.064      0.902      0.034
     C4      0.000      0.000      0.000      0.194      0.806
------------------------------------------------------------
P(LAG2)          C0         C1         C2         C3         C4
     C0      0.847      0.153      0.000      0.000      0.000
     C1      0.081      0.789      0.129      0.000      0.000
     C2      0.005      0.098      0.793      0.098      0.005
     C3      0.000      0.000      0.094      0.871      0.035
     C4      0.000      0.000      0.000      0.102      0.898
------------------------------------------------------------
P(LAG3)          C0         C1         C2         C3         C4
     C0      0.885      0.098      0.000      0.016      0.000
     C1      0.039      0.814      0.140      0.000      0.008
     C2      0.005      0.094      0.777      0.119      0.005
     C3      0.000      0.023      0.129      0.754      0.094
     C4      0.000      0.000      0.000      0.097      0.903
------------------------------------------------------------
P(LAG4)          C0         C1         C2         C3         C4
     C0      0.333      0.667      0.000      0.000      0.000
     C1      0.048      0.774      0.161      0.016      0.000
     C2      0.011      0.161      0.747      0.080      0.000
     C3      0.000      0.010      0.062      0.896      0.031
     C4      0.000      0.000      0.000      0.024      0.976
------------------------------------------------------------
```

Visualize the (spatial) Markov transition probability matrix

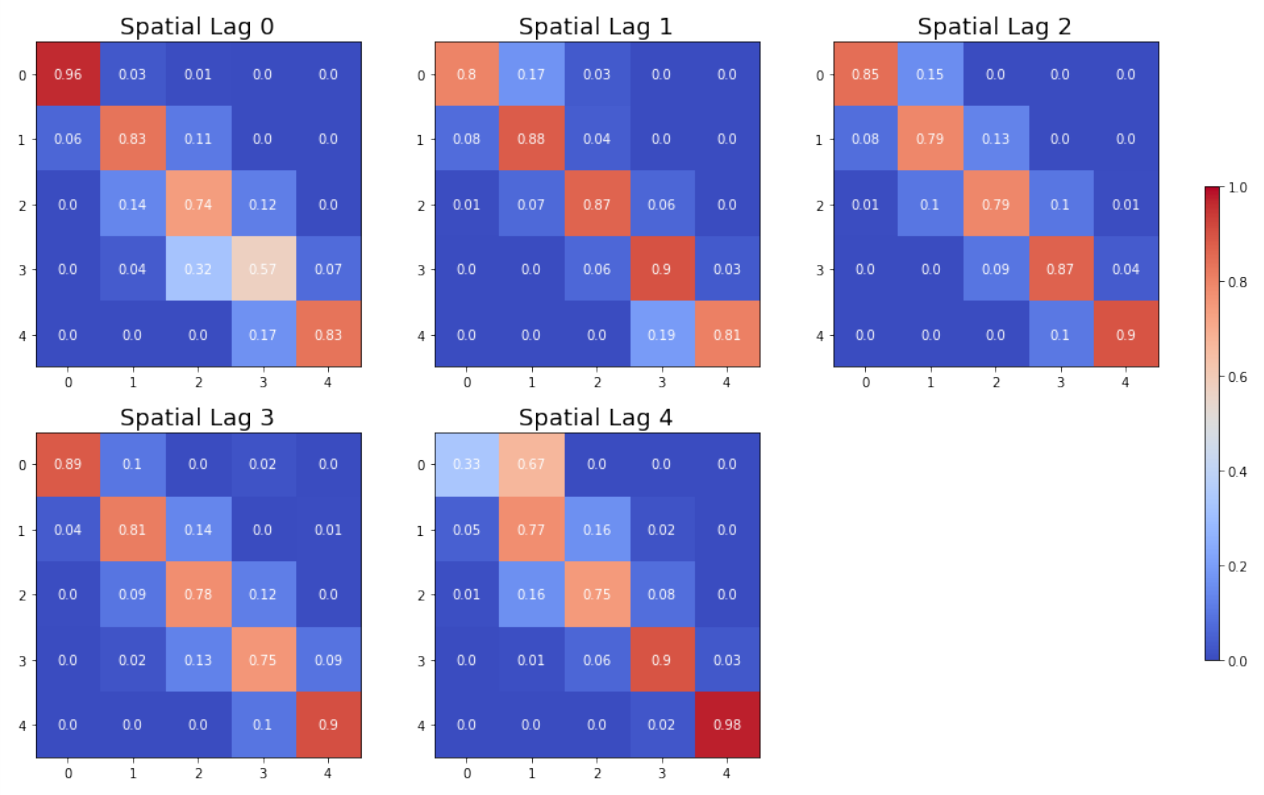
[28]:
```
fig, ax = plt.subplots(figsize = (5,5))
im = ax.imshow(sm.p,cmap = "coolwarm",vmin=0, vmax=1)
# Loop over data dimensions and create text annotations.
for i in range(len(sm.p)):
    for j in range(len(sm.p)):
        text = ax.text(j, i, round(sm.p[i, j], 2),
                       ha="center", va="center", color="w")
ax.figure.colorbar(im, ax=ax)
```

[28]:
```
<matplotlib.colorbar.Colorbar at 0x1a2836c588>
```



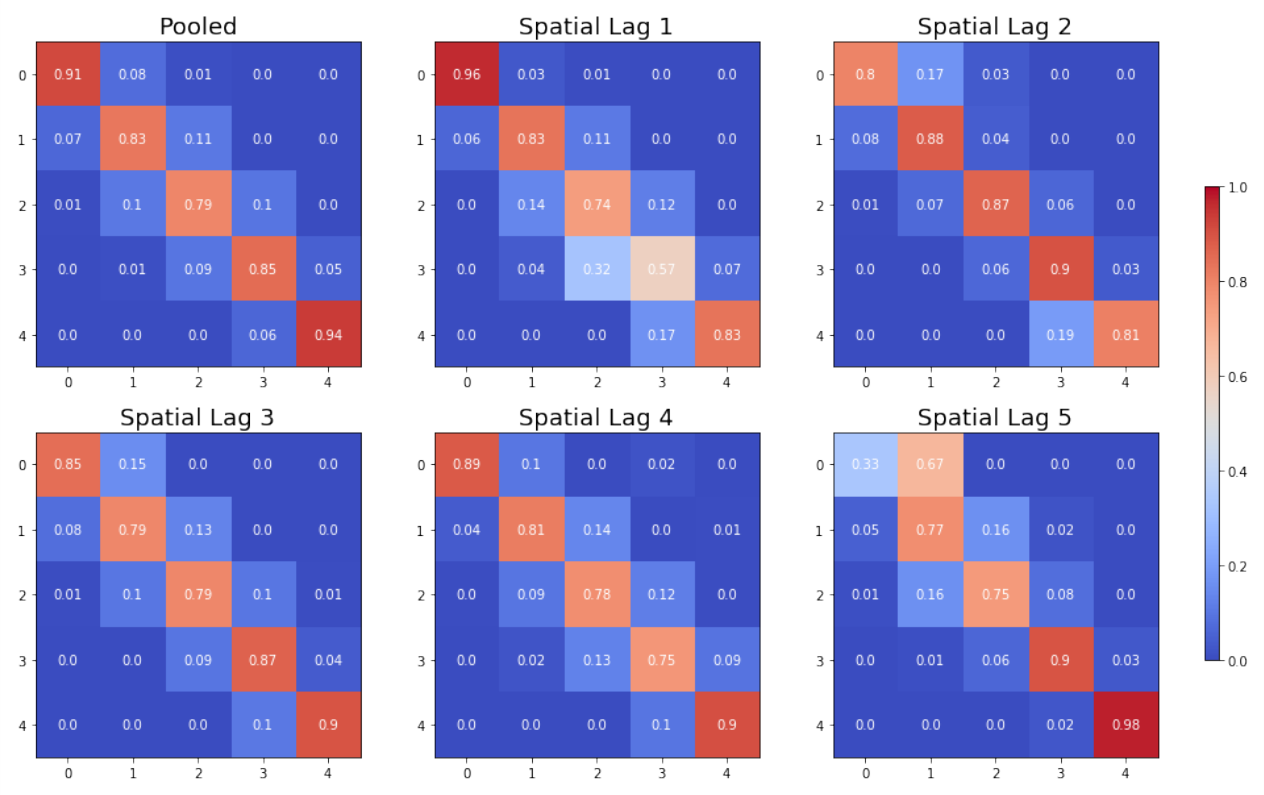
[29]:
```
fig, axes = plt.subplots(2,3,figsize = (15,10))

for i in range(2):
    for j in range(3):
        ax = axes[i,j]
        if i==1 and j==2:
            ax.axis('off')
            continue
        # Loop over data dimensions and create text annotations.
        p_temp = sm.P[i*3+j]
        for x in range(len(p_temp)):
            for y in range(len(p_temp)):
                text = ax.text(y, x, round(p_temp[x, y], 2),
                               ha="center", va="center", color="w")
        im = ax.imshow(p_temp,cmap = "coolwarm",vmin=0, vmax=1)
        ax.set_title("Spatial Lag %d"%(i*3+j),fontsize=18)
fig.subplots_adjust(right=0.92)
cbar_ax = fig.add_axes([0.95, 0.228, 0.01, 0.5])
fig.colorbar(im, cax=cbar_ax)
```

[29]:
```
<matplotlib.colorbar.Colorbar at 0x1a286705f8>
```



[30]:
```
fig, axes = plt.subplots(2,3,figsize = (15,10))
for i in range(2):
    for j in range(3):
        ax = axes[i,j]
        if i==0 and j==0:
            p_temp = sm.p
            im = ax.imshow(p_temp,cmap = "coolwarm",vmin=0, vmax=1)
            ax.set_title("Pooled",fontsize=18)
        else:
            p_temp = sm.P[i*3+j-1]
            im = ax.imshow(p_temp,cmap = "coolwarm",vmin=0, vmax=1)
            ax.set_title("Spatial Lag %d"%(i*3+j),fontsize=18)
        for x in range(len(p_temp)):
            for y in range(len(p_temp)):
                text = ax.text(y, x, round(p_temp[x, y], 2),
                               ha="center", va="center", color="w")

fig.subplots_adjust(right=0.92)
cbar_ax = fig.add_axes([0.95, 0.228, 0.01, 0.5])
fig.colorbar(im, cax=cbar_ax)
#fig.savefig('spatial_markov_us.png', dpi = 300)
```

[30]:
```
<matplotlib.colorbar.Colorbar at 0x1a28f9d198>
```



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 0.977 if their neighbors are in the 5th quintile, but if their neighbors are in the 4th quintile this drops to 0.903.

We can also explore the different steady state distributions implied by these different transition probabilities:

[31]:
```
print(sm.S)
```

```
[[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 ]]
```

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. And, finally the spatially conditional first mean passage times:

[32]:
```
print(sm.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 ]
  [ 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]]
```

(continues on next page)

(continued from previous page)

```
[[  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]]]
```

States in the first income quintile with neighbors in the first quintile return to the first quintile after 2.298 years, after leaving the first quintile. They enter the fourth quintile 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.

Tests for this conditional type of spatial dependence include Likelihood Ratio (LR) test and χ^2 test (Bickenbach and Bode, 2003) as well as a test based on information theory (Kullback et al., 1962). For the first two tests, we could proceed as follows to acquire their statistics, DOF and p-value.

[33]:
```
giddy.markov.Homogeneity_Results(sm.T).summary()
```

```
-------------------------------------------------
             Markov Homogeneity Test
-------------------------------------------------
Number of classes: 5
Number of transitions: 3840
Number of regimes: 5
Regime names: 0, 1, 2, 3, 4
-------------------------------------------------
   Test                   LR                Chi-2
  Stat.               170.659              200.624
    DOF                    60                  60
p-value                 0.000               0.000
-------------------------------------------------
P(H0)         0         1         2         3         4
    0     0.915     0.075     0.009     0.001     0.000
    1     0.066     0.827     0.105     0.001     0.001
    2     0.005     0.103     0.794     0.095     0.003
    3     0.000     0.009     0.094     0.849     0.048
    4     0.000     0.000     0.000     0.062     0.938
-------------------------------------------------
P(0)          0         1         2         3         4
    0     0.963     0.030     0.006     0.000     0.000
    1     0.060     0.832     0.107     0.000     0.000
    2     0.000     0.140     0.740     0.120     0.000
    3     0.000     0.036     0.321     0.571     0.071
    4     0.000     0.000     0.000     0.167     0.833
-------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
P(1)          0         1         2         3         4
    0     0.798     0.168     0.034     0.000     0.000
    1     0.075     0.882     0.042     0.000     0.000
    2     0.005     0.070     0.866     0.059     0.000
    3     0.000     0.000     0.064     0.902     0.034
    4     0.000     0.000     0.000     0.194     0.806
--------------------------------------------------
P(2)          0         1         2         3         4
    0     0.847     0.153     0.000     0.000     0.000
    1     0.081     0.789     0.129     0.000     0.000
    2     0.005     0.098     0.793     0.098     0.005
    3     0.000     0.000     0.094     0.871     0.035
    4     0.000     0.000     0.000     0.102     0.898
--------------------------------------------------
P(3)          0         1         2         3         4
    0     0.885     0.098     0.000     0.016     0.000
    1     0.039     0.814     0.140     0.000     0.008
    2     0.005     0.094     0.777     0.119     0.005
    3     0.000     0.023     0.129     0.754     0.094
    4     0.000     0.000     0.000     0.097     0.903
--------------------------------------------------
P(4)          0         1         2         3         4
    0     0.333     0.667     0.000     0.000     0.000
    1     0.048     0.774     0.161     0.016     0.000
    2     0.011     0.161     0.747     0.080     0.000
    3     0.000     0.010     0.062     0.896     0.031
    4     0.000     0.000     0.000     0.024     0.976
--------------------------------------------------
```

From the above summary table, we can observe that the observed LR test statistic is 170.659 and the observed χ^2 test statistic is 200.624. Their p-values are 0.000, which leads to the rejection of the null hypothesis of conditional spatial independence.

For the last (information theory-based) test, we call the function $kullback$. The result is consistent with LR and χ^2 tests. As shown below, the observed test statistic is 230.03 and its p-value is 2.22e-16, leading to the rejection of the null.

[34]:
```
print(giddy.markov.kullback(sm.T))
```

```
{'Conditional homogeneity': 230.0266246375395, 'Conditional homogeneity dof': 80,
↪'Conditional homogeneity pvalue': 2.220446049250313e-16}
```

LISA Markov

```
giddy.markov.LISA_Markov(self, y, w, permutations=0, significance_level=0.05, geoda_
↪quads=False)
```

The Spatial Markov conditions the transitions on the value of the spatial lag for an observation at the beginning of the transition period. An alternative approach to spatial dynamics is to consider the joint transitions of an observation and its spatial lag in the distribution. By exploiting the form of the static LISA and embedding it in a dynamic context we develop the LISA Markov in which the states of the chain are defined as the four quadrants in the Moran scatter plot, namely, HH(=1), LH(=2), LL(=3), HL(=4). Continuing on with our US example, the LISA transitions are:

[35]:
```
lm = giddy.markov.LISA_Markov(pci.T, w)
print(lm.classes)
```

```
[1 2 3 4]
```

The LISA transitions are:

[36]:
```
print(lm.transitions)
```

```
[[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]]
```

and the estimated transition probability matrix is:

[37]:
```
print(lm.p)
```

```
[[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]]
```

The diagonal elements indicate the staying probabilities and we see that there is greater mobility for observations in quadrants 2 (LH) and 4 (HL) than 1 (HH) and 3 (LL).

The implied long run steady state distribution of the chain is:

[38]:
```
print(lm.steady_state)
```

```
[0.28561505 0.14190226 0.40493672 0.16754598]
```

again reflecting the dominance of quadrants 1 and 3 (positive autocorrelation). The first mean passage time for the LISAs is:

[39]:
```
print(giddy.ergodic.fmpt(lm.p))
```

```
[[ 3.50121609 37.93025465 40.55772829 43.17412009]
 [31.72800152  7.04710419 28.68182751 49.91485137]
 [52.44489385 47.42097495  2.46952168 43.75609676]
 [38.76794022 51.51755827 26.31568558  5.96851095]]
```

To test for dependence between the dynamics of the region and its neighbors, we turn to χ^2 test of independence. Here, the χ^2 statistic, its p-value and degrees of freedom can be obtained from the attribute chi_2. As the p-value is 0.0, the null of independence is clearly rejected.

[40]:
```
print(lm.chi_2)
```

```
(1058.2079036003051, 0.0, 9)
```

Next steps

- Simulation/prediction of Markov chain and spatial Markov chain

References

- Rey, S. J. 2001. "Spatial Empirics for Economic Growth and Convergence." Geographical Analysis 33 (3). Wiley Online Library: 195–214.

- Bickenbach, F., and E. Bode. 2003. "Evaluating the Markov Property in Studies of Economic Convergence." International Regional Science Review 26 (3): 363–92.
- Kullback, S., M. Kupperman, and H. H. Ku. 1962. "Tests for Contingency Tables and Markov Chains." Technometrics: A Journal of Statistics for the Physical, Chemical, and Engineering Sciences 4 (4). JSTOR: 573–608.

doc/notebooks/MarkovBasedMethods.ipynb ends here.

The following section was generated from doc/notebooks/RankMarkov.ipynb

1.2.2 Full Rank Markov and Geographic Rank Markov

Author: Wei Kang weikang9009@gmail.com

[1]:
```
import libpysal as ps
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pandas as pd
import geopandas as gpd
```

Full Rank Markov

[2]:
```
from giddy.markov import FullRank_Markov
```

[3]:
```
income_table = pd.read_csv(ps.examples.get_path("usjoin.csv"))
income_table.head()
```

[3]:

	Name	STATE_FIPS	1929	1930	1931	1932	1933	1934	1935	1936	\
0	Alabama	1	323	267	224	162	166	211	217	251	
1	Arizona	4	600	520	429	321	308	362	416	462	
2	Arkansas	5	310	228	215	157	157	187	207	247	
3	California	6	991	887	749	580	546	603	660	771	
4	Colorado	8	634	578	471	354	353	368	444	542	

	...	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
0	...	23471	24467	25161	26065	27665	29097	30634	31988	32819	32274
1	...	25578	26232	26469	27106	28753	30671	32552	33470	33445	32077
2	...	22257	23532	23929	25074	26465	27512	29041	31070	31800	31493
3	...	32275	32750	32900	33801	35663	37463	40169	41943	42377	40902
4	...	32949	34228	33963	34092	35543	37388	39662	41165	41719	40093

[5 rows x 83 columns]

[4]:
```
pci = income_table[list(map(str,range(1929,2010)))].values
pci
```

[4]:
```
array([[  323,   267,   224, ..., 31988, 32819, 32274],
       [  600,   520,   429, ..., 33470, 33445, 32077],
       [  310,   228,   215, ..., 31070, 31800, 31493],
       ...,
       [  460,   408,   356, ..., 29769, 31265, 31843],
       [  673,   588,   469, ..., 35839, 36594, 35676],
       [  675,   585,   476, ..., 43453, 45177, 42504]])
```

[5]:
```
m = FullRank_Markov(pci)
m.ranks
```

[5]:
```
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]])
```

[6]:
```
m.transitions
```

[6]:
```
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.]])
```

Full rank Markov transition probability matrix

[7]:
```
m.p
```

[7]:
```
array([[0.825 , 0.0625, 0.0625, ..., 0.    , 0.    , 0.    ],
       [0.1   , 0.6375, 0.1125, ..., 0.    , 0.    , 0.    ],
       [0.025 , 0.1625, 0.55  , ..., 0.    , 0.    , 0.    ],
       ...,
       [0.    , 0.    , 0.    , ..., 0.5   , 0.2125, 0.    ],
       [0.    , 0.    , 0.    , ..., 0.1875, 0.675 , 0.025 ],
       [0.    , 0.    , 0.    , ..., 0.025 , 0.0125, 0.9625]])
```

Full rank first mean passage times

[8]:
```
m.fmpt
```

[8]:
```
array([[  48.        ,   87.96280048,   68.1089084 , ...,  443.76689275,
         518.31000749, 1628.59025557],
       [ 225.92564594,   48.        ,   78.75804364, ...,  440.0173313 ,
         514.56045127, 1624.84070661],
       [ 271.55443692,  102.484092  ,   48.        , ...,  438.93288204,
         513.47599512, 1623.75624059],
       ...,
       [ 727.11189921,  570.15910508,  546.61934646, ...,   48.        ,
         117.41906375, 1278.96860316],
       [ 730.40467469,  573.45179415,  549.91216045, ...,   49.70722573,
          48.        , 1202.06279368],
       [ 754.8761577 ,  597.92333477,  574.38361779, ...,   43.23574191,
         104.9460425 ,   48.        ]])
```

[9]:
```
m.sojourn_time
```

[9]:
```
array([ 5.71428571,  2.75862069,  2.22222222,  1.77777778,  1.66666667,
        1.73913043,  1.53846154,  1.53846154,  1.53846154,  1.42857143,
        1.42857143,  1.56862745,  1.53846154,  1.40350877,  1.29032258,
        1.21212121,  1.31147541,  1.37931034,  1.29032258,  1.25      ,
        1.15942029,  1.12676056,  1.25      ,  1.17647059,  1.19402985,
        1.08108108,  1.19402985,  1.25      ,  1.25      ,  1.14285714,
```

(continues on next page)

(continued from previous page)

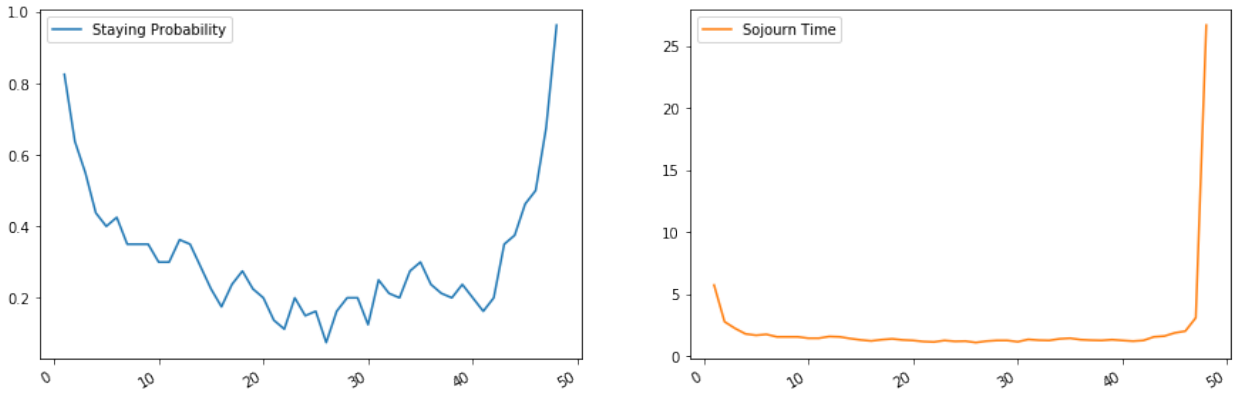
```
       1.33333333,  1.26984127,  1.25      ,  1.37931034,  1.42857143,
       1.31147541,  1.26984127,  1.25      ,  1.31147541,  1.25      ,
       1.19402985,  1.25      ,  1.53846154,  1.6       ,  1.86046512,
       2.        ,  3.07692308, 26.66666667])
```

[10]:
```
df_fullrank = pd.DataFrame(np.c_[m.p.diagonal(),m.sojourn_time], columns=["Staying␣
↪Probability","Sojourn Time"], index = np.arange(m.p.shape[0])+1)
df_fullrank.head()
```

[10]:
```
   Staying Probability  Sojourn Time
1               0.8250      5.714286
2               0.6375      2.758621
3               0.5500      2.222222
4               0.4375      1.777778
5               0.4000      1.666667
```

[11]:
```
df_fullrank.plot(subplots=True, layout=(1,2), figsize=(15,5))
```

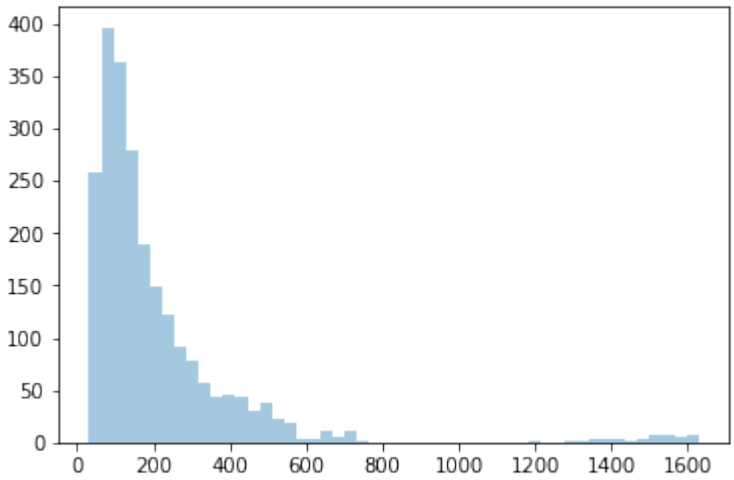
[11]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a2ad213c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1a2ad64668>]],
      dtype=object)
```



[12]:
```
sns.distplot(m.fmpt.flatten(),kde=False)
```

```
/Users/weikang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:␣
↪FutureWarning: Using a non-tuple sequence for multidimensional indexing is␣
↪deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be␣
↪interpreted as an array index, `arr[np.array(seq)]`, which will result either in an␣
↪error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2ae70908>
```



Geographic Rank Markov

[13]:
```
from giddy.markov import GeoRank_Markov, Markov, sojourn_time
gm = GeoRank_Markov(pci)
```

[21]:
```
gm.transitions
```

[21]:
```
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.]])
```

[24]:
```
gm.p
```

[24]:
```
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 ]])
```

[28]:
```
gm.sojourn_time[:10]
```

[28]:
```
array([1.9047619 , 1.23076923, 2.22222222, 1.73913043, 1.15942029,
       3.80952381, 1.70212766, 1.25      , 1.31147541, 1.11111111])
```

[14]:
```
gm.sojourn_time
```

[14]:
```
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,
```

(continues on next page)

(continued from previous page)

```
        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])
```

[15]:
```
gm.fmpt
```

[15]:
```
array([[ 48.        ,  63.35532038,  92.75274652, ...,  82.47515731,
         71.01114491,  68.65737127],
       [108.25928005,  48.        , 127.99032986, ...,  92.03098299,
         63.36652935,  61.82733039],
       [ 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.        ]])
```

[16]:
```
income_table["geo_sojourn_time"] = gm.sojourn_time
i = 0
for state in income_table["Name"]:
    income_table["geo_fmpt_to_" + state] = gm.fmpt[:,i]
    income_table["geo_fmpt_from_" + state] = gm.fmpt[i,:]
    i = i + 1
income_table.head()
```

[16]:

	Name	STATE_FIPS	1929	1930	1931	1932	1933	1934	1935	1936	\
0	Alabama	1	323	267	224	162	166	211	217	251	
1	Arizona	4	600	520	429	321	308	362	416	462	
2	Arkansas	5	310	228	215	157	157	187	207	247	
3	California	6	991	887	749	580	546	603	660	771	
4	Colorado	8	634	578	471	354	353	368	444	542	

	...	geo_fmpt_to_Virginia	geo_fmpt_from_Virginia	\
0	...	72.186055	109.828532	
1	...	67.544447	60.838807	
2	...	73.650943	129.533691	
3	...	71.377700	111.644884	
4	...	69.627179	57.106339	

	geo_fmpt_to_Washington	geo_fmpt_from_Washington	\
0	82.994754	118.769984	
1	76.090895	66.729262	
2	84.071211	138.692513	
3	62.230417	97.908341	
4	66.353930	52.229230	

	geo_fmpt_to_West Virginia	geo_fmpt_from_West Virginia	\
0	82.475157	93.310747	
1	92.030983	62.476705	
2	73.845952	105.806341	
3	104.922271	121.670243	
4	98.797636	66.464398	

(continues on next page)

(continued from previous page)

```
   geo_fmpt_to_Wisconsin  geo_fmpt_from_Wisconsin  geo_fmpt_to_Wyoming  \
0              71.011145               113.652781            68.657371
1              63.366529                61.198703            61.827330
2              72.246827               133.579917            69.774972
3              69.368408               110.668388            59.998457
4              60.762589                52.324565            55.559020

   geo_fmpt_from_Wyoming
0             114.718948
1              63.401978
2             134.733817
3             105.965215
4              53.872702

[5 rows x 180 columns]
```

[17]:
```
geo_table = gpd.read_file(ps.examples.get_path('us48.shp'))
# income_table = pd.read_csv(libpysal.examples.get_path("usjoin.csv"))
complete_table = geo_table.merge(income_table,left_on='STATE_NAME',right_on='Name')
complete_table.head()
```

[17]:
```
     AREA  PERIMETER  STATE_  STATE_ID    STATE_NAME STATE_FIPS_x SUB_REGION  \
0  20.750     34.956       1         1    Washington           53    Pacific
1  45.132     34.527       2         2       Montana           30        Mtn
2   9.571     18.899       3         3         Maine           23      N Eng
3  21.874     21.353       4         4  North Dakota           38    W N Cen
4  22.598     22.746       5         5  South Dakota           46    W N Cen

  STATE_ABBR                                           geometry          Name  \
0         WA  (POLYGON ((-122.400749206543 48.22539520263672...    Washington
1         MT  POLYGON ((-111.4746322631836 44.70223999023438...       Montana
2         ME  (POLYGON ((-69.77778625488281 44.0740737915039...         Maine
3         ND  POLYGON ((-98.73005676269531 45.93829727172852...  North Dakota
4         SD  POLYGON ((-102.7879333496094 42.99532318115234...  South Dakota

           ...            geo_fmpt_to_Virginia  geo_fmpt_from_Virginia  \
0          ...                       71.663055               73.756804
1          ...                       69.918931               59.067897
2          ...                       69.431862               53.872836
3          ...                       69.441690               56.526347
4          ...                       68.229894               61.548209

   geo_fmpt_to_Washington  geo_fmpt_from_Washington  \
0               48.000000                 48.000000
1               76.184088                 64.710823
2               77.512381                 62.862378
3               76.659646                 62.823668
4               78.886304                 68.794083

   geo_fmpt_to_West Virginia  geo_fmpt_from_West Virginia  \
0                 101.592400                    81.692586
1                  90.781850                    58.795201
2                  87.734760                    54.244823
3                  85.031218                    49.511240
4                  88.192659                    55.754109
```

(continues on next page)

(continued from previous page)

```
   geo_fmpt_to_Wisconsin  geo_fmpt_from_Wisconsin  geo_fmpt_to_Wyoming  \
0              65.219124                70.701226            53.126177
1              63.455248                58.975522            60.881954
2              66.257807                56.905741            61.978506
3              67.362718                58.717458            64.386382
4              66.187694                63.802359            64.336311

   geo_fmpt_from_Wyoming
0              64.476985
1              60.553000
2              58.336426
3              59.728719
4              65.070022

[5 rows x 189 columns]
```

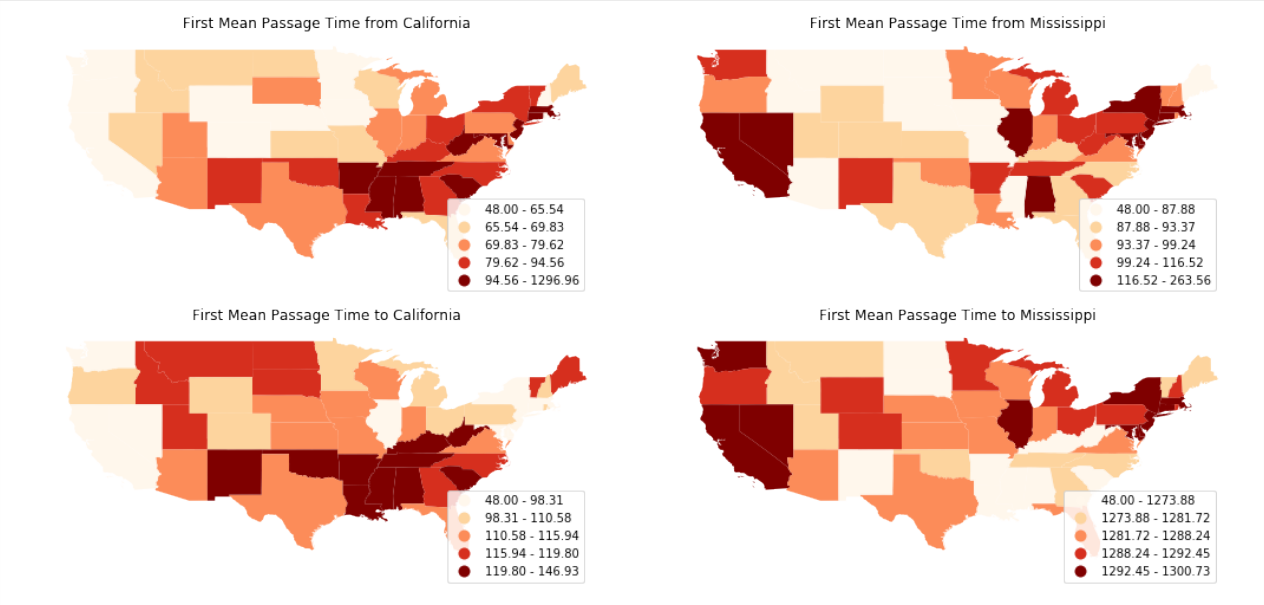
[18]:
```
complete_table.columns
```

[18]:
```
Index(['AREA', 'PERIMETER', 'STATE_', 'STATE_ID', 'STATE_NAME', 'STATE_FIPS_x',
       'SUB_REGION', 'STATE_ABBR', 'geometry', 'Name',
       ...
       'geo_fmpt_to_Virginia', 'geo_fmpt_from_Virginia',
       'geo_fmpt_to_Washington', 'geo_fmpt_from_Washington',
       'geo_fmpt_to_West Virginia', 'geo_fmpt_from_West Virginia',
       'geo_fmpt_to_Wisconsin', 'geo_fmpt_from_Wisconsin',
       'geo_fmpt_to_Wyoming', 'geo_fmpt_from_Wyoming'],
      dtype='object', length=189)
```

Visualizing first mean passage time from/to California/Mississippi:

[19]:
```
fig, axes = plt.subplots(nrows=2, ncols=2,figsize = (15,7))
target_states = ["California","Mississippi"]
directions = ["from","to"]
for i, direction in enumerate(directions):
    for j, target in enumerate(target_states):
        ax = axes[i,j]
        col = direction+"_"+target
        complete_table.plot(ax=ax,column = "geo_fmpt_"+ col,cmap='OrRd',
                    scheme='quantiles', legend=True)
        ax.set_title("First Mean Passage Time "+direction+" "+target)
        ax.axis('off')
        leg = ax.get_legend()
        leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
plt.tight_layout()
```

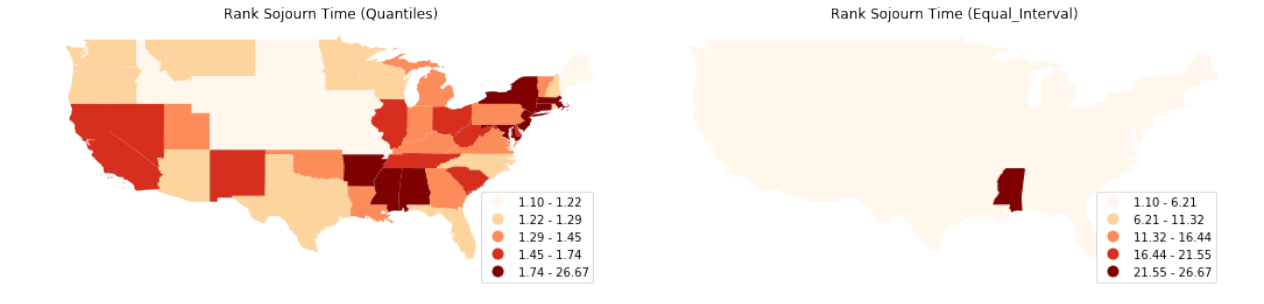
```
/Users/weikang/anaconda3/lib/python3.6/site-packages/pysal/__init__.py:65:␣
↪VisibleDeprecationWarning: PySAL's API will be changed on 2018-12-31. The last␣
↪release made with this API is version 1.14.4. A preview of the next API version is␣
↪provided in the `pysal` 2.0 prelease candidate. The API changes and a guide on how␣
↪to change imports is provided at https://migrating.pysal.org
  ), VisibleDeprecationWarning)
/Users/weikang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:␣
↪FutureWarning: Using a non-tuple sequence for multidimensional indexing is␣
↪deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be␣
↪interpreted as an array index, `arr[np.array(seq)]`, which will result either in an␣
↪error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Visualizing sojourn time for each US state:

[20]:
```
fig, axes = plt.subplots(nrows=1, ncols=2,figsize = (15,7))
schemes = ["Quantiles","Equal_Interval"]
for i, scheme in enumerate(schemes):
    ax = axes[i]
    complete_table.plot(ax=ax,column = "geo_sojourn_time",cmap='OrRd',
                scheme=scheme, legend=True)
    ax.set_title("Rank Sojourn Time ("+scheme+")")
    ax.axis('off')
    leg = ax.get_legend()
    leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
plt.tight_layout()
```

```
/Users/weikang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:␣
↪FutureWarning: Using a non-tuple sequence for multidimensional indexing is␣
↪deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be␣
↪interpreted as an array index, `arr[np.array(seq)]`, which will result either in an␣
↪error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



doc/notebooks/RankMarkov.ipynb ends here.

The following section was generated from `doc/notebooks/MobilityMeasures.ipynb` .....

1.2.3 Measures of Income Mobility

Author: Wei Kang weikang9009@gmail.com, Serge Rey sjsrey@gmail.com

Income mobility could be viewed as a reranking pheonomenon where regions switch income positions while it could also be considered to be happening as long as regions move away from the previous income levels. The former is named absolute mobility and the latter relative mobility.

This notebook introduces how to estimate income mobility measures from longitudinal income data using methods in **giddy**. Currently, five summary mobility estimators are implemented in **giddy.mobility**. All of them are Markov-based, meaning that they are closely related to the discrete Markov Chains methods introduced in Markov Based Methods notebook. More specifically, each of them is derived from a transition probability matrix P. Whether the final estimate is absolute or reletive mobility depends on how the original continuous income data are discretized.

The five Markov-based summary measures of mobility (Formby et al., 2004) are listed below:

Num	Measures	Symbol
1	$$M_P(P) = \frac{m - \sum_{i=1}^{m} p_{ii}}{m-1}$$	P
2	$$M_D(P) = 1-	det(P)
3	$$M_{L2}(P)=1-	λ_2
4	$$M_{B1}(P) = \frac{m - m\sum_{i=1}^{m} \pi_i P_{ii}}{m-1}$$	B1
5	$$M_{B2}(P)=:nbsphinx-math:*frac{1}{m-1}* $\sum$ *{i=1}^m :nbsphinx-math:‘sum‘*$\{j = 1\}$ˆ$m\pi i$ $P\{ij\}$	i-j

π is the inital income distribution. For any transition probability matrix with a quasi-maximal diagonal, all of these mobility measures take values on $[0, 1]$. 0 means immobility and 1 perfect mobility. If the transition probability matrix takes the form of the identity matrix, every region is stuck in its current state implying complete immobility. On the contrary, when each row of P is identical, current state is irrelevant to the probability of moving away to any class. Thus, the transition matrix with identical rows is considered perfect mobile. The larger the mobility estimate, the more mobile the regional income system is. However, it should be noted that these measures try to reveal mobility pattern from different aspects and are thus not comparable to each other. Actually the mean and variance of these measures are different.

We implemented all the above five summary mobility measures in a single method $markov\_mobility$. A parameter $measure$ could be specified to select which measure to calculate. By default, the mobility measure ‘P’ will be estimated.

```
def markov_mobility(p, measure = "P",ini=None)
```

[1]:
```
from giddy import markov,mobility
mobility.markov_mobility?
```

US income mobility example

Similar to Markov Based Methods notebook, we will demonstrate the usage of the mobility methods by an application to data on per capita incomes observed annually from 1929 to 2009 for the lower 48 US states.

[2]:
```
import libpysal
import numpy as np
import mapclassify as mc
```

[3]:
```
income_path = libpysal.examples.get_path("usjoin.csv")
f = libpysal.io.open(income_path)
pci = np.array([f.by_col[str(y)] for y in range(1929, 2010)]) #each column represents␣
↪an state's income time series 1929-2010
q5 = np.array([mc.Quantiles(y).yb for y in pci]).transpose() #each row represents an␣
↪state's income time series 1929-2010
m = markov.Markov(q5)
m.p
```

[3]:
```
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]])
```

After acquiring the estimate of transition probability matrix, we could call the method $markov\_mobility$ to estimate any of the five Markov-based summary mobility indice.

1. Shorrock1's mobility measure

$$M_P = \frac{m - \sum_{i=1}^{m} P_{ii}}{m - 1}$$

`python measure = "P"`

[4]:
```
mobility.markov_mobility(m.p, measure="P")
```

[4]:
```
0.19758992000997844
```

2. Shorroks2's mobility measure

$$M_D = 1 - |\det(P)|$$

`python measure = "D"`

[5]:
```
mobility.markov_mobility(m.p, measure="D")
```

[5]:
```
0.606848546236956
```

3. Sommers and Conlisk's mobility measure

$$M_{L2} = 1 - |\lambda_2|$$

`python measure = "L2"`

[6]:
```
mobility.markov_mobility(m.p, measure = "L2")
```

[6]:
```
0.03978200230815976
```

4. Bartholomew1's mobility measure

$$M_{B1} = \frac{m - m\sum_{i=1}^{m} \pi_i P_{ii}}{m-1}$$

π: the inital income distribution

`python measure = "B1"`

[7]:
```
pi = np.array([0.1,0.2,0.2,0.4,0.1])
mobility.markov_mobility(m.p, measure = "B1", ini=pi)
```

[7]:
```
0.2277675878319787
```

5. Bartholomew2's mobility measure

$$M_{B2} = \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=1}^{m} \pi_i P_{ij} |i-j|$$

π: the inital income distribution

`python measure = "B1"`

[8]:
```
pi = np.array([0.1,0.2,0.2,0.4,0.1])
mobility.markov_mobility(m.p, measure = "B2", ini=pi)
```

[8]:
```
0.04636660119478926
```

Next steps

- Markov-based partial mobility measures
- Other mobility measures:
  - Inequality reduction mobility measures (Trede, 1999)
- Statistical inference for mobility measures

References

- Formby, J. P., W. J. Smith, and B. Zheng. 2004. "Mobility Measurement, Transition Matrices and Statistical Inference." Journal of Econometrics 120 (1). Elsevier: 181–205.
- Trede, Mark. 1999. "Statistical Inference for Measures of Income Mobility / Statistische Inferenz Zur Messung Der Einkommensmobilität." Jahrbücher Für Nationalökonomie Und Statistik / Journal of Economics and Statistics 218 (3/4). Lucius & Lucius Verlagsgesellschaft mbH: 473–90.

doc/notebooks/MobilityMeasures.ipynb ends here.

The following section was generated from doc/notebooks/directional.ipynb

1.2.4 Directional Analysis of Dynamic LISAs

This notebook demonstrates how to use Rose diagram based inference for directional LISAs.

[1]:
```
import libpysal
import numpy as np
from giddy.directional import Rose
%matplotlib inline
```

[2]:
```
f = open(libpysal.examples.get_path('spi_download.csv'), 'r')
lines = f.readlines()
f.close()
```

[3]:
```
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]])
```

[4]:
```
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/"']
```

[5]:
```
snames = [name for name in names if name not in out]
```

[6]:
```
sids = [names.index(name) for name in snames]
```

[7]:
```
states = data[sids,:]
us = data[0]
years = np.arange(1969, 2009)
```

[8]:
```
rel = states/(us*1.)
```

[9]:
```
gal = libpysal.io.open(libpysal.examples.get_path('states48.gal'))
w = gal.read()
w.transform = 'r'
```

[10]:
```
Y = rel[:, [0, -1]]
```

[11]:
```
Y.shape
```

[11]:
```
(48, 2)
```

[12]:
```
Y
```

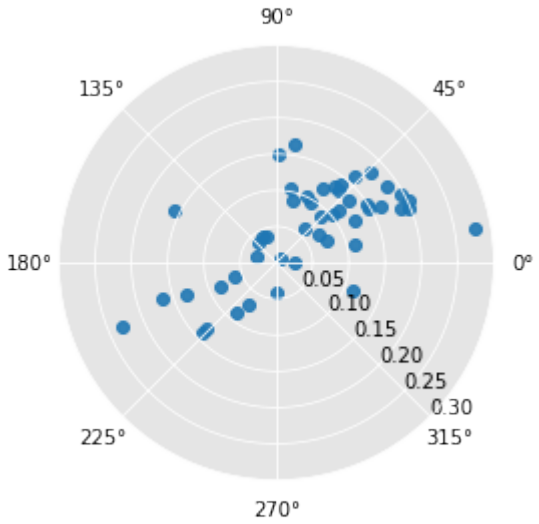
[12]:
```
array([[0.71272158, 0.83983287],
       [0.91110532, 0.85393454],
       [0.68196038, 0.80573518],
       [1.181439  , 1.08538102],
       [0.96115746, 1.06906586],
       [1.25677789, 1.39952248],
       [1.14859228, 1.00773478],
       [0.9535975 , 0.9765967 ],
       [0.82090719, 0.86781238],
       [0.85088634, 0.82257262],
       [1.12956204, 1.05319837],
       [0.9624609 , 0.86064962],
       [0.95542231, 0.93021289],
       [0.92674661, 0.96547951],
       [0.77267987, 0.79775169],
       [0.75234619, 0.90588938],
       [0.81803962, 0.90671011],
       [1.09462982, 1.20319339],
       [1.09098019, 1.27472145],
       [1.08107404, 0.86920513],
       [0.98409802, 1.07035913],
       [0.62643379, 0.75604357],
       [0.93039625, 0.9110376 ],
       [0.85870699, 0.86161958],
       [0.93091762, 0.97368683],
       [1.18091762, 1.02422404],
       [0.97627737, 1.08493335],
       [1.17309698, 1.277308  ],
       [0.76120959, 0.83142658],
       [1.19212722, 1.2125199 ],
       [0.79405631, 0.87902905],
       [0.80787278, 0.99159371],
       [1.01955162, 0.89586649],
       [0.83524505, 0.89497115],
       [0.9580292 , 0.9027308 ],
       [0.99165798, 0.99830879],
       [1.00286757, 1.02884998],
       [0.73540146, 0.81242539],
       [0.7898853 , 0.96152507],
       [0.77085506, 0.86987664],
       [0.87695516, 0.93946478],
       [0.80943691, 0.79446876],
       [0.88112617, 0.96214684],
       [0.92805005, 1.09988062],
       [1.06491137, 1.06588241],
       [0.7278415 , 0.78693295],
       [0.97679875, 0.93929069],
       [0.93508863, 1.20891365]])
```

[13]:
```
np.random.seed(100)
r4 = Rose(Y, w, k=4)
```

Visualization

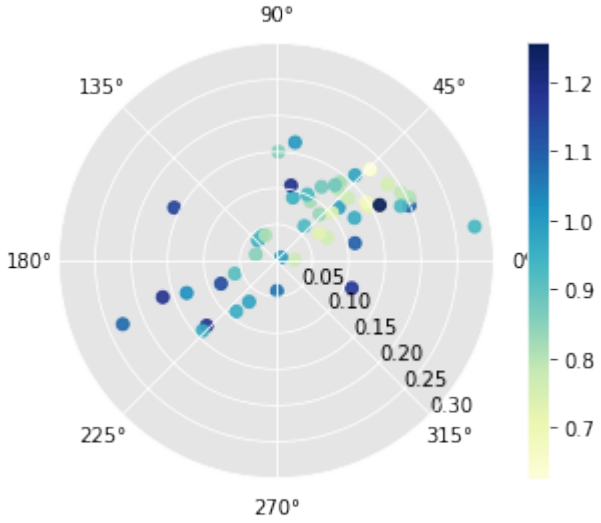
[14]:
```
r4.plot()
```

[14]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.PolarAxesSubplot at 0x1a2a7efb00>)
```



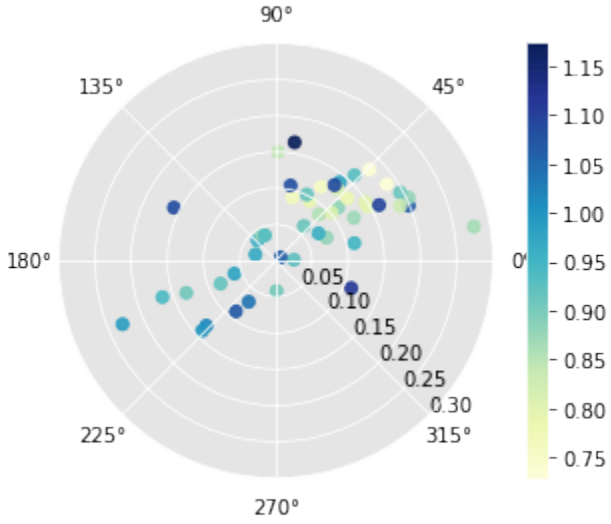
[15]:
```
r4.plot(Y[:,0]) # condition on starting relative income
```

[15]:
```
(<Figure size 432x288 with 2 Axes>,
 <matplotlib.axes._subplots.PolarAxesSubplot at 0x1a2aa8ccc0>)
```



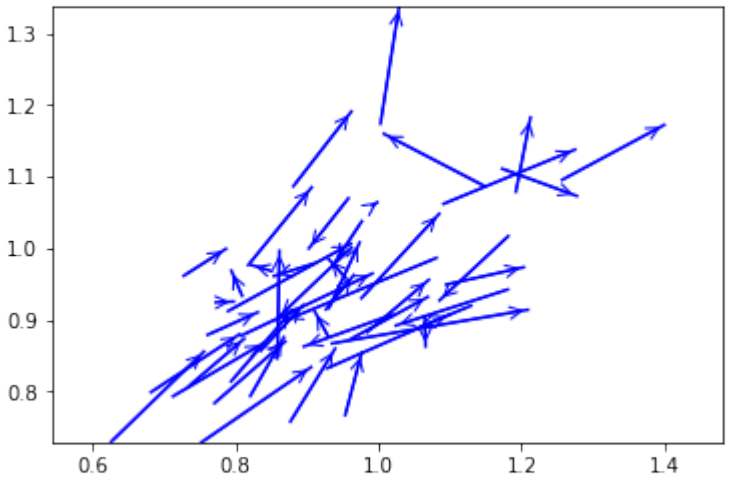
[16]:
```
r4.plot(attribute=r4.lag[:,0]) # condition on the spatial lag of starting relative␣
↪income
```

[16]:
```
(<Figure size 432x288 with 2 Axes>,
 <matplotlib.axes._subplots.PolarAxesSubplot at 0x1a2ac076d8>)
```



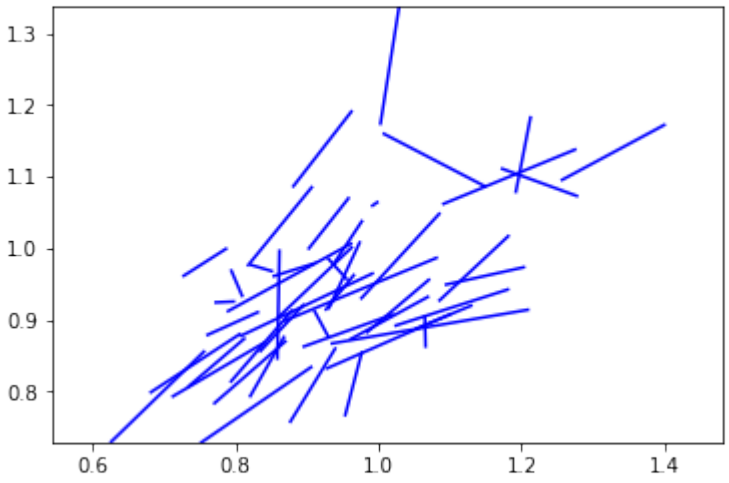
[17]:
```
r4.plot_vectors() # lisa vectors
```

[17]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1a29fafb38>)
```

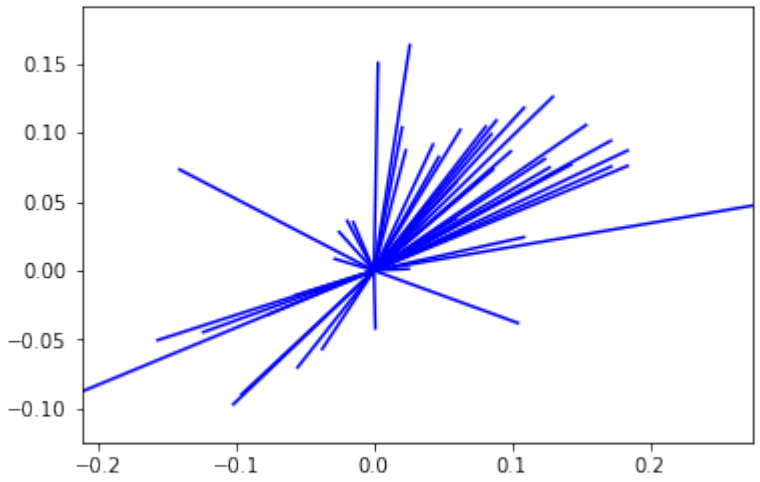


[18]:
```
r4.plot_vectors(arrows=False)
```

[18]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1a2ae61f28>)
```



[19]:
```
r4.plot_origin() # origin standardized
```



Inference

The Rose class contains methods to carry out inference on the circular distribution of the LISA vectors. The first approach is based on a two-sided alternative where the null is that the distribution of the vectors across the segments reflects independence in the movements of the focal unit and its spatial lag. Inference is based on random spatial permutations under the null.

[20]:
```
r4.cuts
```

[20]:
```
array([0.        , 1.57079633, 3.14159265, 4.71238898, 6.28318531])
```

[21]:
```
r4.counts
```

[21]:
```
array([32,  5,  9,  2])
```

[22]:
```
np.random.seed(1234)
```

```
[23]: r4.permute(permutations=999)
```

```
[24]: r4.p
[24]: array([0.028, 0.   , 0.002, 0.004])
```

Here all the four sector counts are signficantly different from their expectation under the null.

A directional test can also be implemented. Here the direction of the departure from the null due to positive co-movement of a focal unit and its spatial lag over the time period results in two two general cases. For sectors in the positive quadrants (I and III), the observed counts are considered extreme if they are larger than expectation, while for the negative quadrants (II, IV) the observed counts are considered extreme if they are small than the expected counts under the null.

```
[25]: r4.permute(alternative='positive', permutations=999)
      r4.p
[25]: array([0.013, 0.001, 0.001, 0.013])
```

```
[26]: r4.expected_perm
[26]: array([27.24824825, 11.56556557,  2.43443443,  6.75175175])
```

Finally, a directional alternative reflecting negative association between the movement of the focal unit and its lag has the complimentary interpretation to the positive alternative: lower counts in I and III, and higher counts in II and IV relative to the null.

```
[27]: r4.permute(alternative='negative', permutations=999)
      r4.p
[27]: array([0.996, 1.   , 1.   , 0.996])
```

..... doc/notebooks/directional.ipynb ends here.

The following section was generated from doc/notebooks/RankbasedMethods.ipynb .....

1.2.5 Rank based Methods

Author: Wei Kang weikang9009@gmail.com, Serge Rey sjsrey@gmail.com

Introduction

This notebook introduces two classic nonparametric statistics of exchange mobility and their spatial extensions. We will demonstrate the usage of these methods by an empirical study for understanding *regional exchange mobility pattern in US*. The dataset is the per capita incomes observed annually from 1929 to 2010 for the lower 48 US states.

- *Kendall's tau*
  - Classic measures:
    - *Classic Kendall's tau*
    - *Local Kendall's tau*
  - Spatial extensions:
    - *Spatial Kendall's tau*
    - *Inter- and Intra-regional decomposition of Kendall's tau*
    - *Local indicator of mobility association-LIMA*

- *Theta statistic of exchange mobility*

Regional exchange mobility pattern in US 1929-2009

Firstly we load in the US dataset:

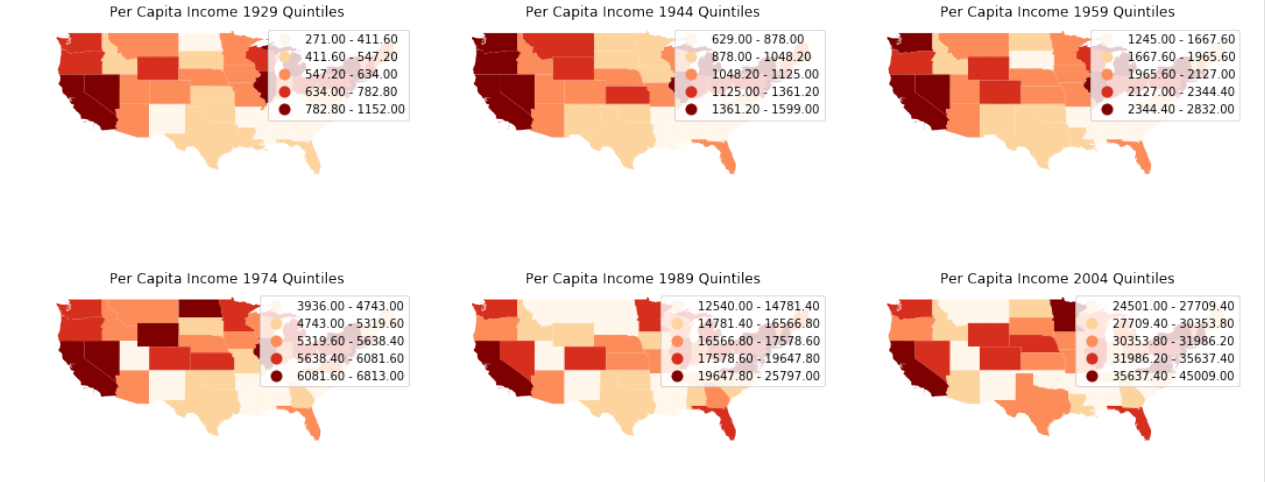
[1]:
```
import pandas as pd
import libpysal
import geopandas as gpd
import numpy as np

geo_table = gpd.read_file(libpysal.examples.get_path('us48.shp'))
income_table = pd.read_csv(libpysal.examples.get_path("usjoin.csv"))
complete_table = geo_table.merge(income_table,left_on='STATE_NAME',right_on='Name')
complete_table.head()
```

[1]:
```
     AREA  PERIMETER  STATE_  STATE_ID    STATE_NAME STATE_FIPS_x SUB_REGION  \
0  20.750     34.956       1         1    Washington           53    Pacific
1  45.132     34.527       2         2       Montana           30        Mtn
2   9.571     18.899       3         3         Maine           23      N Eng
3  21.874     21.353       4         4  North Dakota           38    W N Cen
4  22.598     22.746       5         5  South Dakota           46    W N Cen

  STATE_ABBR                                           geometry          Name  \
0         WA  (POLYGON ((-122.400749206543 48.22539520263672...    Washington
1         MT  POLYGON ((-111.4746322631836 44.70223999023438...       Montana
2         ME  (POLYGON ((-69.77778625488281 44.0740737915039...         Maine
3         ND  POLYGON ((-98.73005676269531 45.93829727172852...  North Dakota
4         SD  POLYGON ((-102.7879333496094 42.99532318115234...  South Dakota

   ...     2000   2001   2002   2003   2004   2005   2006   2007   2008   2009
0  ...    31528  32053  32206  32934  34984  35738  38477  40782  41588  40619
1  ...    22569  24342  24699  25963  27517  28987  30942  32625  33293  32699
2  ...    25623  27068  27731  28727  30201  30721  32340  33620  34906  35268
3  ...    25068  26118  26770  29109  29676  31644  32856  35882  39009  38672
4  ...    26115  27531  27727  30072  31765  32726  33320  35998  38188  36499

[5 rows x 92 columns]
```

We will visualize the spatial distributions of per capita incomes of US states across 1929 to 2009 to obtain a first impression of the dynamics.

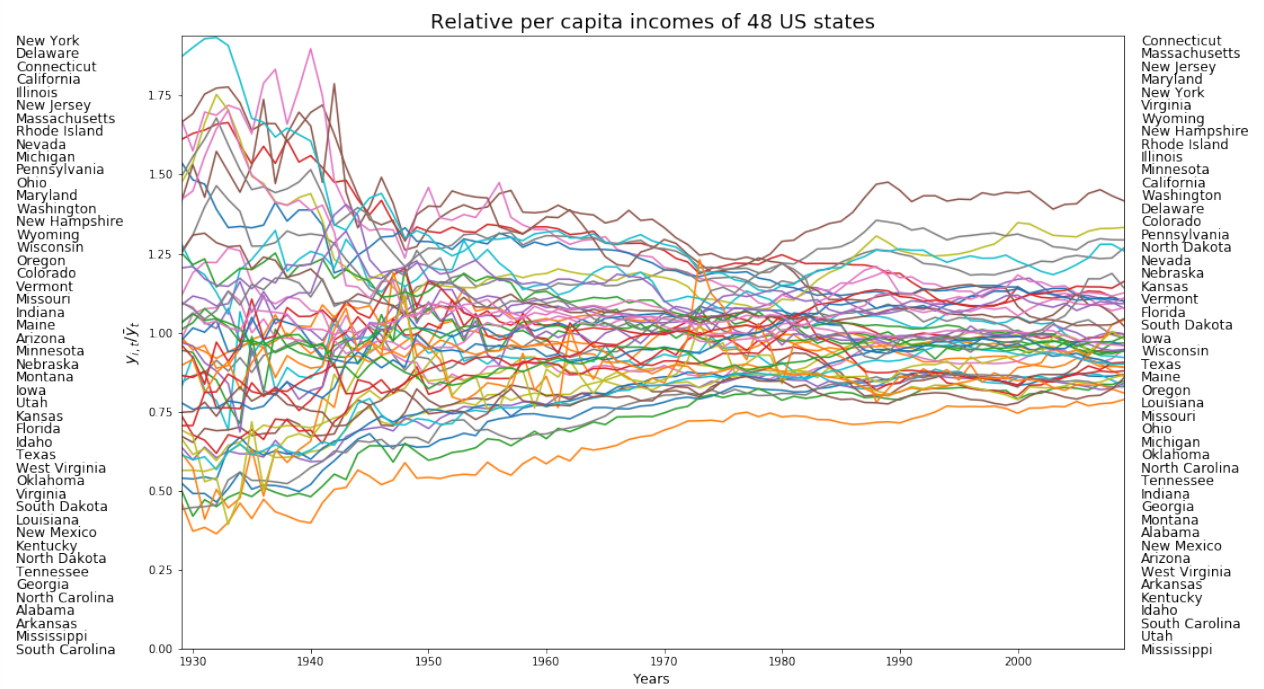
[2]:
```
import matplotlib.pyplot as plt
%matplotlib inline

index_year = range(1929,2010,15)
fig, axes = plt.subplots(nrows=2, ncols=3,figsize = (15,7))
for i in range(2):
    for j in range(3):
        ax = axes[i,j]
        complete_table.plot(ax=ax, column=str(index_year[i*3+j]), cmap='OrRd', scheme=
↪'quantiles', legend=True)
        ax.set_title('Per Capita Income %s Quintiles'%str(index_year[i*3+j]))
        ax.axis('off')
plt.tight_layout()
```

```
/Users/weikang/anaconda3/lib/python3.6/site-packages/pysal/__init__.py:65:␣
↪VisibleDeprecationWarning: PySAL's API will be changed on 2018-12-31. The last␣
↪release made with this API is version 1.14.4. A preview of the next API version is␣
↪provided in the `pysal` 2.0 prelease candidate. The API changes and a guide on how␣
↪to change imports is provided at https://pysal.org/about
  ), VisibleDeprecationWarning)
/Users/weikang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713:␣
↪FutureWarning: Using a non-tuple sequence for multidimensional indexing is␣
↪deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be␣
↪interpreted as an array index, `arr[np.array(seq)]`, which will result either in an␣
↪error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



[3]:
```
years = range(1929,2010)
names = income_table['Name']
pci = income_table.drop(['Name','STATE_FIPS'], 1).values.T
rpci= (pci.T / pci.mean(axis=1)).T
order1929 = np.argsort(rpci[0,:])
order2009 = np.argsort(rpci[-1,:])
names1929 = names[order1929[::-1]]
names2009 = names[order2009[::-1]]
first_last = np.vstack((names1929,names2009))
from pylab import rcParams
rcParams['figure.figsize'] = 15,10
p = plt.plot(years,rpci)
for i in range(48):
    plt.text(1915,1.91-(i*0.041), first_last[0][i],fontsize=12)
    plt.text(2010.5,1.91-(i*0.041), first_last[1][i],fontsize=12)
plt.xlim((years[0], years[-1]))
plt.ylim((0, 1.94))
plt.ylabel(r"$y_{i,t}/\bar{y}_t$",fontsize=14)
plt.xlabel('Years',fontsize=12)
plt.title('Relative per capita incomes of 48 US states',fontsize=18)
```

[3]:
```
Text(0.5,1,'Relative per capita incomes of 48 US states')
```



The above figure displays the trajectories of relative per capita incomes of 48 US states. It is quite obvious that states were swapping positions across 1929-2009. We will demonstrate how to quantify the exchange mobility as well as how to assess the regional and local contribution to the overall exchange mobility. We will ultilize BEA regions and base on it for constructing the block weight matrix.

BEA regional scheme divide US states into 8 regions:

- New England Region
- Mideast Region
- Great Lakes Region
- Plains Region
- Southeast Region
- Southwest Region
- Rocky Mountain Region
- Far West Region

As the dataset does not contain information regarding BEA regions, we manually input the regional information:

[4]:
```
BEA_regions = ["New England Region","Mideast Region","Great Lakes Region","Plains␣
↪Region","Southeast Region","Southwest Region","Rocky Mountain Region","Far West␣
↪Region"]
BEA_regions_abbr = ["NENG","MEST","GLAK","PLNS","SEST","SWST","RKMT","FWST"]
BEA = pd.DataFrame({ 'Region code' : np.arange(1,9,1), 'BEA region' : BEA_regions,
↪'BEA abbr':BEA_regions_abbr})
BEA
```

[4]:

	Region code	BEA region	BEA abbr
0	1	New England Region	NENG
1	2	Mideast Region	MEST
2	3	Great Lakes Region	GLAK

(continues on next page)

(continued from previous page)

```
3             4          Plains Region     PLNS
4             5       Southeast Region     SEST
5             6       Southwest Region     SWST
6             7  Rocky Mountain Region     RKMT
7             8        Far West Region     FWST
```

[5]:
```
region_code = list(np.repeat(1,6))+list(np.repeat(2,6))+list(np.repeat(3,5))+list(np.
↪repeat(4,7))+list(np.repeat(5,12))+list(np.repeat(6,4))+list(np.repeat(7,
↪5))+list(np.repeat(8,6))
state_code = ['09','23','25','33','44','50','10','11','24','34','36','42','17','18',
↪'26','39','55','19','20','27','29','31','38','46','01','05','12','13','21','22','28
↪','37','45','47','51','54','04','35','40','48','08','16','30','49','56','02','06',
↪'15','32','41','53']
state_region = pd.DataFrame({'Region code':region_code,"State code":state_code})
state_region_all = state_region.merge(BEA,left_on='Region code',right_on='Region code
↪')
complete_table = complete_table.merge(state_region_all,left_on='STATE_FIPS_x',right_
↪on='State code')
complete_table.head()
```

[5]:
```
     AREA  PERIMETER  STATE_  STATE_ID    STATE_NAME STATE_FIPS_x SUB_REGION  \
0  20.750     34.956       1         1    Washington           53    Pacific
1  45.132     34.527       2         2       Montana           30        Mtn
2   9.571     18.899       3         3         Maine           23      N Eng
3  21.874     21.353       4         4  North Dakota           38    W N Cen
4  22.598     22.746       5         5  South Dakota           46    W N Cen

  STATE_ABBR                                           geometry          Name  \
0         WA  (POLYGON ((-122.400749206543 48.22539520263672...    Washington
1         MT  POLYGON ((-111.4746322631836 44.70223999023438...       Montana
2         ME  (POLYGON ((-69.77778625488281 44.0740737915039...         Maine
3         ND  POLYGON ((-98.73005676269531 45.93829727172852...  North Dakota
4         SD  POLYGON ((-102.7879333496094 42.99532318115234...  South Dakota

     ...      2004   2005   2006   2007   2008   2009  Region code  \
0    ...     34984  35738  38477  40782  41588  40619            8
1    ...     27517  28987  30942  32625  33293  32699            7
2    ...     30201  30721  32340  33620  34906  35268            1
3    ...     29676  31644  32856  35882  39009  38672            4
4    ...     31765  32726  33320  35998  38188  36499            4

   State code            BEA region  BEA abbr
0          53        Far West Region      FWST
1          30  Rocky Mountain Region      RKMT
2          23     New England Region      NENG
3          38          Plains Region      PLNS
4          46          Plains Region      PLNS

[5 rows x 96 columns]
```

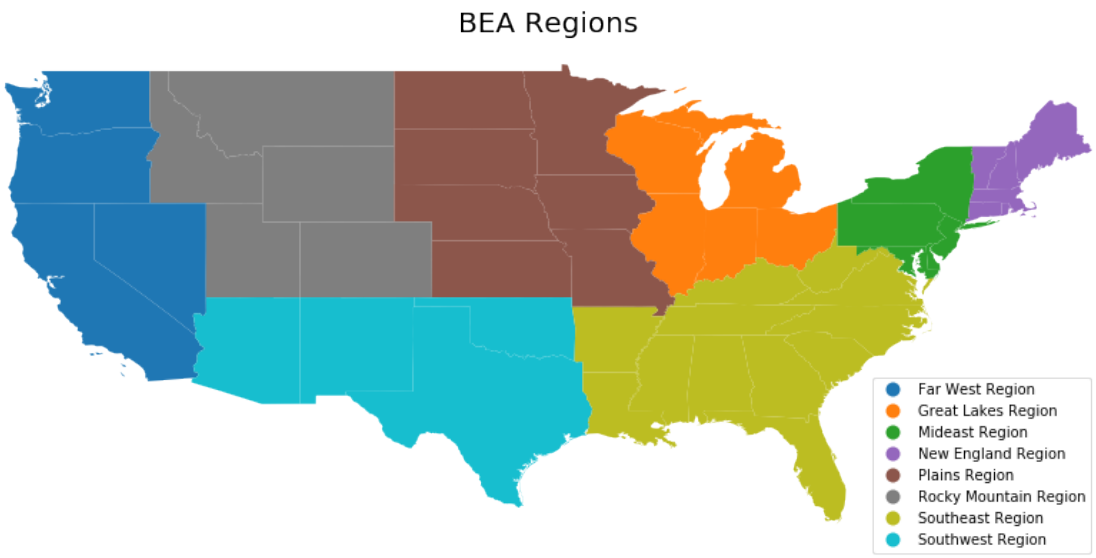
The BEA regions are visualized below:

[6]:
```
fig, ax = plt.subplots(nrows=1, ncols=1,figsize = (15,8))
beaplot = complete_table.plot(ax=ax,column="BEA region", legend=True)
leg = ax.get_legend()
leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
beaplot.set_title("BEA Regions",fontdict={"fontsize":20})
```

(continues on next page)

(continued from previous page)

```
ax.set_axis_off()
```



Kendall's tau

Kendall's τ statistic is based on a comparison of the number of pairs of n observations that have concordant ranks between two variables. For measuring exchange mobility in **giddy**, the two variables in question are the values of an attribute measured at two points in time over n spatial units. This classic measure of rank correlation indicates how much relative stability there has been in the map pattern over the two periods. Spatial decomposition of Kendall's τ could be classified into three spatial scales: global spatial decomposition , inter- and intra-regional decomposition and local spatial decomposition. More details will be given latter.

Classic Kendall's tau

Kendall's τ statistic is a global measure of exchange mobility. For n spatial units over two periods, it is formally defined as follows:

$$\tau = \frac{c - d}{(n(n-1))/2}$$

where c is the number of concordant pairs (two spatial units which do not exchange ranks over two periods), and d is the number of discordant pairs (two spatial units which exchange ranks over two periods). $-1 \leq \tau \leq 1$. Smaller τ indicates higher exchange mobility.

In giddy, class Tau requires two inputs: a cross-section of income values at one period (x) and a cross-section of income values at another period (y):

```
giddy.rank.Tau(self, x, y)
```

We will construct a Tau instance by specifying the incomes in two periods. Here, we look at the global exchange mobility of US states between 1929 and 2009.

[7]:
```
import giddy
```

[8]:
```
tau = giddy.rank.Tau(complete_table["1929"],complete_table["2009"])
tau
```

[8]:
```
<giddy.rank.Tau at 0x1a2130cf60>
```

[9]:
```
tau.concordant
```

[9]:
```
856.0
```

[10]:
```
tau.discordant
```

[10]:
```
271.0
```

There are 856 concordant pairs of US states between 1929 and 2009, and 271 discordant pairs.

[11]:
```
tau.tau
```

[11]:
```
0.5188470576690462
```

[12]:
```
tau.tau_p
```

[12]:
```
1.9735720263920198e-07
```

The observed Kendall's τ statistic is 0.519 and its p-value is 1.974×10^{-7}. Therefore, we will reject the null hypothesis of no assocation between 1929 and 2009 at the 5% significance level.

Spatial Kendall's tau

The spatial Kendall's τ decomposes all pairs into those that are spatial neighbors and those that are not, and examines whether the rank correlation is different between the two sets (Rey, 2014).

$$\tau_w = \frac{\iota'(W \circ S)\iota}{\iota' W \iota}$$

W is the spatial weight matrix, S is the concordance matrix and ι is the $(n, 1)$ unity vector. The null hypothesis is the spatial randomness of rank exchanges. The inference of τ_w could be conducted based on random spatial permutation of incomes at two periods.

```
giddy.rank.SpatialTau(self, x, y, w, permutations=0)
```

For illustration, we turn back to the case of incomes in US states over 1929-2009:

[13]:
```
from libpysal.weights import block_weights
w = block_weights(complete_table["BEA region"])
np.random.seed(12345)
tau_w = giddy.rank.SpatialTau(complete_table["1929"],complete_table["2009"],w,999)
```

```
/Users/weikang/Google Drive (weikang@ucr.edu)/python_repos/pysal-refactor/libpysal/
↪libpysal/weights/weights.py:170: UserWarning: The weights matrix is not fully␣
↪connected. There are 8 components
  warnings.warn("The weights matrix is not fully connected. There are %d components"
↪% self.n_components)
```

[14]:
```
tau_w.concordant
```

[14]:
```
856.0
```

```
[15]: tau_w.concordant_spatial
[15]: 103

[16]: tau_w.discordant
[16]: 271.0

[17]: tau_w.discordant_spatial
[17]: 41
```

Out of 856 concordant pairs of spatial units, 103 belong to the same region (and are considered neighbors); out of 271 discordant pairs of spatial units, 41 belong to the same region.

```
[18]: tau_w.tau_spatial
[18]: 0.4305555555555556

[19]: tau_w.tau_spatial_psim
[19]: 0.001
```

The estimate of spatial Kendall's τ is 0.431 and its p-value is 0.001 which is much smaller than the significance level 0.05. Therefore, we reject the null of spatial randomness of exchange mobility. The fact that $\tau_w = 0.431$ is smaller than the global average $\tau = 0.519$ implies that globally a significant number of rank exchanges happened between states within the same region though we do not know the specific region or regions hosting these rank exchanges. A more thorough decomposition of τ such as inter- and intra-regional indicators and local indicators will provide insights on this issue.

Inter- and Intra-regional decomposition of Kendall's tau

A meso-level view on the exchange mobility pattern is provided by inter- and intra-regional decomposition of Kendall's τ. This decomposition can shed light on specific regions hosting most rank exchanges. More precisely, insteading of examining the concordance relationship between any two neighboring spatial units in the whole study area, for a specific region A, we examine the concordance relationship between any two spatial units within region A (neighbors), resulting in the intraregional concordance statistic for A; or we could examine the concordance relationship between any spatial unit in region A and any spatial unit in region B (nonneighbors), resulting in the interregional concordance statistic for A and B. If there are k regions, there will be k intraregional concordance statistics and $(k-1)^2$ interregional concordance statistics, we could organize them into a (k, k) matrix where the diagonal elements are intraregional concordance statistics and nondiagnoal elements are interregional concordance statistics.

Formally, this inter- and intra-regional concordance statistic matrix is defined as follows (Rey, 2016):

$$T = \frac{P(H \circ S)P'}{PHP'}$$

P is a (k, n) binary matrix where $p_{j,i} = 1$ if spatial unit i is in region j and $p_{j,i} = 0$ otherwise. H is a (n, n) matrix with 0 on diagnoal and 1 on other places. $\circ$ is the Hadamard product. Inference could be based on random spatial permutation of incomes at two periods, similar to spatial τ.

To obtain an estimate for the inter- and intra-regional indicator matrix, we use the $Tau\_Regional$ class:

```
giddy.rank.Tau_Regional(self, x, y, regime, permutations=0)
```

Here, $regime$ is an 1-dimensional array of size n. Each element is the id of which region an spatial unit belongs to.

[20]:
```
giddy.rank.Tau_Regional?
```

Similar to before, we go back to the case of incomes in US states over 1929-2009:

[21]:
```
np.random.seed(12345)
tau_w = giddy.rank.Tau_Regional(complete_table["1929"],complete_table["2009"],
↪complete_table["BEA region"],999)
tau_w
```

```
/Users/weikang/Google Drive (weikang@ucr.edu)/python_repos/pysal-refactor/libpysal/
↪libpysal/weights/weights.py:170: UserWarning: The weights matrix is not fully␣
↪connected. There are 8 components
  warnings.warn("The weights matrix is not fully connected. There are %d components"
↪% self.n_components)
```

[21]:
```
<giddy.rank.Tau_Regional at 0x1a212d85c0>
```

[22]:
```
tau_w.tau_reg
```

[22]:
```
array([[ 0.66666667,  0.5       ,  0.3       ,  0.41666667,  0.28571429,
         0.5       ,  0.79166667,  0.875     ],
       [ 0.5       ,  0.4       ,  0.52      ,  0.26666667, -0.48571429,
         0.52      ,  0.53333333,  0.6       ],
       [ 0.3       ,  0.52      ,  0.        ,  0.4       ,  0.88571429,
         0.76      ,  0.93333333,  1.        ],
       [ 0.41666667,  0.26666667,  0.4       ,  0.86666667,  0.47619048,
         0.83333333,  0.86111111,  0.91666667],
       [ 0.28571429, -0.48571429,  0.88571429,  0.47619048, -0.14285714,
         0.42857143,  0.69047619,  0.14285714],
       [ 0.5       ,  0.52      ,  0.76      ,  0.83333333,  0.42857143,
         0.8       ,  0.06666667,  0.1       ],
       [ 0.79166667,  0.53333333,  0.93333333,  0.86111111,  0.69047619,
         0.06666667,  0.54545455,  0.33333333],
       [ 0.875     ,  0.6       ,  1.        ,  0.91666667,  0.14285714,
         0.1       ,  0.33333333,  0.        ]])
```

The attribute tau_reg gives the inter- and intra-regional concordance statistic matrix. Higher values represents lower exchange mobility. Obviously there are some negative values indicating high exchange mobility. Attribute tau_reg_pvalues gives pvalues for all inter- and intra-regional concordance statistics:

[23]:
```
tau_w.tau_reg_pvalues
```

[23]:
```
array([[0.586, 0.516, 0.196, 0.37 , 0.151, 0.526, 0.051, 0.104],
       [0.516, 0.41 , 0.583, 0.114, 0.001, 0.532, 0.526, 0.472],
       [0.196, 0.583, 0.102, 0.316, 0.011, 0.156, 0.001, 0.014],
       [0.37 , 0.114, 0.316, 0.122, 0.41 , 0.034, 0.003, 0.026],
       [0.151, 0.001, 0.011, 0.41 , 0.013, 0.344, 0.08 , 0.051],
       [0.526, 0.532, 0.156, 0.034, 0.344, 0.324, 0.005, 0.056],
       [0.051, 0.526, 0.001, 0.003, 0.08 , 0.005, 0.502, 0.136],
       [0.104, 0.472, 0.014, 0.026, 0.051, 0.056, 0.136, 0.166]])
```

We can manipulate these two attribute to obtain significant inter- and intra-regional statistics only (at the 5% significance level):

[24]:
```
tau_w.tau_reg * (tau_w.tau_reg_pvalues<0.05)
```

[24]:
```
array([[ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ,  0.        , -0.48571429,
```

(continues on next page)

(continued from previous page)

```
        0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.88571429,
         0.        ,  0.93333333,  1.        ],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.83333333,  0.86111111,  0.91666667],
       [ 0.        , -0.48571429,  0.88571429,  0.        , -0.14285714,
         0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ,  0.83333333,  0.        ,
         0.        ,  0.06666667,  0.        ],
       [ 0.        ,  0.        ,  0.93333333,  0.86111111,  0.        ,
         0.06666667,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  1.        ,  0.91666667,  0.        ,
         0.        ,  0.        ,  0.        ]])
```

The table below displays the inter- and intra-regional decomposition matrix of Kendall's τ for US states over 1929-2009 based on the 8 BEA regions. Bold numbers indicate significance at the 5% significance level. The negative and significant intra-Southeast concordance statistic (-0.486) indicates that the rank exchanges within Southeast region is significantly more frequent than those between states within and out of Southeast region.

Region	New England	Mideast	Great Lakes	Plains	Southeast	Southwest	Rocky Mountain	Far West
New England	0.667	0.5	0.3	0.417	0.2856	0.5	0.792	0.875
Mideast	0.5	0.4	0.52	0.267	**-0.486**	0.52	0.533	0.6
Great Lakes	0.3	0.52	0	0.4	**0.886**	0.76	**0.933**	1.
Plains	0.417	0.267	0.4	0.867	0.476	**0.833**	**0.861**	**0.917**
Southeast	0.286	**-0.486**	**0.886**	0.476	**-0.143**	0.429	0.690	0.143
Southwest	0.5	0.52	0.76	**0.833**	0.429	0.8	**0.067**	0.1
Rocky Mountain	0.792	0.533	**0.933**	**0.861**	0.69	**0.067**	0.545	0.333
Far West	0.875	0.6	1.	**0.917**	0.143	0.1	0.333	0

Local Kendall's tau

Local Kendall's τ is a local decomposition of classic Kendall's τ which provides an indication of the contribution of spatial unit r's rank changes to the overall level of exchange mobility (Rey, 2016). Focusing on spatial unit r, we formally define it as follows:

$$\tau_r = \frac{c_r - d_r}{n - 1}$$

where c_r is the number of spatial units (except r) which are concordant with r and d_r is the number of spatial units which are discordant with r. Similar to classic Kendall's τ, local τ takes values on $[-1, 1]$. The larger the value, the lower the exchange mobility for r.

```
giddy.rank.Tau_Local(self, x, y)
```

We create a Tau_Local instance for US dynamics 1929-2009:

[25]:
```
tau_r = giddy.rank.Tau_Local(complete_table["1929"],complete_table["2009"])
tau_r
```

[25]:
```
<giddy.rank.Tau_Local at 0x1141187b8>
```

[26]:
```
pd.DataFrame({"STATE_NAME":complete_table['STATE_NAME'].tolist(),"$\\tau_r$":tau_r.
↪tau_local}).head()
```

[26]:

	STATE_NAME	τ_r
0	Washington	0.617021
1	Montana	0.446809
2	Maine	0.404255
3	North Dakota	-0.021277
4	South Dakota	0.319149

Therefore, local concordance measure produces a negative value for North Dakota (-0.0213) indicating that North Dakota exchanged ranks with a lot of states across 1929-2000. On the contrary, the local τ statistic is quite high for Washington (0.617) highlighting a high stability of Washington.

Local indicator of mobility association-LIMA

To reveal the role of space in shaping the exchange mobility pattern for each spatial unit, two spatial variants of local Kendall's τ could be utilized: neighbor set LIMA and neighborhood set LIMA (Rey, 2016). The latter is also the result of a decomposition of local Kendall's τ (into neighboring and nonneighboring parts) as well as a decompostion of spatial Kendall's τ (into its local components).

Neighbor set LIMA

Instead of examining the concordance relationship between a focal spatial unit r and all the other units as what local τ does, neighbor set LIMA focuses on the concordance relationship between a focal spatial unit r and its neighbors only. It is formally defined as follows:

$$\tilde{\tau}_r = \frac{\sum_b w_{r,b} s_{r,b}}{\sum_b w_{r,b}}$$

```
giddy.rank.Tau_Local_Neighbor(self, x, y, w, permutations=0)
```

[27]:
```
tau_wr = giddy.rank.Tau_Local_Neighbor(complete_table["1929"],complete_table["2009"],
↪w,999)
tau_wr
```

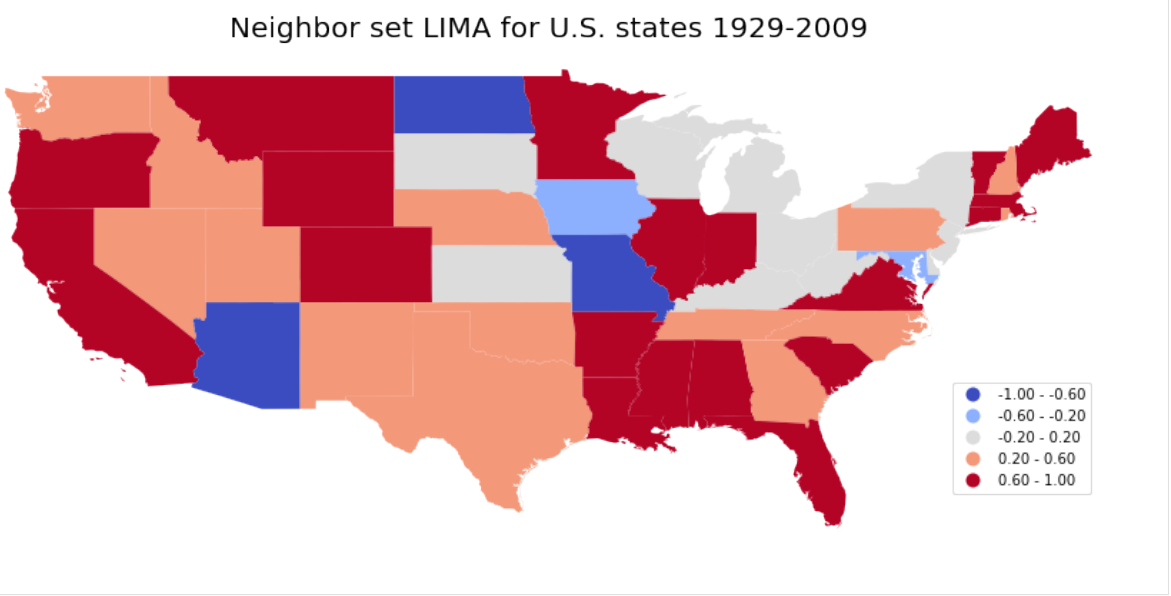
[27]:
```
<giddy.rank.Tau_Local_Neighbor at 0x114118f98>
```

[28]:
```
tau_wr.tau_ln
```

[28]:
```
array([ 0.33333333,  1.        ,  1.        , -0.66666667,  0.        ,
        1.        ,  0.        ,  0.5       ,  1.        ,  0.66666667,
        1.        ,  0.6       , -0.33333333,  1.        ,  0.33333333,
        0.        ,  0.5       ,  1.        ,  0.6       ,  0.        ,
        1.        ,  0.33333333,  0.5       ,  1.        ,  0.        ,
        1.        ,  0.        , -0.09090909, -0.5       ,  1.        ,
        0.09090909,  0.        ,  0.63636364, -1.        , -1.        ,
        0.33333333,  0.27272727,  0.45454545,  0.33333333,  0.33333333,
        0.63636364,  0.81818182,  0.45454545,  0.81818182,  0.81818182,
        0.81818182,  0.81818182,  0.        ])
```

To visualize the spatial distribution of neighbor set LIMA:

[29]:
```
complete_table["tau_ln"] =tau_wr.tau_ln
fig, ax = plt.subplots(nrows=1, ncols=1,figsize = (15,8))
ln_map = complete_table.plot(ax=ax, column="tau_ln", cmap='coolwarm', scheme='equal_
↪interval',legend=True)
leg = ax.get_legend()
leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
ln_map.set_title("Neighbor set LIMA for U.S. states 1929-2009",fontdict={"fontsize":
↪20})
ax.set_axis_off()
```



Therefore, Arizona, North Dakota, and Missouri exchanged ranks with most of their neighbors over 1929-2009 while California, Virginia etc. barely exchanged ranks with their neighbors.

Let see whether neighbor set LIMA statistics are siginificant for these "extreme" states:

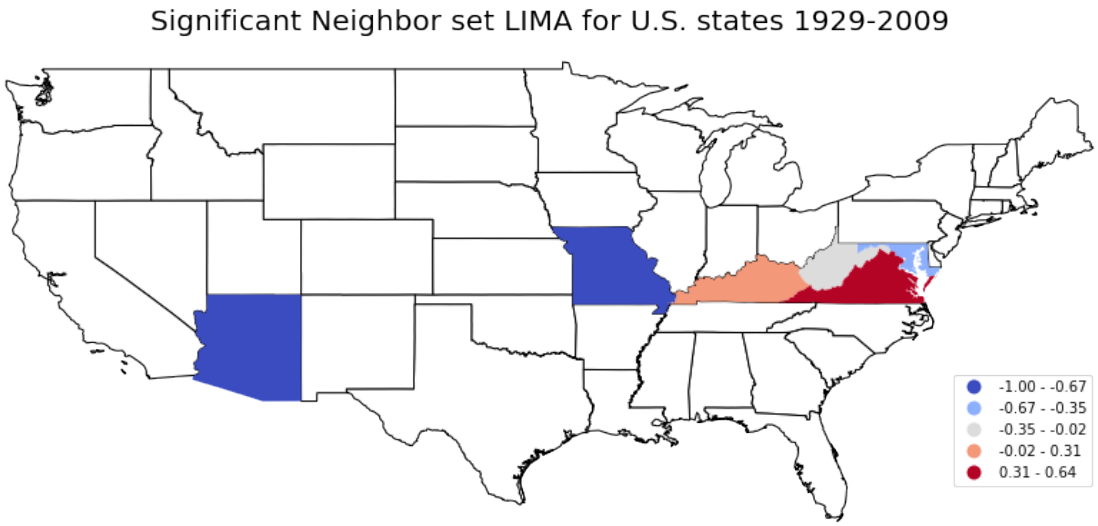
[30]:
```
tau_wr.tau_ln_pvalues
```

[30]:
```
array([0.463, 0.256, 0.165, 0.101, 0.316, 0.336, 0.237, 0.614, 0.292,
       0.325, 0.33 , 0.675, 0.06 , 0.541, 0.412, 0.032, 0.594, 0.791,
       0.575, 0.049, 0.209, 0.48 , 0.488, 0.457, 0.605, 0.409, 0.259,
       0.018, 0.022, 0.405, 0.016, 0.25 , 0.001, 0.001, 0.045, 0.521,
       0.167, 0.363, 0.635, 0.478, 0.417, 0.247, 0.282, 0.423, 0.578,
       0.17 , 0.1  , 0.625])
```

[31]:
```
sig_wr = tau_wr.tau_ln * (tau_wr.tau_ln_pvalues<0.05)
sig_wr
```

[31]:
```
array([ 0.        ,  0.        ,  0.        , -0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        , -0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        , -0.09090909, -0.5       ,  0.        ,
        0.09090909,  0.        ,  0.63636364, -1.        , -1.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ])
```

[32]:
```
complete_table["sig_wr"] =sig_wr
fig, ax = plt.subplots(nrows=1, ncols=1,figsize = (15,8))
complete_table[complete_table["sig_wr"] == 0].plot(ax=ax, color='white',edgecolor=
↪'black')
sig_ln_map = complete_table[complete_table["sig_wr"] != 0].plot(ax=ax,column="sig_wr",
↪cmap='coolwarm',scheme='equal_interval',legend=True)
leg = ax.get_legend()
leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
sig_ln_map.set_title("Significant Neighbor set LIMA for U.S. states 1929-2009",
↪fontdict={"fontsize":20})
ax.set_axis_off()
```



Thus, Arizona and Missouri have significant and negative neighbor set LIMA values, and can be considered as hotspots of rank exchanges. This means that Arizona (or Missouri) tended to exchange ranks with its neighbors than with others over 1929-2009. On the contrary, Virgina has significant and large positive neighbor set LIMA value indicating that it tended to exchange ranks with its nonneighbors than with neighbors.

Neighborhood set LIMA

Neighborhood set LIMA extends neighbor set LIMA $\tilde{\tau}_r$ to consider the concordance relationships between any two spatial units in the subset which is composed of the focal unit r and its neighbors.

```
giddy.rank.Tau_Local_Neighborhood(self, x, y, w, permutations=0)
```

[33]:
```
tau_wwr = giddy.rank.Tau_Local_Neighborhood(complete_table["1929"],complete_table[
↪"2009"],w,999)
tau_wwr
```

[33]:
```
<giddy.rank.Tau_Local_Neighborhood at 0x114118f28>
```

[34]:
```
tau_wwr.tau_lnhood
```

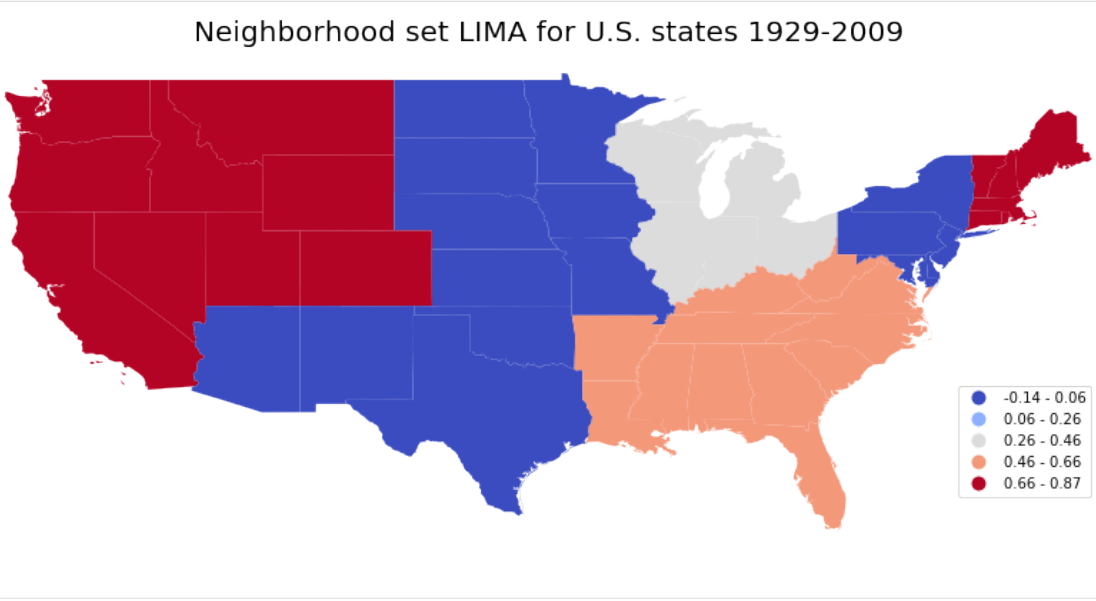
[34]:
```
array([ 0.66666667,  0.8       ,  0.86666667, -0.14285714, -0.14285714,
        0.8       ,  0.4       ,  0.8       ,  0.86666667, -0.14285714,
        0.66666667,  0.86666667, -0.14285714,  0.86666667, -0.14285714,
```

(continues on next page)

(continued from previous page)

```
       0.        ,  0.        ,  0.86666667,  0.86666667,  0.        ,
       0.4       ,  0.66666667,  0.8       ,  0.66666667,  0.4       ,
       0.4       ,  0.        ,  0.54545455,  0.        ,  0.8       ,
       0.54545455, -0.14285714,  0.54545455, -0.14285714,  0.        ,
       0.        ,  0.54545455,  0.54545455,  0.        ,  0.        ,
       0.54545455,  0.54545455,  0.54545455,  0.54545455,  0.54545455,
       0.54545455,  0.54545455,  0.4       ])
```

[35]:
```
complete_table["tau_lnhood"] =tau_wwr.tau_lnhood
fig, ax = plt.subplots(nrows=1, ncols=1,figsize = (15,8))
ln_map = complete_table.plot(ax=ax, column="tau_lnhood", cmap='coolwarm', scheme=
↪'equal_interval',legend=True)
leg = ax.get_legend()
leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
ln_map.set_title("Neighborhood set LIMA for U.S. states 1929-2009",fontdict={"fontsize
↪":20})
ax.set_axis_off()
```



[36]:
```
tau_wwr.tau_lnhood_pvalues
```

[36]:
```
array([0.585, 0.278, 0.104, 0.032, 0.024, 0.295, 0.43 , 0.225, 0.167,
       0.02 , 0.548, 0.116, 0.023, 0.158, 0.017, 0.016, 0.075, 0.225,
       0.168, 0.027, 0.505, 0.66 , 0.146, 0.605, 0.614, 0.37 , 0.08 ,
       0.505, 0.059, 0.358, 0.541, 0.025, 0.185, 0.017, 0.225, 0.151,
       0.541, 0.527, 0.191, 0.12 , 0.519, 0.427, 0.526, 0.442, 0.453,
       0.528, 0.478, 0.617])
```

[37]:
```
sig_lnhood = tau_wwr.tau_lnhood * (tau_wwr.tau_lnhood_pvalues<0.05)
sig_lnhood
```

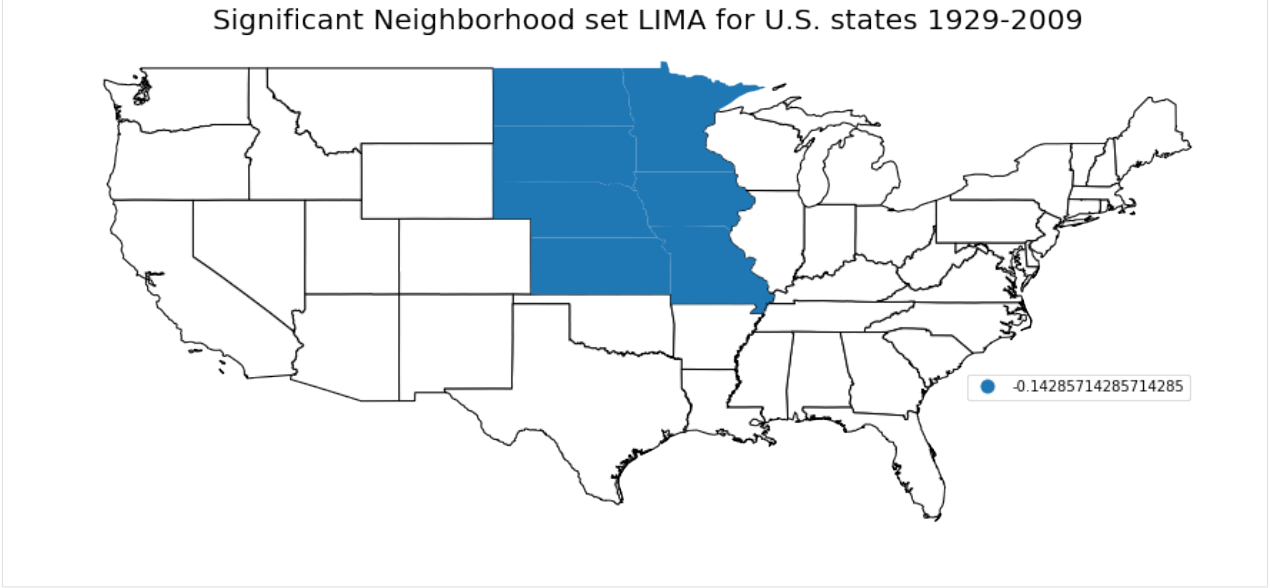
[37]:
```
array([ 0.        ,  0.        ,  0.        , -0.14285714, -0.14285714,
        0.        ,  0.        ,  0.        ,  0.        , -0.14285714,
        0.        ,  0.        , -0.14285714,  0.        , -0.14285714,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
```

(continues on next page)

(continued from previous page)

```
       0.        , -0.14285714,  0.        , -0.14285714,  0.        ,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        ,  0.        ])
```

[38]:
```
complete_table["sig_lnhood"] =sig_lnhood
fig, ax = plt.subplots(nrows=1, ncols=1,figsize = (15,8))
complete_table[complete_table["sig_lnhood"] == 0].plot(ax=ax, color='white',edgecolor=
↪'black')
sig_ln_map = complete_table[complete_table["sig_lnhood"] != 0].plot(ax=ax,column="sig_
↪lnhood",categorical=True,legend=True)
leg = ax.get_legend()
leg.set_bbox_to_anchor((0.8, 0.15, 0.16, 0.2))
sig_ln_map.set_title("Significant Neighborhood set LIMA for U.S. states 1929-2009",
↪fontdict={"fontsize":20})
ax.set_axis_off()
```



Theta statistic of exchange mobility

Θ statistic of exchange mobility

Next steps

- theta statistic

References

- Rey, Sergio J., and Myrna L. Sastré-Gutiérrez. 2010. "Interregional Inequality Dynamics in Mexico." Spatial Economic Analysis 5 (3). Taylor & Francis: 277–98.
- Rey, Sergio J. 2014. "Fast Algorithms for a Space-Time Concordance Measure." Computational Statistics 29 (3-4). Springer: 799–811.

- Rey, Sergio J. 2016. "Space–Time Patterns of Rank Concordance: Local Indicators of Mobility Association with Application to Spatial Income Inequality Dynamics." Annals of the Association of American Geographers. Association of American Geographers 106 (4): 788–803.

[ ]:

..... doc/notebooks/RankbasedMethods.ipynb ends here.

1.3 API reference

1.3.1 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.

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']])
```

(continues on next page)

(continued from previous page)

```
>>> 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.

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.        ]
 [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 ]
 [ 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]
```

(continues on next page)

(continued from previous page)

```
 [ 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'
```

(continues on next page)

(continued from previous page)

```
>>> "%.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
```

(continues on next page)

(continued from previous page)

```
>>> 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.

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.

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.

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],
       [ 76.96801786,  64.7713783 ,  48.        , ...,  73.84595169,
```

(continues on next page)

(continued from previous page)

```
         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.

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'
```

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])
```

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.

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.])
```

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).

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).

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]])
```

1.3.2 Directional LISA

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

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 numpy as np
>>> 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.

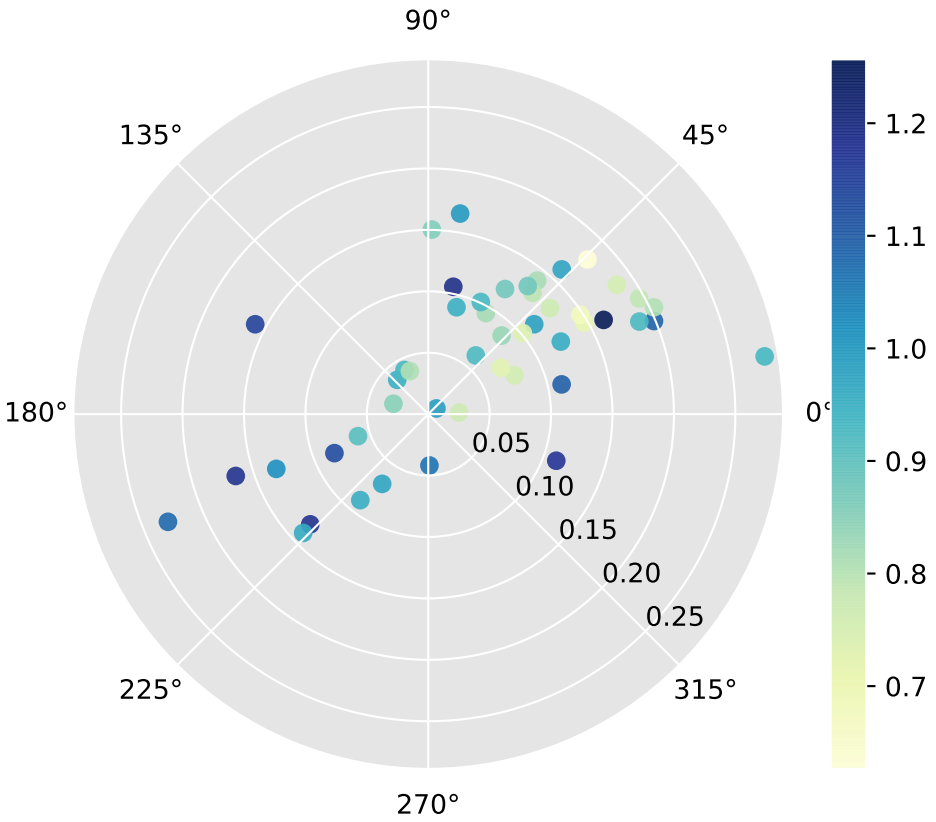
```
>>> r8.permute(alternative='positive')
>>> r8.p
array([0.51, 0.04, 0.28, 0.02, 0.01, 0.14, 0.57, 0.03])
```

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()
```



1.3.3 Economic Mobility Indices

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

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
```

1.3.4 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.

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.

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.

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.

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.

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.

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.

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.

1.3.5 Alignment-based Sequence Methods

giddy.sequence.Sequence(y[, subs_mat, ...])	Pairwise sequence analysis.

giddy.sequence.Sequence

class giddy.sequence.**Sequence**(*y*, *subs_mat=None*, *dist_type=None*, *indel=None*, *cluster_type=None*)

Pairwise sequence analysis.

Dynamic programming if optimal matching.

Parameters

y [array] one row per sequence of neighborhood types for each spatial unit. Sequences could be of varying lengths.

subs_mat [array] (k,k), substitution cost matrix. Should be hollow (0 cost between the same type), symmetric and non-negative.

dist_type [string] "hamming": hamming distance (substitution only and its cost is constant 1) from sklearn.metrics; "markov": utilize empirical transition probabilities to define substitution costs; "interval": differences between states are used to define substitution costs, and indel=k-1; "arbitrary": arbitrary distance if there is not a strong theory guidance: substitution=0.5, indel=1. "tran": transition-oriented optimal matching. Sequence of transitions. Based on [Bie11].

indel [float] insertion/deletion cost.

cluster_type [string] cluster algorithm (specification) used to generate neighborhood types, such as "ward", "kmeans", etc.

Examples

```
>>> import numpy as np
```

1. Testing on unequal string sequences 1.1 substitution cost matrix and indel cost are not given, and will be generated based on the distance type "interval"

```
>>> seq1 = 'ACGGTAG'
>>> seq2 = 'CCTAAG'
>>> seq3 = 'CCTAAGC'
```

(continues on next page)

(continued from previous page)

```
>>> seqAna = Sequence([seq1,seq2,seq3],dist_type="interval")
>>> seqAna.k
4
>>> seqAna.classes
array(['A', 'C', 'G', 'T'], dtype='<U1')
>>> seqAna.subs_mat
array([[0., 1., 2., 3.],
       [1., 0., 1., 2.],
       [2., 1., 0., 1.],
       [3., 2., 1., 0.]])
>>> seqAna.seq_dis_mat
array([[ 0.,  7., 10.],
       [ 7.,  0.,  3.],
       [10.,  3.,  0.]])
```

1.2 User-defined substitution cost matrix and indel cost

```
>>> subs_mat = np.array([[0, 0.76, 0.29, 0.05],[0.30, 0, 0.40, 0.60],[0.16, 0.61,␣
↪0, 0.26],[0.38, 0.20, 0.12, 0]])
>>> indel = subs_mat.max()
>>> seqAna = Sequence([seq1,seq2,seq3], subs_mat=subs_mat, indel=indel)
>>> seqAna.seq_dis_mat
array([[0.  , 1.94, 2.46],
       [1.94, 0.  , 0.76],
       [2.46, 0.76, 0.  ]])
```

1.3 Calculating "hamming" distance will fail on unequal sequences

```
>>> seqAna = Sequence([seq1,seq2,seq3], dist_type="hamming")
Traceback (most recent call last):
ValueError: hamming distance cannot be calculated for sequences of unequal␣
↪lengths!
```

2. Testing on equal string sequences

```
>>> seq1 = 'ACGGTAG'
>>> seq2 = 'CCTAAGA'
>>> seq3 = 'CCTAAGC'
```

2.1 Calculating "hamming" distance

```
>>> seqAna = Sequence([seq1,seq2,seq3], dist_type="hamming")
>>> seqAna.seq_dis_mat
array([[0., 6., 6.],
       [6., 0., 1.],
       [6., 1., 0.]])
```

2.2 User-defined substitution cost matrix and indel cost (distance between different types is always 1 and indel cost is 2) - give the same sequence distance matrix as "hamming" distance

```
>>> subs_mat = np.array([[0., 1., 1., 1.],[1., 0., 1., 1.],[1., 1., 0., 1.],[1.,␣
↪1., 1., 0.]])
>>> indel = 2
>>> seqAna = Sequence([seq1,seq2,seq3], subs_mat=subs_mat, indel=indel)
>>> seqAna.seq_dis_mat
```

(continues on next page)

(continued from previous page)

```
array([[0., 6., 6.],
       [6., 0., 1.],
       [6., 1., 0.]])
```

2.3 User-defined substitution cost matrix and indel cost (distance between different types is always 1 and indel cost is 1) - give a slightly different sequence distance matrix from "hamming" distance since insertion and deletion is happening

```
>>> subs_mat = np.array([[0., 1., 1., 1.],[1., 0., 1., 1.],[1., 1., 0.,1.],[1., 1.
↪, 1., 0.]])
>>> indel = 1
>>> seqAna = Sequence([seq1,seq2,seq3], subs_mat=subs_mat, indel=indel)
>>> seqAna.seq_dis_mat
array([[0., 5., 5.],
       [5., 0., 1.],
       [5., 1., 0.]])
```

3. Not passing proper parameters will raise an error

```
>>> seqAna = Sequence([seq1,seq2,seq3])
Traceback (most recent call last):
ValueError: Please specify a proper `dist_type` or `subs_mat` and `indel` to␣
↪proceed!
```

```
>>> seqAna = Sequence([seq1,seq2,seq3], subs_mat=subs_mat)
Traceback (most recent call last):
ValueError: Please specify a proper `dist_type` or `subs_mat` and `indel` to␣
↪proceed!
```

```
>>> seqAna = Sequence([seq1,seq2,seq3], indel=indel)
Traceback (most recent call last):
ValueError: Please specify a proper `dist_type` or `subs_mat` and `indel` to␣
↪proceed!
```

Attributes

seq_dis_mat [array] (n,n), distance/dissimilarity matrix for each pair of sequences

classes [array] (k,), unique classes

k [int] number of unique classes

label_dict [dict] dictionary - {input label: int value between 0 and k-1 (k is the number of unique classes for the pooled data)}

__init__ (*self*, *y*, *subs_mat=None*, *dist_type=None*, *indel=None*, *cluster_type=None*)
Initialize self. See help(type(self)) for accurate signature.

1.4 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.

[Bie11] Torsten Biemann. A transition-oriented approach to optimal matching. *Sociological Methodology*, 41(1):195–221, 2011. URL: https://doi.org/10.1111/j.1467-9531.2011.01235.x, doi:10.1111/j.1467-9531.2011.01235.x.

[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 Associa-*

tion 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

__init__() (*giddy.directional.Rose method*), 74
__init__() (*giddy.markov.FullRank_Markov method*), 66
__init__() (*giddy.markov.GeoRank_Markov method*), 67
__init__() (*giddy.markov.LISA_Markov method*), 64
__init__() (*giddy.markov.Markov method*), 54
__init__() (*giddy.markov.Spatial_Markov method*), 60
__init__() (*giddy.rank.SpatialTau method*), 81
__init__() (*giddy.rank.Tau method*), 80
__init__() (*giddy.rank.Tau_Local method*), 82
__init__() (*giddy.rank.Tau_Local_Neighbor method*), 84
__init__() (*giddy.rank.Tau_Local_Neighborhood method*), 85
__init__() (*giddy.rank.Tau_Regional method*), 87
__init__() (*giddy.rank.Theta method*), 79
__init__() (*giddy.sequence.Sequence method*), 89

F

fmpt() (*in module giddy.ergodic*), 70
FullRank_Markov (*class in giddy.markov*), 65

G

GeoRank_Markov (*class in giddy.markov*), 66

H

homogeneity() (*in module giddy.markov*), 69

K

kullback() (*in module giddy.markov*), 67

L

LISA_Markov (*class in giddy.markov*), 60

M

Markov (*class in giddy.markov*), 52
markov_mobility() (*in module giddy.mobility*), 77

P

permute() (*giddy.directional.Rose method*), 73
plot() (*giddy.directional.Rose method*), 73
plot_origin() (*giddy.directional.Rose method*), 73
plot_vectors() (*giddy.directional.Rose method*), 73
prais() (*in module giddy.markov*), 68

R

Rose (*class in giddy.directional*), 72

S

Sequence (*class in giddy.sequence*), 87
sojourn_time() (*in module giddy.markov*), 69
Spatial_Markov (*class in giddy.markov*), 54
SpatialTau (*class in giddy.rank*), 80
spillover() (*giddy.markov.LISA_Markov method*), 64
steady_state() (*in module giddy.ergodic*), 70
summary() (*giddy.markov.Spatial_Markov method*), 60

T

Tau (*class in giddy.rank*), 79
Tau_Local (*class in giddy.rank*), 82
Tau_Local_Neighbor (*class in giddy.rank*), 83
Tau_Local_Neighborhood (*class in giddy.rank*), 84
Tau_Regional (*class in giddy.rank*), 85
Theta (*class in giddy.rank*), 78

V

var_fmpt() (*in module giddy.ergodic*), 71