esda Documentation

Release 2.2.0

pysal developers

Dec 18, 2019

CONTENTS:

ESDA is an open-source Python library for the exploratory analysis of spatial data. A subpackage of PySAL (Python Spatial Analysis Library), it is under active development and includes methods for global and local spatial autocorrelation analysis.

CHAPTER

ONE

INSTALLATION

esda supports Python 3.6 and 3.7 only. Please make sure that you are operating in a Python 3 environment.

1.1 Installing released version

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

```
pip install -U esda
```

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 .
```

1.2 Installing development version

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

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

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

CHAPTER

TWO

API REFERENCE

2.1 Gamma Statistic

esda.Gamma(y, w[, operation, standardize, ...])	Gamma index for spatial autocorrelation

2.1.1 esda.Gamma

class esda.**Gamma**(*y*, *w*, *operation='c'*, *standardize=False*, *permutations=999*)

Gamma index for spatial autocorrelation

Parameters

y [array] variable measured across n spatial units

w [W] spatial weights instance can be binary or row-standardized

operation [{'c', 's', 'a'}] attribute similarity function where, 'c' cross product 's' squared difference 'a' absolute difference

standardize [{False, True}] standardize variables first False, keep as is True, standardize to mean zero and variance one

permutations [int] number of random permutations for calculation of pseudo-p_values

Notes

For further technical details see [HGC81].

Examples

use same example as for join counts to show similarity

```
>>> import libpysal, numpy as np
>>> from esda.gamma import Gamma
>>> w = libpysal.weights.lat2W(4,4)
>>> y=np.ones(16)
>>> y[0:8]=0
>>> np.random.seed(12345)
>>> g = Gamma(y,w)
>>> g.g
20.0
```

(continues on next page)

(continued from previous page)

```
>>> round(g.g_z, 3)
3.188
>>> round(g.p_sim_g, 3)
0.003
>>> g.min_g
0.0
>>> g.max_g
20.0
>>> g.mean_g
11.093093093093094
>>> np.random.seed(12345)
>>> g1 = Gamma(y,w,operation='s')
>>> g1.g
8.0
>>> round(g1.g_z, 3)
-3.706
>>> g1.p_sim_g
0.001
>>> g1.min_g
14.0
>>> g1.max_g
48.0
>>> g1.mean_g
25.623623623623622
>>> np.random.seed(12345)
>>> g2 = Gamma(y,w,operation='a')
>>> g2.g
8.0
>>> round(g2.g_z, 3)
-3.706
>>> g2.p_sim_g
0.001
>>> g2.min_g
14.0
>>> g2.max_g
48.0
>>> g2.mean_g
25.623623623623622
>>> np.random.seed(12345)
>>> g3 = Gamma(y,w,standardize=True)
>>> g3.g
32.0
>>> round(g3.g_z, 3)
3.706
>>> g3.p_sim_g
0.001
>>> g3.min_g
-48.0
>>> g3.max_g
20.0
>>> g3.mean_g
-3.2472472472472473
>>> np.random.seed(12345)
>>> def func(z,i,j):
...     q = z[i]*z[j]
...     return q
...
```

(continues on next page)

(continued from previous page)

```
>>> g4 = Gamma(y,w,operation=func)
>>> g4.g
20.0
>>> round(g4.g_z, 3)
3.188
>>> round(g4.p_sim_g, 3)
0.003
```

Attributes

y [array] original variable

w [W] original w object

op [{'c', 's', 'a'}] attribute similarity function, as per parameters attribute similarity function

stand [{False, True}] standardization

permutations [int] number of permutations

gamma [float] value of Gamma index

sim_g [array] (if permutations>0) vector of Gamma index values for permuted samples

p_sim_g [array] (if permutations>0) p-value based on permutations (one-sided) null: spatial randomness alternative: the observed Gamma is more extreme than under randomness implemented as a two-sided test

mean_g [float] average of permuted Gamma values

min_g [float] minimum of permuted Gamma values

max_g [float] maximum of permuted Gamma values

Methods

by_col	

__init__(*self*, *y*, *w*, *operation='c'*, *standardize=False*, *permutations=999*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, w[, operation, ...])	Initialize self.
by_col(df, cols[, w, inplace, pvalue, outvals])	

Attributes

p_sim	new name to fit with Moran module

2.2 Geary Statistic

esda.Geary(y, w[, transformation, permutations])	Global Geary C Autocorrelation statistic

2.2.1 esda.Geary

class esda.**Geary**(*y*, *w*, *transformation='r'*, *permutations=999*)

Global Geary C Autocorrelation statistic

Parameters

y [array] (n, 1) attribute vector

w [W] spatial weights

transformation [{'R', 'B', 'D', 'U', 'V'}] weights transformation, default is row-standardized. Other options include "B": binary, "D": doubly-standardized, "U": untransformed (general weights), "V": variance-stabilizing.

permutations [int] number of random permutations for calculation of pseudo-p_values

Notes

Technical details and derivations can be found in [CO81].

Examples

```
>>> import libpysal
>>> from esda.geary import Geary
>>> w = libpysal.io.open(libpysal.examples.get_path("book.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("book.txt"))
>>> y = np.array(f.by_col['y'])
>>> c = Geary(y,w,permutations=0)
>>> round(c.C,7)
0.3330108
>>> round(c.p_norm,7)
9.2e-05
>>>
```

Attributes

y [array] original variable

w [W] spatial weights

permutations [int] number of permutations

C [float] value of statistic

EC [float] expected value

VC [float] variance of G under normality assumption

z_norm [float] z-statistic for C under normality assumption

z_rand [float] z-statistic for C under randomization assumption

p_norm [float] p-value under normality assumption (one-tailed)

p_rand [float] p-value under randomization assumption (one-tailed)

sim [array] (if permutations!=0) vector of I values for permutated samples

p_sim [float] (if permutations!=0) p-value based on permutations (one-tailed) null: sptial randomness alternative: the observed C is extreme it is either extremely high or extremely low

EC_sim [float] (if permutations!=0) average value of C from permutations

VC_sim [float] (if permutations!=0) variance of C from permutations

seC_sim [float] (if permutations!=0) standard deviation of C under permutations.

z_sim [float] (if permutations!=0) standardized C based on permutations

p_z_sim [float] (if permutations!=0) p-value based on standard normal approximation from permutations (one-tailed)

Methods

by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Geary statistic on a dataframe

__init__ (*self*, *y*, *w*, *transformation='r'*, *permutations=999*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, w[, transformation, ...])	Initialize self.
by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Geary statistic on a dataframe

2.3 Getis-Ord Statistics

esda.G(y, w[, permutations])	Global G Autocorrelation Statistic
esda.G_Local(y, w[, transform, ...])	Generalized Local G Autocorrelation

2.3.1 esda.G

class esda.**G**(*y*, *w*, *permutations=999*)

Global G Autocorrelation Statistic

Parameters

y [array (n,1)] Attribute values

w [W] DistanceBand W spatial weights based on distance band

permutations [int] the number of random permutations for calculating pseudo p_values

Notes

Moments are based on normality assumption.

For technical details see [GO10] and [OG10].

Examples

```
>>> import libpysal
>>> import numpy
>>> numpy.random.seed(10)
```

Preparing a point data set

```
>>> points = [(10, 10), (20, 10), (40, 10), (15, 20), (30, 20), (30, 30)]
```

Creating a weights object from points

```
>>> w = libpysal.weights.DistanceBand(points,threshold=15)
>>> w.transform = "B"
```

Preparing a variable

```
>>> y = numpy.array([2, 3, 3.2, 5, 8, 7])
```

Applying Getis and Ord G test

```
>>> from esda.getisord import G
>>> g = G(y,w)
```

Examining the results

```
>>> round(g.G, 3)
0.557
```

```
>>> round(g.p_norm, 3)
0.173
```

Attributes

y [array] original variable

w [W] DistanceBand W spatial weights based on distance band

permutation [int] the number of permutations

G [float] the value of statistic

EG [float] the expected value of statistic

VG [float] the variance of G under normality assumption

z_norm [float] standard normal test statistic

p_norm [float] p-value under normality assumption (one-sided)

sim [array] (if permutations > 0) vector of G values for permutated samples

p_sim [float] p-value based on permutations (one-sided) null: spatial randomness alternative: the observed G is extreme it is either extremely high or extremely low

EG_sim [float] average value of G from permutations

VG_sim [float] variance of G from permutations

seG_sim [float] standard deviation of G under permutations.

z_sim [float] standardized G based on permutations

p_z_sim [float] p-value based on standard normal approximation from permutations (one-sided)

Methods

by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a G statistic on a dataframe

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

Methods

__init__(self, y, w[, permutations])	Initialize self.
by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a G statistic on a dataframe

2.3.2 esda.G_Local

class esda.**G_Local** (*y*, *w*, *transform='R'*, *permutations=999*, *star=False*)
Generalized Local G Autocorrelation

Parameters

y [array] variable

w [W] DistanceBand, weights instance that is based on threshold distance and is assumed to be aligned with y

transform [{'R', 'B'}] the type of w, either 'B' (binary) or 'R' (row-standardized)

permutations [int] the number of random permutations for calculating pseudo p values

star [boolean] whether or not to include focal observation in sums (default: False)

Notes

To compute moments of Gs under normality assumption, PySAL considers w is either binary or row-standardized. For binary weights object, the weight value for self is 1 For row-standardized weights object, the weight value for self is 1/(the number of its neighbors + 1).

For technical details see [GO10] and [OG10].

Examples

```
>>> import libpysal
>>> import numpy
>>> numpy.random.seed(10)
```

Preparing a point data set

```
>>> points = [(10, 10), (20, 10), (40, 10), (15, 20), (30, 20), (30, 30)]
```

Creating a weights object from points

```
>>> w = libpysal.weights.DistanceBand(points,threshold=15)
```

Preparing a variable

```
>>> y = numpy.array([2, 3, 3.2, 5, 8, 7])
```

Applying Getis and Ord local G test using a binary weights object

```
>>> from esda.getisord import G_Local
>>> lg = G_Local(y,w,transform='B')
```

Examining the results

```
>>> lg.Zs
array([-1.0136729 , -0.04361589,  1.31558703, -0.31412676,  1.15373986,
        1.77833941])
>>> round(lg.p_sim[0], 3)
0.101
```

p-value based on standard normal approximation from permutations >>> round(lg.p_z_sim[0], 3) 0.154

```
>>> numpy.random.seed(10)
```

Applying Getis and Ord local G* test using a binary weights object

```
>>> lg_star = G_Local(y,w,transform='B',star=True)
```

Examining the results

```
>>> lg_star.Zs
array([-1.39727626, -0.28917762,  0.65064964, -0.28917762,  1.23452088,
        2.02424331])
>>> round(lg_star.p_sim[0], 3)
0.101
```

```
>>> numpy.random.seed(12345)
```

Applying Getis and Ord local G test using a row-standardized weights object

```
>>> lg = G_Local(y,w,transform='R')
```

Examining the results

```
>>> lg.Zs
array([-0.62074534, -0.01780611,  1.31558703, -0.12824171,  0.28843496,
        1.77833941])
>>> round(lg.p_sim[0], 3)
0.103
```

```
>>> numpy.random.seed(10)
```

Applying Getis and Ord local G* test using a row-standardized weights object

```
>>> lg_star = G_Local(y,w,transform='R',star=True)
```

Examining the results

```
>>> lg_star.Zs
array([-0.62488094, -0.09144599,  0.41150696, -0.09144599,  0.24690418,
        1.28024388])
>>> round(lg_star.p_sim[0], 3)
0.101
```

Attributes

y [array] original variable

w [DistanceBand W] original weights object

permutations [int] the number of permutations

Gs [array] of floats, the value of the orginal G statistic in Getis & Ord (1992)

EGs [float] expected value of Gs under normality assumption the values is scalar, since the expectation is identical across all observations

VGs [array] of floats, variance values of Gs under normality assumption

Zs [array] of floats, standardized Gs

p_norm [array] of floats, p-value under normality assumption (one-sided) for two-sided tests, this value should be multiplied by 2

sim [array] of arrays of floats (if permutations>0), vector of I values for permutated samples

p_sim [array] of floats, p-value based on permutations (one-sided) null - spatial randomness alternative - the observed G is extreme it is either extremely high or extremely low

EG_sim [array] of floats, average value of G from permutations

VG_sim [array] of floats, variance of G from permutations

seG_sim [array] of floats, standard deviation of G under permutations.

z_sim [array] of floats, standardized G based on permutations

p_z_sim [array] of floats, p-value based on standard normal approximation from permutations (one-sided)

Methods

by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a G_Local statistic on a dataframe

calc

__init__(*self*, *y*, *w*, *transform='R'*, *permutations=999*, *star=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, w[, transform, ...])	Initialize self.
by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a G_Local statistic on a dataframe
calc(self)	

2.4 Join Count Statistics

esda.Join_Counts(y, w[, permutations])	Binary Join Counts

2.4.1 esda.Join_Counts

class esda.**Join_Counts**(*y*, *w*, *permutations=999*)
Binary Join Counts

Parameters

y [array] binary variable measured across n spatial units

w [W] spatial weights instance

permutations [int] number of random permutations for calculation of pseudo-p_values

Notes

Technical details and derivations can be found in [CO81].

Examples

```
>>> import numpy as np
>>> import libpysal
>>> w = libpysal.weights.lat2W(4, 4)
>>> y = np.ones(16)
>>> y[0:8] = 0
>>> np.random.seed(12345)
>>> from esda.join_counts import Join_Counts
>>> jc = Join_Counts(y, w)
>>> jc.bb
10.0
>>> jc.bw
4.0
>>> jc.ww
10.0
>>> jc.J
24.0
>>> len(jc.sim_bb)
999
>>> round(jc.p_sim_bb, 3)
0.003
>>> round(np.mean(jc.sim_bb), 3)
5.547
>>> np.max(jc.sim_bb)
10.0
>>> np.min(jc.sim_bb)
0.0
>>> len(jc.sim_bw)
999
>>> jc.p_sim_bw
1.0
>>> np.mean(jc.sim_bw)
12.811811811811811
>>> np.max(jc.sim_bw)
24.0
>>> np.min(jc.sim_bw)
7.0
>>> round(jc.chi2_p, 3)
0.004
>>> jc.p_sim_chi2
0.002
```

Attributes

y [array] original variable

w [W] original w object

permutations [int] number of permutations

bb [float] number of black-black joins

ww [float] number of white-white joins

bw [float] number of black-white joins

J [float] number of joins

sim_bb [array] (if permutations>0) vector of bb values for permuted samples

p_sim_bb [array]

(if permutations>0) p-value based on permutations (one-sided) null: spatial randomness alternative: the observed bb is greater than under randomness

mean_bb [float] average of permuted bb values

min_bb [float] minimum of permuted bb values

max_bb [float] maximum of permuted bb values

sim_bw [array] (if permutations>0) vector of bw values for permuted samples

p_sim_bw [array] (if permutations>0) p-value based on permutations (one-sided) null: spatial randomness alternative: the observed bw is greater than under randomness

mean_bw [float] average of permuted bw values

min_bw [float] minimum of permuted bw values

max_bw [float] maximum of permuted bw values

chi2 [float] Chi-square statistic on contingency table for join counts

chi2_p [float] Analytical p-value for chi2

chi2_dof [int] Degrees of freedom for analytical chi2

crosstab [DataFrame] Contingency table for observed join counts

expected [DataFrame] Expected contingency table for the null

p_sim_chi2 [float] p-value for chi2 under random spatial permutations

Methods

by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Join_Count statistic on a dataframe

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

Methods

__init__(self, y, w[, permutations])	Initialize self.
by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Join_Count statistic on a dataframe

2.5 Moran Statistics

esda.Moran(y, w[, transformation, ...])	Moran's I Global Autocorrelation Statistic
esda.Moran_BV(x, y, w[, transformation, ...])	Bivariate Moran's I
esda.Moran_BV_matrix(variables, w[, ...])	Bivariate Moran Matrix
esda.Moran_Local(y, w[, transformation, ...])	Local Moran Statistics
esda.Moran_Local_BV(x, y, w[, ...])	Bivariate Local Moran Statistics
esda.Moran_Rate(e, b, w[, adjusted, ...])	Adjusted Moran's I Global Autocorrelation Statistic for Rate Variables [AR99]
esda.Moran_Local_Rate(e, b, w[, adjusted, ...])	Adjusted Local Moran Statistics for Rate Variables [Assuncao1999]

2.5.1 esda.Moran

class esda.**Moran**(*y*, *w*, *transformation='r'*, *permutations=999*, *two_tailed=True*)

Moran's I Global Autocorrelation Statistic

Parameters

y [array] variable measured across n spatial units

w [W] spatial weights instance

transformation [string] weights transformation, default is row-standardized "r". Other options include "B": binary, "D": doubly-standardized, "U": untransformed (general weights), "V": variance-stabilizing.

permutations [int] number of random permutations for calculation of pseudo-p_values

two_tailed [boolean] If True (default) analytical p-values for Moran are two tailed, otherwise if False, they are one-tailed.

Notes

Technical details and derivations can be found in [CO81].

Examples

```
>>> import libpysal
>>> w = libpysal.io.open(libpysal.examples.get_path("stl.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("stl_hom.txt"))
>>> y = np.array(f.by_col['HR8893'])
>>> from esda.moran import Moran
>>> mi = Moran(y,  w)
>>> round(mi.I, 3)
0.244
>>> mi.EI
-0.012987012987012988
>>> mi.p_norm
0.00027147862770937614
```

SIDS example replicating OpenGeoda >>> w = libpysal.io.open(libpysal.examples.get_path("sids2.gal")).read() >>> f = libpysal.io.open(libpysal.examples.get_path("sids2.dbf")) >>> SIDR = np.array(f.by_col("SIDR74")) >>> mi = Moran(SIDR, w) >>> round(mi.I, 3) 0.248 >>> mi.p_norm 0.0001158330781489969

One-tailed

```
>>> mi_1 = Moran(SIDR,  w, two_tailed=False)
>>> round(mi_1.I, 3)
0.248
>>> round(mi_1.p_norm, 4)
0.0001
```

Attributes

y [array] original variable

w [W] original w object

permutations [int] number of permutations

I [float] value of Moran's I

EI [float] expected value under normality assumption

VI_norm [float] variance of I under normality assumption

seI_norm [float] standard deviation of I under normality assumption

z_norm [float] z-value of I under normality assumption

p_norm [float] p-value of I under normality assumption

VI_rand [float] variance of I under randomization assumption

seI_rand [float] standard deviation of I under randomization assumption

z_rand [float] z-value of I under randomization assumption

p_rand [float] p-value of I under randomization assumption

two_tailed [boolean] If True p_norm and p_rand are two-tailed, otherwise they are one-tailed.

sim [array] (if permutations>0) vector of I values for permuted samples

p_sim [array] (if permutations>0) p-value based on permutations (one-tailed) null: spatial randomness alternative: the observed I is extreme if it is either extremely greater or extremely lower than the values obtained based on permutations

EI_sim [float] (if permutations>0) average value of I from permutations

VI_sim [float] (if permutations>0) variance of I from permutations

seI_sim [float] (if permutations>0) standard deviation of I under permutations.

z_sim [float] (if permutations>0) standardized I based on permutations

p_z_sim [float] (if permutations>0) p-value based on standard normal approximation from permutations

Methods

by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Moran statistic on a dataframe

__init__ (*self*, *y*, *w*, *transformation='r'*, *permutations=999*, *two_tailed=True*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, w[, transformation, ...])	Initialize self.
by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Moran statistic on a dataframe

2.5.2 esda.Moran_BV

class esda.**Moran_BV**(*x*, *y*, *w*, *transformation='r'*, *permutations=999*)
Bivariate Moran's I

Parameters

x [array] x-axis variable

y [array] wy will be on y axis

w [W] weight instance assumed to be aligned with y

transformation [{'R', 'B', 'D', 'U', 'V'}] weights transformation, default is row-standardized "r". Other options include "B": binary, "D": doubly-standardized, "U": untransformed (general weights), "V": variance-stabilizing.

permutations [int] number of random permutations for calculation of pseudo p_values

Notes

Inference is only based on permutations as analytical results are not too reliable.

Examples

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

Set random number generator seed so we can replicate the example

```
>>> np.random.seed(10)
```

Open the sudden infant death dbf file and read in rates for 74 and 79 converting each to a numpy array

```
>>> f = libpysal.io.open(libpysal.examples.get_path("sids2.dbf"))
>>> SIDR74 = np.array(f.by_col['SIDR74'])
>>> SIDR79 = np.array(f.by_col['SIDR79'])
```

Read a GAL file and construct our spatial weights object

```
>>> w = libpysal.io.open(libpysal.examples.get_path("sids2.gal")).read()
```

Create an instance of Moran_BV >>> from esda.moran import Moran_BV >>> mbi = Moran_BV(SIDR79, SIDR74, w)

What is the bivariate Moran's I value

```
>>> round(mbi.I, 3)
0.156
```

Based on 999 permutations, what is the p-value of our statistic

```
>>> round(mbi.p_z_sim, 3)
0.001
```

Attributes

zx [array] original x variable standardized by mean and std

zy [array] original y variable standardized by mean and std

w [W] original w object

permutation [int] number of permutations

I [float] value of bivariate Moran's I

sim [array] (if permutations>0) vector of I values for permuted samples

p_sim [float] (if permutations>0) p-value based on permutations (one-sided) null: spatial randomness alternative: the observed I is extreme it is either extremely high or extremely low

EI_sim [array] (if permutations>0) average value of I from permutations

VI_sim [array] (if permutations>0) variance of I from permutations

seI_sim [array] (if permutations>0) standard deviation of I under permutations.

z_sim [array] (if permutations>0) standardized I based on permutations

p_z_sim [float] (if permutations>0) p-value based on standard normal approximation from permutations

Methods

by_col(df, x[, y, w, inplace, pvalue, outvals])	Function to compute a Moran_BV statistic on a dataframe

__init__(*self*, *x*, *y*, *w*, *transformation='r'*, *permutations=999*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(self, x, y, w[, transformation, ...])	Initialize self.
`by_col`(df, x[, y, w, inplace, pvalue, outvals])	Function to compute a Moran_BV statistic on a dataframe

2.5.3 esda.Moran_BV_matrix

esda.**Moran_BV_matrix**(*variables*, *w*, *permutations=0*, *varnames=None*)

Bivariate Moran Matrix

Calculates bivariate Moran between all pairs of a set of variables.

Parameters

variables [array or pandas.DataFrame] sequence of variables to be assessed

w [W] a spatial weights object

permutations [int] number of permutations

varnames [list, optional if variables is an array] Strings for variable names. Will add an attribute to *Moran_BV* objects in results needed for plotting in *splot* or *.plot()*. Default =None. Note: If variables is a *pandas.DataFrame* varnames will automatically be generated

Returns

——-

results [dictionary] (i, j) is the key for the pair of variables, values are the Moran_BV objects.

Examples

open dbf

```
>>> import libpysal
>>> f = libpysal.io.open(libpysal.examples.get_path("sids2.dbf"))
```

pull of selected variables from dbf and create numpy arrays for each

```
>>> varnames = ['SIDR74',  'SIDR79',  'NWR74',  'NWR79']
>>> vars = [np.array(f.by_col[var]) for var in varnames]
```

create a contiguity matrix from an external gal file

```
>>> w = libpysal.io.open(libpysal.examples.get_path("sids2.gal")).read()
```

create an instance of Moran_BV_matrix

```
>>> from esda.moran import Moran_BV_matrix
>>> res = Moran_BV_matrix(vars,  w,  varnames = varnames)
```

check values

```
>>> round(res[(0,  1)].I,7)
0.1936261
>>> round(res[(3,  0)].I,7)
0.3770138
```

2.5.4 esda.Moran_Local

class esda.**Moran_Local**(*y*, *w*, *transformation='r'*, *permutations=999*, *geoda_quads=False*)

Local Moran Statistics

Parameters

y [array] (n,1), attribute array

w [W] weight instance assumed to be aligned with y

transformation [{'R', 'B', 'D', 'U', 'V'}] weights transformation, default is row-standardized "r". Other options include "B": binary, "D": doubly-standardized, "U": untransformed (general weights), "V": variance-stabilizing.

permutations [int] number of random permutations for calculation of pseudo p_values

geoda_quads [boolean] (default=False) 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

Notes

For technical details see [Ans95].

Examples

```
>>> import libpysal
>>> import numpy as np
>>> np.random.seed(10)
>>> w = libpysal.io.open(libpysal.examples.get_path("desmith.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("desmith.txt"))
>>> y = np.array(f.by_col['z'])
>>> from esda.moran import Moran_Local
>>> lm = Moran_Local(y, w, transformation = "r", permutations = 99)
>>> lm.q
array([4, 4, 4, 2, 3, 3, 1, 4, 3, 3])
>>> lm.p_z_sim[0]
0.24669152541631179
>>> lm = Moran_Local(y, w, transformation = "r", permutations = 99,
↪              geoda_quads=True)
>>> lm.q
array([4, 4, 4, 3, 2, 2, 1, 4, 2, 2])
```

Note random components result is slightly different values across architectures so the results have been removed from doctests and will be moved into unittests that are conditional on architectures

Attributes

y [array] original variable

w [W] original w object

permutations [int] number of random permutations for calculation of pseudo p_values

Is [array] local Moran's I values

q [array] (if permutations>0) values indicate quandrant location 1 HH, 2 LH, 3 LL, 4 HL

sim [array (permutations by n)] (if permutations>0) I values for permuted samples

p_sim [array] (if permutations>0) p-values based on permutations (one-sided) null: spatial randomness alternative: the observed Ii is further away or extreme from the median of simulated values. It is either extremely high or extremely low in the distribution of simulated Is.

EI_sim [array] (if permutations>0) average values of local Is from permutations

VI_sim [array] (if permutations>0) variance of Is from permutations

seI_sim [array] (if permutations>0) standard deviations of Is under permutations.

z_sim [arrray] (if permutations>0) standardized Is based on permutations

p_z_sim [array] (if permutations>0) p-values based on standard normal approximation from permutations (one-sided) for two-sided tests, these values should be multiplied by 2

Methods

by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Moran_Local statistic on a dataframe

calc

__init__(*self*, *y*, *w*, *transformation='r'*, *permutations=999*, *geoda_quads=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, w[, transformation, ...])	Initialize self.
by_col(df, cols[, w, inplace, pvalue, outvals])	Function to compute a Moran_Local statistic on a dataframe
calc(self, w, z)	

2.5.5 esda.Moran_Local_BV

class esda.**Moran_Local_BV**(*x*, *y*, *w*, *transformation='r'*, *permutations=999*, *geoda_quads=False*)
Bivariate Local Moran Statistics

Parameters

x [array] x-axis variable

y [array] (n,1), wy will be on y axis

w [W] weight instance assumed to be aligned with y

transformation [{'R', 'B', 'D', 'U', 'V'}] weights transformation, default is row-standardized "r". Other options include "B": binary, "D": doubly-standardized, "U": untransformed

(general weights), "V": variance-stabilizing.

permutations [int] number of random permutations for calculation of pseudo p_values

geoda_quads [boolean] (default=False) 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

Examples

```
>>> import libpysal
>>> import numpy as np
>>> np.random.seed(10)
>>> w = libpysal.io.open(libpysal.examples.get_path("sids2.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("sids2.dbf"))
>>> x = np.array(f.by_col['SIDR79'])
>>> y = np.array(f.by_col['SIDR74'])
>>> from esda.moran import Moran_Local_BV
>>> lm =Moran_Local_BV(x, y, w, transformation = "r",
↪   permutations = 99)
>>> lm.q[:10]
array([3, 4, 3, 4, 2, 1, 4, 4, 2, 4])
>>> lm = Moran_Local_BV(x, y, w, transformation = "r",
↪    permutations = 99, geoda_quads=True)
>>> lm.q[:10]
array([2, 4, 2, 4, 3, 1, 4, 4, 3, 4])
```

Note random components result is slightly different values across architectures so the results have been removed from doctests and will be moved into unittests that are conditional on architectures

Attributes

zx [array] original x variable standardized by mean and std

zy [array] original y variable standardized by mean and std

w [W] original w object

permutations [int] number of random permutations for calculation of pseudo p_values

Is [float] value of Moran's I

q [array] (if permutations>0) values indicate quandrant location 1 HH, 2 LH, 3 LL, 4 HL

sim [array] (if permutations>0) vector of I values for permuted samples

p_sim [array] (if permutations>0) p-value based on permutations (one-sided) null: spatial randomness alternative: the observed Ii is further away or extreme from the median of simulated values. It is either extremelyi high or extremely low in the distribution of simulated Is.

EI_sim [array] (if permutations>0) average values of local Is from permutations

VI_sim [array] (if permutations>0) variance of Is from permutations

seI_sim [array] (if permutations>0) standard deviations of Is under permutations.

z_sim [arrray] (if permutations>0) standardized Is based on permutations

p_z_sim [array] (if permutations>0) p-values based on standard normal approximation from permutations (one-sided) for two-sided tests, these values should be multiplied by 2

Methods

by_col(df, x[, y, w, inplace, pvalue, outvals])	Function to compute a Moran_Local_BV statistic on a dataframe

calc

__init__(*self*, *x*, *y*, *w*, *transformation='r'*, *permutations=999*, *geoda_quads=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, x, y, w[, transformation, ...])	Initialize self.
by_col(df, x[, y, w, inplace, pvalue, outvals])	Function to compute a Moran_Local_BV statistic on a dataframe
calc(self, w, zx, zy)	

2.5.6 esda.Moran_Rate

class esda.**Moran_Rate**(*e*, *b*, *w*, *adjusted=True*, *transformation='r'*, *permutations=999*, *two_tailed=True*)
Adjusted Moran's I Global Autocorrelation Statistic for Rate Variables [AR99]

Parameters

e [array] an event variable measured across n spatial units

b [array] a population-at-risk variable measured across n spatial units

w [W] spatial weights instance

adjusted [boolean] whether or not Moran's I needs to be adjusted for rate variable

transformation [{'R', 'B', 'D', 'U', 'V'}] weights transformation, default is row-standardized "r". Other options include "B": binary, "D": doubly-standardized, "U": untransformed (general weights), "V": variance-stabilizing.

two_tailed [boolean] If True (default), analytical p-values for Moran's I are two-tailed, otherwise they are one tailed.

permutations [int] number of random permutations for calculation of pseudo p_values

Examples

```
>>> import libpysal
>>> w = libpysal.io.open(libpysal.examples.get_path("sids2.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("sids2.dbf"))
>>> e = np.array(f.by_col('SID79'))
>>> b = np.array(f.by_col('BIR79'))
>>> from esda.moran import Moran_Rate
>>> mi = Moran_Rate(e, b,  w, two_tailed=False)
>>> "%6.4f" % mi.I
```

(continues on next page)

(continued from previous page)

```
'0.1662'
>>> "%6.4f" % mi.p_norm
'0.0042'
```

Attributes

y [array] rate variable computed from parameters e and b if adjusted is True, y is standardized rates otherwise, y is raw rates

w [W] original w object

permutations [int] number of permutations

I [float] value of Moran's I

EI [float] expected value under normality assumption

VI_norm [float] variance of I under normality assumption

seI_norm [float] standard deviation of I under normality assumption

z_norm [float] z-value of I under normality assumption

p_norm [float] p-value of I under normality assumption

VI_rand [float] variance of I under randomization assumption

seI_rand [float] standard deviation of I under randomization assumption

z_rand [float] z-value of I under randomization assumption

p_rand [float] p-value of I under randomization assumption

two_tailed [boolean] If True, p_norm and p_rand are two-tailed p-values, otherwise they are one-tailed.

sim [array] (if permutations>0) vector of I values for permuted samples

p_sim [array] (if permutations>0) p-value based on permutations (one-sided) null: spatial randomness alternative: the observed I is extreme if it is either extremely greater or extremely lower than the values obtained from permutaitons

EI_sim [float] (if permutations>0) average value of I from permutations

VI_sim [float] (if permutations>0) variance of I from permutations

seI_sim [float] (if permutations>0) standard deviation of I under permutations.

z_sim [float] (if permutations>0) standardized I based on permutations

p_z_sim [float] (if permutations>0) p-value based on standard normal approximation from

Methods

`by_col`(df, events, populations[, w, ...])	Function to compute a Moran_Rate statistic on a dataframe

__init__(*self*, *e*, *b*, *w*, *adjusted=True*, *transformation='r'*, *permutations=999*, *two_tailed=True*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(self, e, b, w[, adjusted, ...])	Initialize self.
`by_col`(df, events, populations[, w, ...])	Function to compute a Moran_Rate statistic on a dataframe

2.5.7 esda.Moran_Local_Rate

class esda.**Moran_Local_Rate**(*e*, *b*, *w*, *adjusted=True*, *transformation='r'*, *permutations=999*, *geoda_quads=False*)

Adjusted Local Moran Statistics for Rate Variables [Assuncao1999]

Parameters

e [array] (n,1), an event variable across n spatial units

b [array] (n,1), a population-at-risk variable across n spatial units

w [W] weight instance assumed to be aligned with y

adjusted [boolean] whether or not local Moran statistics need to be adjusted for rate variable

transformation [{'R', 'B', 'D', 'U', 'V'}] weights transformation, default is row-standardized "r". Other options include "B": binary, "D": doubly-standardized, "U": untransformed (general weights), "V": variance-stabilizing.

permutations [int] number of random permutations for calculation of pseudo p_values

geoda_quads [boolean] (default=False) 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

Attributes

———-

y [array] rate variables computed from parameters e and b if adjusted is True, y is standardized rates otherwise, y is raw rates

w [W] original w object

permutations [int] number of random permutations for calculation of pseudo p_values

I [float] value of Moran's I

q [array] (if permutations>0) values indicate quandrant location 1 HH, 2 LH, 3 LL, 4 HL

sim [array] (if permutations>0) vector of I values for permuted samples

p_sim [array] (if permutations>0) p-value based on permutations (one-sided) null: spatial randomness alternative: the observed Ii is further away or extreme from the median of simulated Iis. It is either extremely high or extremely low in the distribution of simulated Is

EI_sim [float] (if permutations>0) average value of I from permutations

VI_sim [float] (if permutations>0) variance of I from permutations

seI_sim [float] (if permutations>0) standard deviation of I under permutations.

z_sim [float] (if permutations>0) standardized I based on permutations

p_z_sim [float] (if permutations>0) p-value based on standard normal approximation from permutations (one-sided) for two-sided tests, these values should be multiplied by 2

Examples

```
>>> import libpysal
>>> import numpy as np
>>> np.random.seed(10)
>>> w = libpysal.io.open(libpysal.examples.get_path("sids2.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("sids2.dbf"))
>>> e = np.array(f.by_col('SID79'))
>>> b = np.array(f.by_col('BIR79'))
>>> from esda.moran import Moran_Local_Rate
>>> lm = Moran_Local_Rate(e, b, w, transformation = "r", permutations = 99)
>>> lm.q[:10]
array([2, 4, 3, 1, 2, 1, 1, 4, 2, 4])
>>> lm = Moran_Local_Rate(e, b, w,  transformation = "r",  permutations = 99, ␣
↪geoda_quads=True)
>>> lm.q[:10]
array([3, 4, 2, 1, 3, 1, 1, 4, 3, 4])
```

Note random components result is slightly different values across architectures so the results have been removed from doctests and will be moved into unittests that are conditional on architectures

Methods

by_col(df, events, populations[, w, ...])	Function to compute a Moran_Local_Rate statistic on a dataframe

calc

__init__(*self*, *e*, *b*, *w*, *adjusted=True*, *transformation='r'*, *permutations=999*, *geoda_quads=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, e, b, w[, adjusted, ...])	Initialize self.
by_col(df, events, populations[, w, ...])	Function to compute a Moran_Local_Rate statistic on a dataframe
calc(self, w, z)	

2.6 Spatial Pearson Statistics

esda.Spatial_Pearson([connectivity, ...])	Global Spatial Pearson Statistic
esda.Local_Spatial_Pearson([connectivity, ...])	Local Spatial Pearson Statistic

2.6.1 esda.Spatial_Pearson

class esda.**Spatial_Pearson**(*connectivity=None*, *permutations=999*)
Global Spatial Pearson Statistic

Methods

fit(self, x, y)	bivariate spatial pearson's R based on Eq.
get_params(self[, deep])	Get parameters for this estimator.
set_params(self, **params)	Set the parameters of this estimator.

__init__(*self*, *connectivity=None*, *permutations=999*)
Initialize a spatial pearson estimator

Attributes

association_: numpy.ndarray (2,2) array containg the estimated Lee spatial pearson correlation coefficients, where element [0,1] is the spatial correlation coefficient, and elements [0,0] and [1,1] are the "spatial smoothing factor"

reference_distribution_: numpy.ndarray (n_permutations, 2,2) distribution of correlation matrices for randomly-shuffled maps.

significance_: numpy.ndarray (2,2) permutation-based p-values for the fraction of times the observed correlation was more extreme than the simulated correlations.

Methods

__init__(self[, connectivity, permutations])	Initialize a spatial pearson estimator
fit(self, x, y)	bivariate spatial pearson's R based on Eq.
get_params(self[, deep])	Get parameters for this estimator.
set_params(self, **params)	Set the parameters of this estimator.

2.6.2 esda.Local_Spatial_Pearson

class esda.**Local_Spatial_Pearson**(*connectivity=None*, *permutations=999*)
Local Spatial Pearson Statistic

Methods

fit(self, x, y)	bivariate local pearson's R based on Eq.
get_params(self[, deep])	Get parameters for this estimator.
set_params(self, **params)	Set the parameters of this estimator.

__init__(*self*, *connectivity=None*, *permutations=999*)
Initialize a spatial local pearson estimator

Notes

Technical details and derivations can be found in [Lee01].

Methods

__init__(self[, connectivity, permutations])	Initialize a spatial local pearson estimator
fit(self, x, y)	bivariate local pearson's R based on Eq.
get_params(self[, deep])	Get parameters for this estimator.
set_params(self, **params)	Set the parameters of this estimator.

2.7 Modifiable Areal Unit Tests

esda.Smaup(n, k, rho)	S-maup: Statistical Test to Measure the Sensitivity to the Modifiable Areal Unit Problem

2.7.1 esda.Smaup

class esda.**Smaup**(*n*, *k*, *rho*)

S-maup: Statistical Test to Measure the Sensitivity to the Modifiable Areal Unit Problem

Parameters

n [int] number of spatial units

k [int] number of regions

rho [float] rho value (level of spatial autocorrelation) ranges from -1 to 1

Notes

Technical details and derivations can be found in [DLP18].

Examples

```
>>> import libpysal
>>> import numpy as np
>>> from esda.moran import Moran
>>> from esda.smaup import Smaup
>>> w = libpysal.io.open(libpysal.examples.get_path("stl.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("stl_hom.txt"))
>>> y = np.array(f.by_col['HR8893'])
>>> rho = Moran(y,  w).I
>>> n = len(y)
>>> k = int(n/2)
>>> s = Smaup(n,k,rho)
>>> s.smaup
0.15221341690376405
>>> s.critical_01
```

(continues on next page)

(continued from previous page)

```
0.38970613333333337
>>> s.critical_05
0.3557221333333333
>>> s.critical_1
0.3157950666666666
>>> s.summary
'Pseudo p-value > 0.10 (H0 is not rejected)'
```

SIDS example replicating OpenGeoda

```
>>> w = libpysal.io.open(libpysal.examples.get_path("sids2.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("sids2.dbf"))
>>> SIDR = np.array(f.by_col("SIDR74"))
>>> from esda.moran import Moran
>>> rho = Moran(SIDR,  w).I
>>> n = len(y)
>>> k = int(n/2)
>>> s = Smaup(n,k,rho)
>>> s.smaup
0.15176796553181948
>>> s.critical_01
0.38970613333333337
>>> s.critical_05
0.3557221333333333
>>> s.critical_1
0.3157950666666666
>>> s.summary
'Pseudo p-value > 0.10 (H0 is not rejected)'
```

Attributes

n [int] number of spatial units

k [int] number of regions

rho [float] rho value (level of spatial autocorrelation) ranges from -1 to 1

smaup [float] : S-maup statistic (M)

critical_01 [float] : critical value at 0.99 confidence level

critical_05 [float] : critical value at 0.95 confidence level

critical_1 [float] : critical value at 0.90 confidence level

summary [string] : message with interpretation of results

__init__(*self*, *n*, *k*, *rho*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, n, k, rho)	Initialize self.

2.8 Utility Functions

esda.fdr(pvalues[, alpha])	Calculate the p-value cut-off to control for the false discovery rate (FDR) for multiple testing.

2.8.1 esda.fdr

esda.**fdr**(*pvalues*, *alpha=0.05*)

Calculate the p-value cut-off to control for the false discovery rate (FDR) for multiple testing.

If by controlling for FDR, all of n null hypotheses are rejected, the conservative Bonferroni bound (alpha/n) is returned instead.

Returns

: float Adjusted criterion for rejecting the null hypothesis. If by controlling for FDR, all of n null hypotheses are rejected, the conservative Bonferroni bound (alpha/n) is returned.

Notes

For technical details see [BY01] and [dCS06].

Examples

```
>>> import libpysal
>>> import numpy as np
>>> np.random.seed(10)
>>> w = libpysal.io.open(libpysal.examples.get_path("stl.gal")).read()
>>> f = libpysal.io.open(libpysal.examples.get_path("stl_hom.txt"))
>>> y = np.array(f.by_col['HR8893'])
>>> from esda.moran import Moran_Local
>>> from esda import fdr
>>> lm = Moran_Local(y, w, transformation = "r", permutations = 999)
>>> fdr(lm.p_sim, 0.1)
0.002564102564102564
>>> fdr(lm.p_sim, 0.05) #return the conservative Bonferroni bound
0.000641025641025641
```

CHAPTER

THREE

REFERENCES

CHAPTER

FOUR

INTRODUCTION

esda implements measures for the exploratory analysis spatial data and is part of the PySAL family

Details are available in the esda api.

CHAPTER

FIVE

DEVELOPMENT

esda development is hosted on github.

Discussions of development occurs on the developer list as well as gitter.

CHAPTER

SIX

GETTING INVOLVED

If you are interested in contributing to PySAL please see our development guidelines.

CHAPTER

SEVEN

BUG REPORTS

To search for or report bugs, please see esda's issues.

CHAPTER

EIGHT

CITING ESDA

If you use PySAL-esda in a scientific publication, we would appreciate citations to the following paper:

> PySAL: A Python Library of Spatial Analytical Methods, *Rey, S.J. and L. Anselin*, Review of Regional Studies 37, 5-27 2007.
>
> Bibtex entry:

```
@Article{pysal2007,
  author={Rey, Sergio J. and Anselin, Luc},
  title={{PySAL: A Python Library of Spatial Analytical Methods}},
  journal={The Review of Regional Studies},
  year=2007,
  volume={37},
  number={1},
  pages={5-27},
  keywords={Open Source; Software; Spatial}
}
```

CHAPTER

NINE

LICENSE INFORMATION

See the file “LICENSE.txt” for information on the history of this software, terms & conditions for usage, and a DISCLAIMER OF ALL WARRANTIES.

BIBLIOGRAPHY

[Ans95] Luc Anselin. Local indicators of spatial association-LISA. *Geographical Analysis*, 27(2):93–115, Sep 1995. URL: http://dx.doi.org/10.1111/j.1538-4632.1995.tb00338.x, doi:10.1111/j.1538-4632.1995.tb00338.x.

[AR99] Renato M. Assuncao and Edna A. Reis. A new proposal to adjust Moran's I for population density. *Statistics in Medicine*, 18(16):2147–2162, Aug 1999. URL: http://dx.doi.org/10.1002/(sici)1097-0258(19990830)18:16\T1\textless{}2147::aid-sim179\T1\textgreater{}3.0.co;2-i, doi:10.1002/(sici)1097-0258(19990830)18:16<2147::aid-sim179>3.0.co;2-i.

[BY01] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics*, 29(4):1165–1188, 2001. URL: http://www.jstor.org/stable/2674075.

[CO81] A.D. Cliff and J.K. Ord. *Spatial Processes: Models and Applications*. Pion, London, 1981.

[dCS06] Marcia Caldas de Castro and Burton H. Singer. Controlling the false discovery rate: a new application to account for multiple and dependent tests in local statistics of spatial association. *Geographical Analysis*, 38(2):180–208, April 2006. URL: http://dx.doi.org/10.1111/j.0016-7363.2006.00682.x, doi:10.1111/j.0016-7363.2006.00682.x.

[DLP18] Juan C. Duque, H. Laniado, and A. Polo. S-maup: statistical test to measure the sensitivity to the modifiable areal unit problem. *PLOS ONE*, 13(11):1–25, 11 2018. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0207377, doi:https://doi.org/10.1371/journal.pone.0207377.

[GO10] Arthur Getis and J. K. Ord. The analysis of spatial association by use of distance statistics. *Geographical Analysis*, 24(3):189–206, Sep 2010. URL: http://dx.doi.org/10.1111/j.1538-4632.1992.tb00261.x, doi:10.1111/j.1538-4632.1992.tb00261.x.

[HGC81] L. J. Hubert, R. G. Golledge, and C. M. Costanzo. Generalized procedures for evaluating spatial autocorrelation. *Geographical Analysis*, 13(3):224–233, Sep 1981. URL: http://dx.doi.org/10.1111/j.1538-4632.1981.tb00731.x, doi:10.1111/j.1538-4632.1981.tb00731.x.

[Lee01] Sang-Il Lee. Developing a bivariate spatial association measure: an integration of Pearson's r and Moran's I. *Journal of Geographical Systems*, 3(4):369–385, Dec 2001. URL: https://doi.org/10.1007/s101090100064, doi:10.1007/s101090100064.

[OG10] J. K. Ord and Arthur Getis. Local spatial autocorrelation statistics: distributional issues and an application. *Geographical Analysis*, 27(4):286–306, Sep 2010. URL: http://dx.doi.org/10.1111/j.1538-4632.1995.tb00912.x, doi:10.1111/j.1538-4632.1995.tb00912.x.

INDEX

Symbols

F

G

J

L

M

S