spreg Documentation

Release 1.1.0

pysal developers

Dec 03, 2019

CONTENTS:

spreg, short for "spatial regression," is a python package to estimate simultaneous autoregressive spatial regression models. These models are useful when modeling processes where observations interact with one another. For more information on these models, consult the Spatial Regression short course by Luc Anselin (Spring, 2017), with the Center for Spatial Data Science at the University of Chicago:

CHAPTER

ONE

INSTALLATION

spreg is installable using the Python Package Manager, *pip*. To install:

```
pip install spreg
```

Further, all of the stable functionality is *also* available in PySAL, the Python Spatial Analysis Library. PySAL can be installed using *pip* or *conda*:

```
pip install pysal #or
conda install pysal
```

CHAPTER

TWO

API REFERENCE

CHAPTER

THREE

SPATIAL REGRESSION MODELS

These are the standard spatial regression models supported by the *spreg* package. Each of them contains a significant amount of detail in their docstring discussing how they're used, how they're fit, and how to interpret the results.

spreg.OLS(y, x[, w, robust, gwk, sig2n_k, ...])	Ordinary least squares with results and diagnostics.
spreg.ML_Lag(y, x, w[, method, epsilon, ...])	ML estimation of the spatial lag model with all results and diagnostics; [Ans88]
spreg.ML_Error(y, x, w[, method, epsilon, ...])	ML estimation of the spatial error model with all results and diagnostics; [Ans88]
spreg.GM_Lag(y, x[, yend, q, w, w_lags, ...])	Spatial two stage least squares (S2SLS) with results and diagnostics; Anselin (1988) [Ans88]
spreg.GM_Error(y, x, w[, vm, name_y, ...])	GMM method for a spatial error model, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].
spreg.GM_Error_Het(y, x, w[, max_iter, ...])	GMM method for a spatial error model with heteroskedasticity, with results and diagnostics; based on [ADKP10], following [Ans11].
spreg.GM_Error_Hom(y, x, w[, max_iter, ...])	GMM method for a spatial error model with homoskedasticity, with results and diagnostics; based on Drukker et al.
spreg.GM_Combo(y, x[, yend, q, w, w_lags, ...])	GMM method for a spatial lag and error model with endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].
spreg.GM_Combo_Het(y, x[, yend, q, w, ...])	GMM method for a spatial lag and error model with heteroskedasticity and endogenous variables, with results and diagnostics; based on [ADKP10], following [Ans11].
spreg.GM_Combo_Hom(y, x[, yend, q, w, ...])	GMM method for a spatial lag and error model with homoskedasticity and endogenous variables, with results and diagnostics; based on Drukker et al.
spreg.GM_Endog_Error(y, x, yend, q, w[, vm, ...])	GMM method for a spatial error model with endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].
spreg.GM_Endog_Error_Het(y, x, yend, q, w[, ...])	GMM method for a spatial error model with heteroskedasticity and endogenous variables, with results and diagnostics; based on [ADKP10], following [Ans11].
spreg.GM_Endog_Error_Hom(y, x, yend, q, w[, ...])	GMM method for a spatial error model with homoskedasticity and endogenous variables, with results and diagnostics; based on Drukker et al.

Continued on next page

Table 1 – continued from previous page

spreg.TSLS(y, x, yend, q[, w, robust, gwk, ...])	Two stage least squares with results and diagnostics.
spreg.ThreeSLS(bigy, bigX, bigyend, bigq[, ...])	User class for 3SLS estimation

3.1 spreg.OLS

class spreg.**OLS**(*y*, *x*, *w=None*, *robust=None*, *gwk=None*, *sig2n_k=True*, *nonspat_diag=True*, *spat_diag=False*, *moran=False*, *white_test=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)

Ordinary least squares with results and diagnostics.

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

w [pysal W object] Spatial weights object (required if running spatial diagnostics)

robust [string] If 'white', then a White consistent estimator of the variance-covariance matrix is given. If 'hac', then a HAC consistent estimator of the variance-covariance matrix is given. Default set to None.

gwk [pysal W object] Kernel spatial weights needed for HAC estimation. Note: matrix must have ones along the main diagonal.

sig2n_k [boolean] If True, then use n-k to estimate sigma^2. If False, use n.

nonspat_diag [boolean] If True, then compute non-spatial diagnostics on the regression.

spat_diag [boolean] If True, then compute Lagrange multiplier tests (requires w). Note: see moran for further tests.

moran [boolean] If True, compute Moran's I on the residuals. Note: requires spat_diag=True.

white_test [boolean] If True, compute White's specification robust test. (requires non-spat_diag=True)

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; also, the actual OLS class requires data to be passed in as numpy arrays so the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an nx1 numpy array.

```
>>> hoval = db.by_col("HOVAL")
>>> y = np.array(hoval)
>>> y.shape = (len(hoval), 1)
```

Extract CRIME (crime) and INC (income) vectors from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). spreg.OLS adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("CRIME"))
>>> X = np.array(X).T
```

The minimum parameters needed to run an ordinary least squares regression are the two numpy arrays containing the independent variable and dependent variables respectively. To make the printed results more meaningful, the user can pass in explicit names for the variables used; this is optional.

```
>>> ols = OLS(y, X, name_y='home value', name_x=['income','crime'], name_ds=
↪'columbus', white_test=True)
```

spreg.OLS computes the regression coefficients and their standard errors, t-stats and p-values. It also computes a large battery of diagnostics on the regression. In this example we compute the white test which by default isn't ('white_test=True'). All of these results can be independently accessed as attributes of the regression object created by running spreg.OLS. They can also be accessed at one time by printing the summary attribute of the regression object. In the example below, the parameter on crime is -0.4849, with a t-statistic of -2.6544 and p-value of 0.01087.

```
>>> ols.betas
array([[ 46.42818268],
       [  0.62898397],
       [ -0.48488854]])
>>> print round(ols.t_stat[2][0],3)
-2.654
>>> print round(ols.t_stat[2][1],3)
0.011
>>> print round(ols.r2,3)
0.35
```

Or we can easily obtain a full summary of all the results nicely formatted and ready to be printed:

```
>>> print ols.summary
REGRESSION
----------
SUMMARY OF OUTPUT: ORDINARY LEAST SQUARES
-----------------------------------------
Data set            :    columbus
Dependent Variable  :  home value                Number of Observations:          ␣
↪49
Mean dependent var  :     38.4362                Number of Variables   :          ␣
↪ 3
S.D. dependent var  :     18.4661                Degrees of Freedom    :          ␣
↪46
```

(continues on next page)

(continued from previous page)

```
R-squared           :      0.3495
Adjusted R-squared  :      0.3212
Sum squared residual:   10647.015                F-statistic           :     12.
↪3582
Sigma-square        :     231.457                Prob(F-statistic)     :   5.064e-
↪05
S.E. of regression  :      15.214                Log likelihood        :    -201.
↪368
Sigma-square ML     :     217.286                Akaike info criterion :     408.
↪735
S.E of regression ML:     14.7406                Schwarz criterion     :     414.
↪411

------------------------------------------------------------------------------
↪--
            Variable     Coefficient       Std.Error     t-Statistic
↪Probability
------------------------------------------------------------------------------
↪--
            CONSTANT      46.4281827      13.1917570       3.5194844       0.
↪0009867
               crime      -0.4848885       0.1826729      -2.6544086       0.
↪0108745
              income       0.6289840       0.5359104       1.1736736       0.
↪2465669
------------------------------------------------------------------------------
↪--

REGRESSION DIAGNOSTICS
MULTICOLLINEARITY CONDITION NUMBER           12.538

TEST ON NORMALITY OF ERRORS
TEST                             DF        VALUE           PROB
Jarque-Bera                       2           39.706           0.0000

DIAGNOSTICS FOR HETEROSKEDASTICITY
RANDOM COEFFICIENTS
TEST                             DF        VALUE           PROB
Breusch-Pagan test                2           5.767           0.0559
Koenker-Bassett test              2           2.270           0.3214

SPECIFICATION ROBUST TEST
TEST                             DF        VALUE           PROB
White                             5           2.906           0.7145
================================ END OF REPORT
↪====================================
```

If the optional parameters w and spat_diag are passed to spreg.OLS, spatial diagnostics will also be computed for the regression. These include Lagrange multiplier tests and Moran's I of the residuals. The w parameter is a PySAL spatial weights matrix. In this example, w is built directly from the shapefile columbus.shp, but w can also be read in from a GAL or GWT file. In this case a rook contiguity weights matrix is built, but PySAL also offers queen contiguity, distance weights and k nearest neighbor weights among others. In the example, the Moran's I of the residuals is 0.204 with a standardized value of 2.592 and a p-value of 0.0095.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

(continues on next page)

(continued from previous page)

```
>>> ols = OLS(y, X, w, spat_diag=True, moran=True, name_y='home value', name_x=[
↪'income','crime'], name_ds='columbus')
>>> ols.betas
array([[ 46.42818268],
       [  0.62898397],
       [ -0.48488854]])
>>> print round(ols.moran_res[0],3)
0.204
>>> print round(ols.moran_res[1],3)
2.592
>>> print round(ols.moran_res[2],4)
0.0095
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

robust [string] Adjustment for robust standard errors

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

r2 [float] R squared

ar2 [float] Adjusted R squared

utu [float] Sum of squared residuals

sig2 [float] Sigma squared used in computations

sig2ML [float] Sigma squared (maximum likelihood)

f_stat [tuple] Statistic (float), p-value (float)

logll [float] Log likelihood

aic [float] Akaike information criterion

schwarz [float] Schwarz information criterion

std_err [array] 1xk array of standard errors of the betas

t_stat [list of tuples] t statistic; each tuple contains the pair (statistic, p-value), where each is a float

mulColli [float] Multicollinearity condition number

jarque_bera [dictionary] 'jb': Jarque-Bera statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int)

breusch_pagan [dictionary] 'bp': Breusch-Pagan statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int)

koenker_bassett [dictionary] 'kb': Koenker-Bassett statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int)

white [dictionary] 'wh': White statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int)

lm_error [tuple] Lagrange multiplier test for spatial error model; tuple contains the pair (statistic, p-value), where each is a float

lm_lag [tuple] Lagrange multiplier test for spatial lag model; tuple contains the pair (statistic, p-value), where each is a float

rlm_error [tuple] Robust lagrange multiplier test for spatial error model; tuple contains the pair (statistic, p-value), where each is a float

rlm_lag [tuple] Robust lagrange multiplier test for spatial lag model; tuple contains the pair (statistic, p-value), where each is a float

lm_sarma [tuple] Lagrange multiplier test for spatial SARMA model; tuple contains the pair (statistic, p-value), where each is a float

moran_res [tuple] Moran's I for the residuals; tuple containing the triple (Moran's I, standardized Moran's I, p-value)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

sig2n [float] Sigma squared (computed with n in the denominator)

sig2n_k [float] Sigma squared (computed with n-k in the denominator)

xtx [float] $X'X$

xtxi [float] $(X'X)^{-1}$

__init__(*self*, *y*, *x*, *w=None*, *robust=None*, *gwk=None*, *sig2n_k=True*, *nonspat_diag=True*, *spat_diag=False*, *moran=False*, *white_test=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x[, w, robust, gwk, ...])	Initialize self.

Attributes

mean_y
sig2n
sig2n_k
std_y
utu
vm

3.2 spreg.ML_Lag

class spreg.**ML_Lag**(*y*, *x*, *w*, *method='full'*, *epsilon=1e-07*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)

ML estimation of the spatial lag model with all results and diagnostics; [Ans88]

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

w [pysal W object] Spatial weights object

method [string] if 'full', brute force calculation (full matrix expressions) if 'ord', Ord eigenvalue method

epsilon [float] tolerance criterion in mimimize_scalar function and inverse_product

spat_diag [boolean] if True, include spatial diagnostics

vm [boolean] if True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

```
>>> import numpy as np
>>> import libpysal
>>> db =  libpysal.io.open(libpysal.examples.get_path("baltim.dbf"),'r')
>>> ds_name = "baltim.dbf"
>>> y_name = "PRICE"
>>> y = np.array(db.by_col(y_name)).T
>>> y.shape = (len(y),1)
>>> x_names = ["NROOM","NBATH","PATIO","FIREPL","AC","GAR","AGE","LOTSZ","SQFT"]
>>> x = np.array([db.by_col(var) for var in x_names]).T
>>> ww = ps.open(ps.examples.get_path("baltim_q.gal"))
>>> w = ww.read()
>>> ww.close()
>>> w_name = "baltim_q.gal"
>>> w.transform = 'r'
>>> mllag = ML_Lag(y,x,w,name_y=y_name,name_x=x_names,                name_w=w_
↪name,name_ds=ds_name)
>>> np.around(mllag.betas, decimals=4)
```

(continues on next page)

(continued from previous page)

```
array([[ 4.3675],
       [ 0.7502],
       [ 5.6116],
       [ 7.0497],
       [ 7.7246],
       [ 6.1231],
       [ 4.6375],
       [-0.1107],
       [ 0.0679],
       [ 0.0794],
       [ 0.4259]])
>>> "{0:.6f}".format(mllag.rho)
'0.425885'
>>> "{0:.6f}".format(mllag.mean_y)
'44.307180'
>>> "{0:.6f}".format(mllag.std_y)
'23.606077'
>>> np.around(np.diag(mllag.vm1), decimals=4)
array([  23.8716,    1.1222,    3.0593,    7.3416,    5.6695,    5.4698,
          2.8684,    0.0026,    0.0002,    0.0266,    0.0032,  220.1292])
>>> np.around(np.diag(mllag.vm), decimals=4)
array([ 23.8716,   1.1222,   3.0593,   7.3416,   5.6695,   5.4698,
         2.8684,   0.0026,   0.0002,   0.0266,   0.0032])
>>> "{0:.6f}".format(mllag.sig2)
'151.458698'
>>> "{0:.6f}".format(mllag.logll)
'-832.937174'
>>> "{0:.6f}".format(mllag.aic)
'1687.874348'
>>> "{0:.6f}".format(mllag.schwarz)
'1724.744787'
>>> "{0:.6f}".format(mllag.pr2)
'0.727081'
>>> "{0:.4f}".format(mllag.pr2_e)
'0.7062'
>>> "{0:.4f}".format(mllag.utu)
'31957.7853'
>>> np.around(mllag.std_err, decimals=4)
array([ 4.8859,  1.0593,  1.7491,  2.7095,  2.3811,  2.3388,  1.6936,
        0.0508,  0.0146,  0.1631,  0.057 ])
>>> np.around(mllag.z_stat, decimals=4)
array([[ 0.8939,  0.3714],
       [ 0.7082,  0.4788],
       [ 3.2083,  0.0013],
       [ 2.6018,  0.0093],
       [ 3.2442,  0.0012],
       [ 2.6181,  0.0088],
       [ 2.7382,  0.0062],
       [-2.178 ,  0.0294],
       [ 4.6487,  0.    ],
       [ 0.4866,  0.6266],
       [ 7.4775,  0.    ]])
>>> mllag.name_y
'PRICE'
>>> mllag.name_x
['CONSTANT', 'NROOM', 'NBATH', 'PATIO', 'FIREPL', 'AC', 'GAR', 'AGE', 'LOTSZ',
↪'SQFT', 'W_PRICE']
```

(continues on next page)

(continued from previous page)

```
>>> mllag.name_w
'baltim_q.gal'
>>> mllag.name_ds
'baltim.dbf'
>>> mllag.title
'MAXIMUM LIKELIHOOD SPATIAL LAG (METHOD = FULL)'
>>> mllag = ML_Lag(y,x,w,method='ord',name_y=y_name,name_x=x_names,
↪name_w=w_name,name_ds=ds_name)
>>> np.around(mllag.betas, decimals=4)
array([[ 4.3675],
       [ 0.7502],
       [ 5.6116],
       [ 7.0497],
       [ 7.7246],
       [ 6.1231],
       [ 4.6375],
       [-0.1107],
       [ 0.0679],
       [ 0.0794],
       [ 0.4259]])
>>> "{0:.6f}".format(mllag.rho)
'0.425885'
>>> "{0:.6f}".format(mllag.mean_y)
'44.307180'
>>> "{0:.6f}".format(mllag.std_y)
'23.606077'
>>> np.around(np.diag(mllag.vm1), decimals=4)
array([  23.8716,    1.1222,    3.0593,    7.3416,    5.6695,    5.4698,
          2.8684,    0.0026,    0.0002,    0.0266,    0.0032,  220.1292])
>>> np.around(np.diag(mllag.vm), decimals=4)
array([ 23.8716,   1.1222,   3.0593,   7.3416,   5.6695,   5.4698,
         2.8684,   0.0026,   0.0002,   0.0266,   0.0032])
>>> "{0:.6f}".format(mllag.sig2)
'151.458698'
>>> "{0:.6f}".format(mllag.logll)
'-832.937174'
>>> "{0:.6f}".format(mllag.aic)
'1687.874348'
>>> "{0:.6f}".format(mllag.schwarz)
'1724.744787'
>>> "{0:.6f}".format(mllag.pr2)
'0.727081'
>>> "{0:.6f}".format(mllag.pr2_e)
'0.706198'
>>> "{0:.4f}".format(mllag.utu)
'31957.7853'
>>> np.around(mllag.std_err, decimals=4)
array([ 4.8859,  1.0593,  1.7491,  2.7095,  2.3811,  2.3388,  1.6936,
        0.0508,  0.0146,  0.1631,  0.057 ])
>>> np.around(mllag.z_stat, decimals=4)
array([[ 0.8939,  0.3714],
       [ 0.7082,  0.4788],
       [ 3.2083,  0.0013],
       [ 2.6018,  0.0093],
       [ 3.2442,  0.0012],
       [ 2.6181,  0.0088],
       [ 2.7382,  0.0062],
```

(continues on next page)

(continued from previous page)

```
       [-2.178 ,  0.0294],
       [ 4.6487,  0.    ],
       [ 0.4866,  0.6266],
       [ 7.4775,  0.    ]])
>>> mllag.name_y
'PRICE'
>>> mllag.name_x
['CONSTANT', 'NROOM', 'NBATH', 'PATIO', 'FIREPL', 'AC', 'GAR', 'AGE', 'LOTSZ',
↪'SQFT', 'W_PRICE']
>>> mllag.name_w
'baltim_q.gal'
>>> mllag.name_ds
'baltim.dbf'
>>> mllag.title
'MAXIMUM LIKELIHOOD SPATIAL LAG (METHOD = ORD)'
```

Attributes

betas [array] (k+1)x1 array of estimated coefficients (rho first)

rho [float] estimate of spatial autoregressive coefficient

u [array] nx1 array of residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant, excluding the rho)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

method [string] log Jacobian method if 'full': brute force (full matrix computations)

epsilon [float] tolerance criterion used in minimize_scalar function and inverse_product

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (k+1 x k+1), all coefficients

vm1 [array] Variance covariance matrix (k+2 x k+2), includes sig2

sig2 [float] Sigma squared used in computations

logll [float] maximized log-likelihood (including constant terms)

aic [float] Akaike information criterion

schwarz [float] Schwarz criterion

predy_e [array] predicted values from reduced form

e_pred [array] prediction errors using reduced form predicted values

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form))

utu [float] Sum of squared residuals

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

__init__(*self*, *y*, *x*, *w*, *method='full'*, *epsilon=1e-07*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, w[, method, epsilon, ...])	Initialize self.

Attributes

mean_y
sig2n
sig2n_k
std_y
utu
vm

3.3 spreg.ML_Error

class spreg.**ML_Error**(*y*, *x*, *w*, *method='full'*, *epsilon=1e-07*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
ML estimation of the spatial error model with all results and diagnostics; [Ans88]

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

w [Sparse matrix] Spatial weights sparse matrix

method [string] if 'full', brute force calculation (full matrix expressions) if 'ord', Ord eigenvalue method if 'LU', LU sparse matrix decomposition

epsilon [float] tolerance criterion in mimimize_scalar function and inverse_product

spat_diag [boolean] if True, include spatial diagnostics (not implemented yet)

vm [boolean] if True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> np.set_printoptions(suppress=True) #prevent scientific format
>>> db = libpysal.io.open(examples.get_path("south.dbf"),'r')
>>> y_name = "HR90"
>>> y = np.array(db.by_col(y_name))
>>> y.shape = (len(y),1)
>>> x_names = ["RD90","PS90","UE90","DV90"]
>>> x = np.array([db.by_col(var) for var in x_names]).T
>>> ww = libpysal.io.open(libpysal.examples.get_path("south_q.gal"))
>>> w = ww.read()
>>> ww.close()
>>> w_name = "south_q.gal"
>>> w.transform = 'r'
>>> mlerr = ML_Error(y,x,w,name_y=y_name,name_x=x_names,                name_w=w_
↪name,name_ds=ds_name)
>>> np.around(mlerr.betas, decimals=4)
array([[ 6.1492],
       [ 4.4024],
       [ 1.7784],
       [-0.3781],
       [ 0.4858],
       [ 0.2991]])
>>> "{0:.4f}".format(mlerr.lam)
'0.2991'
>>> "{0:.4f}".format(mlerr.mean_y)
'9.5493'
>>> "{0:.4f}".format(mlerr.std_y)
'7.0389'
>>> np.around(np.diag(mlerr.vm), decimals=4)
array([ 1.0648,  0.0555,  0.0454,  0.0061,  0.0148,  0.0014])
>>> np.around(mlerr.sig2, decimals=4)
array([[ 32.4069]])
>>> "{0:.4f}".format(mlerr.logll)
'-4471.4071'
>>> "{0:.4f}".format(mlerr.aic)
'8952.8141'
>>> "{0:.4f}".format(mlerr.schwarz)
'8979.0779'
>>> "{0:.4f}".format(mlerr.pr2)
'0.3058'
>>> "{0:.4f}".format(mlerr.utu)
'48534.9148'
>>> np.around(mlerr.std_err, decimals=4)
array([ 1.0319,  0.2355,  0.2132,  0.0784,  0.1217,  0.0378])
>>> np.around(mlerr.z_stat, decimals=4)
array([[  5.9593,   0.    ],
```

(continues on next page)

(continued from previous page)

```
       [ 18.6902,   0.    ],
       [  8.3422,   0.    ],
       [ -4.8233,   0.    ],
       [  3.9913,   0.0001],
       [  7.9089,   0.    ]])
>>> mlerr.name_y
'HR90'
>>> mlerr.name_x
['CONSTANT', 'RD90', 'PS90', 'UE90', 'DV90', 'lambda']
>>> mlerr.name_w
'south_q.gal'
>>> mlerr.name_ds
'south.dbf'
>>> mlerr.title
'MAXIMUM LIKELIHOOD SPATIAL ERROR (METHOD = FULL)'
```

Attributes

betas [array] (k+1)x1 array of estimated coefficients (rho first)

lam [float] estimate of spatial autoregressive coefficient

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant, excluding lambda)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

method [string] log Jacobian method if 'full': brute force (full matrix computations)

epsilon [float] tolerance criterion used in minimize_scalar function and inverse_product

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

varb [array] Variance covariance matrix (k+1 x k+1) - includes var(lambda)

vm1 [array] variance covariance matrix for lambda, sigma (2 x 2)

sig2 [float] Sigma squared used in computations

logll [float] maximized log-likelihood (including constant terms)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

utu [float] Sum of squared residuals

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

Methods

get_x_lag	

__init__(*self*, *y*, *x*, *w*, *method='full'*, *epsilon=1e-07*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, w[, method, epsilon, ...])	Initialize self.
get_x_lag(self, w, regimes_att)	

Attributes

mean_y
sig2n
sig2n_k
std_y
utu
vm

3.4 spreg.GM_Lag

class spreg.**GM_Lag**(*y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *robust=None*, *gwk=None*, *sig2n_k=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)
Spatial two stage least squares (S2SLS) with results and diagnostics; Anselin (1988) [Ans88]

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x); cannot be used in combination with h

w [pysal W object] Spatial weights object

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

robust [string] If 'white', then a White consistent estimator of the variance-covariance matrix is given. If 'hac', then a HAC consistent estimator of the variance-covariance matrix is given. Default set to None.

gwk [pysal W object] Kernel spatial weights needed for HAC estimation. Note: matrix must have ones along the main diagonal.

sig2n_k [boolean] If True, then use n-k to estimate sigma^2. If False, use n.

spat_diag [boolean] If True, then compute Anselin-Kelejian test

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis. Since we will need some tests for our model, we also import the diagnostics module.

```
>>> import numpy as np
>>> import libpysal
>>> import spreg.diagnostics as D
```

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("columbus.dbf"),'r')
```

Extract the HOVAL column (home value) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("HOVAL"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) and CRIME (crime rates) vectors from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in, but this can be overridden by passing constant=False.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("CRIME"))
>>> X = np.array(X).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

This class runs a lag model, which means that includes the spatial lag of the dependent variable on the right-hand side of the equation. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional. The default most basic model to be run would be:

```
>>> reg=GM_Lag(y, X, w=w, w_lags=2, name_x=['inc', 'crime'], name_y='hoval', name_
↪ds='columbus')
>>> reg.betas
array([[ 45.30170561],
       [  0.62088862],
       [ -0.48072345],
       [  0.02836221]])
```

Once the model is run, we can obtain the standard error of the coefficient estimates by calling the diagnostics module:

```
>>> D.se_betas(reg)
array([ 17.91278862,   0.52486082,   0.1822815 ,   0.31740089])
```

But we can also run models that incorporates corrected standard errors following the White procedure. For that, we will have to include the optional parameter `robust='white'`:

```
>>> reg=GM_Lag(y, X, w=w, w_lags=2, robust='white', name_x=['inc', 'crime'], name_
↪y='hoval', name_ds='columbus')
>>> reg.betas
array([[ 45.30170561],
       [  0.62088862],
       [ -0.48072345],
       [  0.02836221]])
```

And we can access the standard errors from the model object:

```
>>> reg.std_err
array([ 20.47077481,   0.50613931,   0.20138425,   0.38028295])
```

The class is flexible enough to accomodate a spatial lag model that, besides the spatial lag of the dependent variable, includes other non-spatial endogenous regressors. As an example, we will assume that CRIME is actually endogenous and we decide to instrument for it with DISCBD (distance to the CBD). We reload the X including INC only and define CRIME as endogenous and DISCBD as instrument:

```
>>> X = np.array(db.by_col("INC"))
>>> X = np.reshape(X, (49,1))
>>> yd = np.array(db.by_col("CRIME"))
>>> yd = np.reshape(yd, (49,1))
>>> q = np.array(db.by_col("DISCBD"))
>>> q = np.reshape(q, (49,1))
```

And we can run the model again:

```
>>> reg=GM_Lag(y, X, w=w, yend=yd, q=q, w_lags=2, name_x=['inc'], name_y='hoval',␣
↪name_yend=['crime'], name_q=['discbd'], name_ds='columbus')
>>> reg.betas
array([[ 100.79359082],
       [  -0.50215501],
       [  -1.14881711],
       [  -0.38235022]])
```

Once the model is run, we can obtain the standard error of the coefficient estimates by calling the diagnostics module:

```
>>> D.se_betas(reg)
array([ 53.0829123 ,   1.02511494,   0.57589064,   0.59891744])
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

kstar [integer] Number of endogenous variables.

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments

z [array] nxk array of variables (combination of x and yend)

h [array] nxl array of instruments (combination of x and q)

robust [string] Adjustment for robust standard errors

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form))

utu [float] Sum of squared residuals

sig2 [float] Sigma squared used in computations

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

ak_test [tuple] Anselin-Kelejian test; tuple contains the pair (statistic, p-value)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

sig2n [float] Sigma squared (computed with n in the denominator)

sig2n_k [float] Sigma squared (computed with n-k in the denominator)

hth [float] $H'H$

hthi [float] $(H'H)^{-1}$

varb [array] $(Z'H(H'H)^{-1}H'Z)^{-1}$

zthhthi [array] $Z'H(H'H)^{-1}$

pfora1a2 [array] n(zthhthi)'varb

__init__(*self*, *y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *robust=None*, *gwk=None*, *sig2n_k=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x[, yend, q, w, w_lags, ...])	Initialize self.

Attributes

mean_y
pfora1a2
sig2n
sig2n_k
std_y
utu
vm

3.5 spreg.GM_Error

class spreg.**GM_Error**(*y*, *x*, *w*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)

GMM method for a spatial error model, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

w [pysal W object] Spatial weights object (always needed)

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> dbf = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array([dbf.by_col('HOVAL')]).T
```

Extract CRIME (crime) and INC (income) vectors from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent

variables (not including a constant). By default this class adds a vector of ones to the independent variables passed in.

```
>>> names_to_extract = ['INC', 'CRIME']
>>> x = np.array([dbf.by_col(name) for name in names_to_extract]).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will use `columbus.gal`, which contains contiguity relationships between the observations in the Columbus dataset we are using throughout this example. Note that, in order to read the file, not only to open it, we need to append '.read()' at the end of the command.

```
>>> w = libpysal.io.open(libpysal.examples.get_path("columbus.gal"), 'r').read()
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, his allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform='r'
```

We are all set with the preliminars, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> model = GM_Error(y, x, w=w, name_y='hoval', name_x=['income', 'crime'], name_
↪ds='columbus')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. Note that because we are running the classical GMM error model from 1998/99, the spatial parameter is obtained as a point estimate, so although you get a value for it (there are for coefficients under model.betas), you cannot perform inference on it (there are only three values in model.se_betas).

```
>>> print model.name_x
['CONSTANT', 'income', 'crime', 'lambda']
>>> np.around(model.betas, decimals=4)
array([[ 47.6946],
       [  0.7105],
       [ -0.5505],
       [  0.3257]])
>>> np.around(model.std_err, decimals=4)
array([ 12.412 ,   0.5044,   0.1785])
>>> np.around(model.z_stat, decimals=6)
array([[  3.84261100e+00,   1.22000000e-04],
       [  1.40839200e+00,   1.59015000e-01],
       [ -3.08424700e+00,   2.04100000e-03]])
>>> round(model.sig2,4)
198.5596
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

vm [array] Variance covariance matrix (kxk)

sig2 [float] Sigma squared used in computations

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

__init__(*self*, *y*, *x*, *w*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, w[, vm, name_y, ...])	Initialize self.

Attributes

mean_y
std_y

3.6 spreg.GM_Error_Het

class spreg.**GM_Error_Het**(*y*, *x*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
GMM method for a spatial error model with heteroskedasticity, with results and diagnostics; based on [ADKP10], following [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

w [pysal W object] Spatial weights object

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

step1c [boolean] If True, then include Step 1c from [ADKP10].

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("HOVAL"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) and CRIME (crime) vectors from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this class adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("CRIME"))
>>> X = np.array(X).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, his allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Error_Het(y, X, w=w, step1c=True, name_y='home value', name_x=[
↪'income', 'crime'], name_ds='columbus')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that explicitly accounts for heteroskedasticity and that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter.

```
>>> print reg.name_x
['CONSTANT', 'income', 'crime', 'lambda']
```

Hence, we find the same number of betas as of standard errors, which we calculate taking the square root of the diagonal of the variance-covariance matrix:

```
>>> print np.around(np.hstack((reg.betas,np.sqrt(reg.vm.diagonal()).reshape(4,
↪1))),4)
[[ 47.9963  11.479 ]
 [  0.7105   0.3681]
 [ -0.5588   0.1616]
 [  0.4118   0.168 ]]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10].

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10].

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

vm [array] Variance covariance matrix (kxk)

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

xtx [float] $X'X$

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

__init__(*self*, *y*, *x*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, w[, max_iter, epsilon, ...])	Initialize self.

Attributes

mean_y
std_y

3.7 spreg.GM_Error_Hom

class spreg.**GM_Error_Hom**(*y*, *x*, *w*, *max_iter=1*, *epsilon=1e-05*, *A1='hom_sc'*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
GMM method for a spatial error model with homoskedasticity, with results and diagnostics; based on Drukker et al. (2013) [DEP13], following Anselin (2011) [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

w [pysal W object] Spatial weights object

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

A1 [string] If A1='het', then the matrix A1 is defined as in Arraiz et al. If A1='hom', then as in [Ans11]. If A1='hom_sc' (default), then as in [DEP13] and [DPR13].

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("HOVAL"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) and CRIME (crime) vectors from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this class adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("CRIME"))
>>> X = np.array(X).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, his allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminars, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Error_Hom(y, X, w=w, A1='hom_sc', name_y='home value', name_x=[
↪'income', 'crime'], name_ds='columbus')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that assumes homoskedasticity but that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. This is why you obtain as many coefficient estimates as standard errors, which you calculate taking the square root of the diagonal of the variance-covariance matrix of the parameters:

```
>>> print np.around(np.hstack((reg.betas,np.sqrt(reg.vm.diagonal()).reshape(4,
↪1))),4)
[[ 47.9479  12.3021]
 [  0.7063   0.4967]
 [ -0.556    0.179 ]
 [  0.4129   0.1835]]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10].

iteration [integer] Number of iterations of steps 2a and 2b from Arraiz et al.

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

vm [array] Variance covariance matrix (kxk)

sig2 [float] Sigma squared used in computations

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

xtx [float] $X'X$

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

__init__(*self*, *y*, *x*, *w*, *max_iter=1*, *epsilon=1e-05*, *A1='hom_sc'*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, w[, max_iter, epsilon, ...])	Initialize self.

Attributes

mean_y
std_y

3.8 spreg.GM_Combo

class spreg.**GM_Combo**(*y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
GMM method for a spatial lag and error model with endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object (always needed)

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("columbus.dbf"),'r')
```

Extract the CRIME column (crime rates) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X = np.array(X).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

The Combo class runs an SARAR model, that is a spatial lag+error model. In this case we will run a simple version of that, where we have the spatial effects as well as exogenous variables. Since it is a spatial model, we have to pass in the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Combo(y, X, w=w, name_y='crime', name_x=['income'], name_ds='columbus
↪')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. Note that because we are running the classical GMM error

model from 1998/99, the spatial parameter is obtained as a point estimate, so although you get a value for it (there are for coefficients under model.betas), you cannot perform inference on it (there are only three values in model.se_betas). Also, this regression uses a two stage least squares estimation method that accounts for the endogeneity created by the spatial lag of the dependent variable. We can check the betas:

```
>>> print reg.name_z
['CONSTANT', 'income', 'W_crime', 'lambda']
>>> print np.around(np.hstack((reg.betas[:-1],np.sqrt(reg.vm.diagonal()).
↪reshape(3,1))),3)
[[ 39.059  11.86 ]
 [ -1.404   0.391]
 [  0.467   0.2  ]]
```

And lambda:

```
>>> print 'lambda: ', np.around(reg.betas[-1], 3)
lambda:  [-0.048]
```

This class also allows the user to run a spatial lag+error model with the extra feature of including non-spatial endogenous regressors. This means that, in addition to the spatial lag and error, we consider some of the variables on the right-hand side of the equation as endogenous and we instrument for this. As an example, we will include HOVAL (home value) as endogenous and will instrument with DISCBD (distance to the CSB). We first need to read in the variables:

```
>>> yd = []
>>> yd.append(db.by_col("HOVAL"))
>>> yd = np.array(yd).T
>>> q = []
>>> q.append(db.by_col("DISCBD"))
>>> q = np.array(q).T
```

And then we can run and explore the model analogously to the previous combo:

```
>>> reg = GM_Combo(y, X, yd, q, w=w, name_x=['inc'], name_y='crime', name_yend=[
↪'hoval'], name_q=['discbd'], name_ds='columbus')
>>> print reg.name_z
['CONSTANT', 'inc', 'hoval', 'W_crime', 'lambda']
>>> names = np.array(reg.name_z).reshape(5,1)
>>> print np.hstack((names[0:4,:], np.around(np.hstack((reg.betas[:-1], np.
↪sqrt(reg.vm.diagonal()).reshape(4,1))),4)))
[['CONSTANT' '50.0944' '14.3593']
 ['inc' '-0.2552' '0.5667']
 ['hoval' '-0.6885' '0.3029']
 ['W_crime' '0.4375' '0.2314']]
```

```
>>> print 'lambda: ', np.around(reg.betas[-1], 3)
lambda:  [ 0.254]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

z [array] nxk array of variables (combination of x and yend)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form))

sig2 [float] Sigma squared used in computations (based on filtered residuals)

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

__init__(*self*, *y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x[, yend, q, w, w_lags, ...])	Initialize self.

Attributes

mean_y
std_y

3.9 spreg.GM_Combo_Het

class spreg.**GM_Combo_Het**(*y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *inv_method='power_exp'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)

GMM method for a spatial lag and error model with heteroskedasticity and endogenous variables, with results and diagnostics; based on [ADKP10], following [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object (always needed)

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

step1c [boolean] If True, then include Step 1c from [ADKP10].

inv_method [string] If "power_exp", then compute inverse using the power expansion. If "true_inv", then compute the true inverse. Note that true_inv will fail for large n.

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("HOVAL"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this class adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X = np.array(X).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, his allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

The Combo class runs an SARAR model, that is a spatial lag+error model. In this case we will run a simple version of that, where we have the spatial effects as well as exogenous variables. Since it is a spatial model, we have to pass in the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Combo_Het(y, X, w=w, step1c=True, name_y='hoval', name_x=['income'],␣
↪name_ds='columbus')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that explicitly accounts for heteroskedasticity and that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. Hence, we find the same number of betas as of standard errors, which we calculate taking the square root of the diagonal of the variance-covariance matrix:

```
>>> print reg.name_z
['CONSTANT', 'income', 'W_hoval', 'lambda']
>>> print np.around(np.hstack((reg.betas,np.sqrt(reg.vm.diagonal()).reshape(4,
↪1))),4)
[[  9.9753  14.1435]
 [  1.5742   0.374 ]
 [  0.1535   0.3978]
 [  0.2103   0.3924]]
```

This class also allows the user to run a spatial lag+error model with the extra feature of including non-spatial endogenous regressors. This means that, in addition to the spatial lag and error, we consider some of the variables on the right-hand side of the equation as endogenous and we instrument for this. As an example, we will include CRIME (crime rates) as endogenous and will instrument with DISCBD (distance to the CSB). We first need to read in the variables:

```
>>> yd = []
>>> yd.append(db.by_col("CRIME"))
>>> yd = np.array(yd).T
>>> q = []
>>> q.append(db.by_col("DISCBD"))
>>> q = np.array(q).T
```

And then we can run and explore the model analogously to the previous combo:

```
>>> reg = GM_Combo_Het(y, X, yd, q, w=w, step1c=True, name_x=['inc'], name_y=
↪'hoval', name_yend=['crime'], name_q=['discbd'], name_ds='columbus')
>>> print reg.name_z
['CONSTANT', 'inc', 'crime', 'W_hoval', 'lambda']
>>> print np.round(reg.betas,4)
[[ 113.9129]
 [  -0.3482]
 [  -1.3566]
 [  -0.5766]
 [   0.6561]]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments

z [array] nxk array of variables (combination of x and yend)

h [array] nxl array of instruments (combination of x and q)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10].

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10].

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form))

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

hth [float] $H'H$

__init__(*self*, *y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *inv_method='power_exp'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x[, yend, q, w, w_lags, ...])	Initialize self.

Attributes

mean_y
std_y

3.10 spreg.GM_Combo_Hom

class spreg.**GM_Combo_Hom**(*y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *max_iter=1*, *epsilon=1e-05*, *A1='hom_sc'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)

GMM method for a spatial lag and error model with homoskedasticity and endogenous variables, with results and diagnostics; based on Drukker et al. (2013) [DEP13], following Anselin (2011) [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object (always necessary)

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

A1 [string] If A1='het', then the matrix A1 is defined as in [ADKP10]. If A1='hom', then as in [Ans11]. If A1='hom_sc' (default), then as in [DEP13] and [DPR13].

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("HOVAL"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this class adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X = np.array(X).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, his allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

Example only with spatial lag

The Combo class runs an SARAR model, that is a spatial lag+error model. In this case we will run a simple version of that, where we have the spatial effects as well as exogenous variables. Since it is a spatial model, we have to pass in the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Combo_Hom(y, X, w=w, A1='hom_sc', name_x=['inc'],            name_y=
↪'hoval', name_yend=['crime'], name_q=['discbd'],            name_ds='columbus')
>>> print np.around(np.hstack((reg.betas,np.sqrt(reg.vm.diagonal()).reshape(4,
↪1))),4)
[[ 10.1254  15.2871]
 [  1.5683   0.4407]
 [  0.1513   0.4048]
 [  0.2103   0.4226]]
```

This class also allows the user to run a spatial lag+error model with the extra feature of including non-spatial endogenous regressors. This means that, in addition to the spatial lag and error, we consider some of the variables on the right-hand side of the equation as endogenous and we instrument for this. As an example, we will include CRIME (crime rates) as endogenous and will instrument with DISCBD (distance to the CSB). We first need to read in the variables:

```
>>> yd = []
>>> yd.append(db.by_col("CRIME"))
>>> yd = np.array(yd).T
>>> q = []
```

(continues on next page)

(continued from previous page)

```
>>> q.append(db.by_col("DISCBD"))
>>> q = np.array(q).T
```

And then we can run and explore the model analogously to the previous combo:

```
>>> reg = GM_Combo_Hom(y, X, yd, q, w=w, A1='hom_sc',            name_ds=
↪'columbus')
>>> betas = np.array([['CONSTANT'],['inc'],['crime'],['W_hoval'],['lambda']])
>>> print np.hstack((betas, np.around(np.hstack((reg.betas, np.sqrt(reg.vm.
↪diagonal()).reshape(5,1))),5)))
[['CONSTANT' '111.7705' '67.75191']
 ['inc' '-0.30974' '1.16656']
 ['crime' '-1.36043' '0.6841']
 ['W_hoval' '-0.52908' '0.84428']
 ['lambda' '0.60116' '0.18605']]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments

z [array] nxk array of variables (combination of x and yend)

h [array] nxl array of instruments (combination of x and q)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10].

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10].

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form))

sig2 [float] Sigma squared used in computations (based on filtered residuals)

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

hth [float] $H'H$

__init__(*self*, *y*, *x*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *max_iter=1*, *epsilon=1e-05*, *A1='hom_sc'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x[, yend, q, w, w_lags, ...])	Initialize self.

Attributes

mean_y
std_y

3.11 spreg.GM_Endog_Error

class spreg.**GM_Endog_Error**(*y*, *x*, *yend*, *q*, *w*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
GMM method for a spatial error model with endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object (always needed)

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> dbf = libpysal.io.open(libpysal.examples.get_path("columbus.dbf"),'r')
```

Extract the CRIME column (crime rates) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array([dbf.by_col('CRIME')]).T
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x = np.array([dbf.by_col('INC')]).T
```

In this case we consider HOVAL (home value) is an endogenous regressor. We tell the model that this is so by passing it in a different parameter from the exogenous variables (x).

```
>>> yend = np.array([dbf.by_col('HOVAL')]).T
```

Because we have endogenous variables, to obtain a correct estimate of the model, we need to instrument for HOVAL. We use DISCBD (distance to the CBD) for this and hence put it in the instruments parameter, 'q'.

```
>>> q = np.array([dbf.by_col('DISCBD')]).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will use `columbus.gal`, which contains contiguity

relationships between the observations in the Columbus dataset we are using throughout this example. Note that, in order to read the file, not only to open it, we need to append '.read()' at the end of the command.

```
>>> w = libpysal.io.open(libpysal.examples.get_path("columbus.gal"), 'r').read()
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform='r'
```

We are all set with the preliminars, we are good to run the model. In this case, we will need the variables (exogenous and endogenous), the instruments and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> from spreg import GM_Endog_Error
>>> model = GM_Endog_Error(y, x, yend, q, w=w, name_x=['inc'], name_y='crime',␣
↪name_yend=['hoval'], name_q=['discbd'], name_ds='columbus')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. Note that because we are running the classical GMM error model from 1998/99, the spatial parameter is obtained as a point estimate, so although you get a value for it (there are for coefficients under model.betas), you cannot perform inference on it (there are only three values in model.se_betas). Also, this regression uses a two stage least squares estimation method that accounts for the endogeneity created by the endogenous variables included.

```
>>> print model.name_z
['CONSTANT', 'inc', 'hoval', 'lambda']
>>> np.around(model.betas, decimals=4)
array([[ 82.573 ],
       [  0.581 ],
       [ -1.4481],
       [  0.3499]])
>>> np.around(model.std_err, decimals=4)
array([ 16.1381,   1.3545,   0.7862])
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

z [array] nxk array of variables (combination of x and yend)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

sig2 [float] Sigma squared used in computations

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

__init__(*self*, *y*, *x*, *yend*, *q*, *w*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, yend, q, w[, vm, ...])	Initialize self.

Attributes

mean_y
std_y

3.12 spreg.GM_Endog_Error_Het

class spreg.**GM_Endog_Error_Het**(*y*, *x*, *yend*, *q*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *inv_method='power_exp'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
GMM method for a spatial error model with heteroskedasticity and endogenous variables, with results and diagnostics; based on [ADKP10], following [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

step1c [boolean] If True, then include Step 1c from [ADKP10].

inv_method [string] If "power_exp", then compute inverse using the power expansion. If "true_inv", then compute the true inverse. Note that true_inv will fail for large n.

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("HOVAL"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this class adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X = np.array(X).T
```

In this case we consider CRIME (crime rates) is an endogenous regressor. We tell the model that this is so by passing it in a different parameter from the exogenous variables (x).

```
>>> yd = []
>>> yd.append(db.by_col("CRIME"))
>>> yd = np.array(yd).T
```

Because we have endogenous variables, to obtain a correct estimate of the model, we need to instrument for CRIME. We use DISCBD (distance to the CBD) for this and hence put it in the instruments parameter, 'q'.

```
>>> q = []
>>> q.append(db.by_col("DISCBD"))
>>> q = np.array(q).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, his allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables (exogenous and endogenous), the instruments and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Endog_Error_Het(y, X, yd, q, w=w, step1c=True, name_x=['inc'], name_
↪y='hoval', name_yend=['crime'], name_q=['discbd'], name_ds='columbus')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that explicitly accounts for heteroskedasticity and that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. Hence, we find the same number of betas as of standard errors, which we calculate taking the square root of the diagonal of the variance-covariance matrix:

```
>>> print reg.name_z
['CONSTANT', 'inc', 'crime', 'lambda']
>>> print np.around(np.hstack((reg.betas,np.sqrt(reg.vm.diagonal()).reshape(4,
↪1))),4)
[[ 55.3971  28.8901]
 [  0.4656   0.7731]
 [ -0.6704   0.468 ]
 [  0.4114   0.1777]]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments

z [array] nxk array of variables (combination of x and yend)

h [array] nxl array of instruments (combination of x and q)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10].

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10].

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

hth [float] $H'H$

__init__(*self*, *y*, *x*, *yend*, *q*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *inv_method='power_exp'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)

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

Methods

`__init__`(self, y, x, yend, q, w[, max_iter, ...])	Initialize self.

Attributes

mean_y
std_y

3.13 spreg.GM_Endog_Error_Hom

class spreg.**GM_Endog_Error_Hom**(*y*, *x*, *yend*, *q*, *w*, *max_iter=1*, *epsilon=1e-05*, *A1='hom_sc'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)

GMM method for a spatial error model with homoskedasticity and endogenous variables, with results and diagnostics; based on Drukker et al. (2013) [DEP13], following Anselin (2011) [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

A1 [string] If A1='het', then the matrix A1 is defined as in [ADKP10]. If A1='hom', then as in [Ans11]. If A1='hom_sc' (default), then as in [DEP13] and [DPR13].

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Extract the HOVAL column (home values) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("HOVAL"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this class adds a vector of ones to the independent variables passed in.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X = np.array(X).T
```

In this case we consider CRIME (crime rates) is an endogenous regressor. We tell the model that this is so by passing it in a different parameter from the exogenous variables (x).

```
>>> yd = []
>>> yd.append(db.by_col("CRIME"))
>>> yd = np.array(yd).T
```

Because we have endogenous variables, to obtain a correct estimate of the model, we need to instrument for CRIME. We use DISCBD (distance to the CBD) for this and hence put it in the instruments parameter, 'q'.

```
>>> q = []
>>> q.append(db.by_col("DISCBD"))
>>> q = np.array(q).T
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, his allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminars, we are good to run the model. In this case, we will need the variables (exogenous and endogenous), the instruments and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Endog_Error_Hom(y, X, yd, q, w=w, A1='hom_sc', name_x=['inc'], name_
↪y='hoval', name_yend=['crime'], name_q=['discbd'], name_ds='columbus')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that assumes homoskedasticity but that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. Hence, we find the same number of betas as of standard errors, which we calculate taking the square root of the diagonal of the variance-covariance matrix:

```
>>> print reg.name_z
['CONSTANT', 'inc', 'crime', 'lambda']
>>> print np.around(np.hstack((reg.betas,np.sqrt(reg.vm.diagonal()).reshape(4,
↪1))),4)
[[ 55.3658  23.496 ]
 [  0.4643   0.7382]
 [ -0.669    0.3943]
 [  0.4321   0.1927]]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments

z [array] nxk array of variables (combination of x and yend)

h [array] nxl array of instruments (combination of x and q)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10].

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10].

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

sig2 [float] Sigma squared used in computations

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

hth [float] $H'H$

__init__(*self*, *y*, *x*, *yend*, *q*, *w*, *max_iter=1*, *epsilon=1e-05*, *A1='hom_sc'*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, yend, q, w[, max_iter, ...])	Initialize self.

Attributes

mean_y
std_y

3.14 spreg.TSLS

class spreg.**TSLS**(*y*, *x*, *yend*, *q*, *w=None*, *robust=None*, *gwk=None*, *sig2n_k=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)
Two stage least squares with results and diagnostics.

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object (required if running spatial diagnostics)

robust [string] If 'white', then a White consistent estimator of the variance-covariance matrix is given. If 'hac', then a HAC consistent estimator of the variance-covariance matrix is given.

Default set to None.

gwk [pysal W object] Kernel spatial weights needed for HAC estimation. Note: matrix must have ones along the main diagonal.

sig2n_k [boolean] If True, then use n-k to estimate sigma^2. If False, use n.

spat_diag [boolean] If True, then compute Anselin-Kelejian test (requires w)

vm [boolean] If True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("columbus.dbf"),'r')
```

Extract the CRIME column (crime rates) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in, but this can be overridden by passing constant=False.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X = np.array(X).T
```

In this case we consider HOVAL (home value) is an endogenous regressor. We tell the model that this is so by passing it in a different parameter from the exogenous variables (x).

```
>>> yd = []
>>> yd.append(db.by_col("HOVAL"))
>>> yd = np.array(yd).T
```

Because we have endogenous variables, to obtain a correct estimate of the model, we need to instrument for HOVAL. We use DISCBD (distance to the CBD) for this and hence put it in the instruments parameter, 'q'.

```
>>> q = []
>>> q.append(db.by_col("DISCBD"))
>>> q = np.array(q).T
```

We are all set with the preliminars, we are good to run the model. In this case, we will need the variables (exogenous and endogenous) and the instruments. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = TSLS(y, X, yd, q, name_x=['inc'], name_y='crime', name_yend=['hoval'],␣
↪name_q=['discbd'], name_ds='columbus')
>>> print reg.betas
[[ 88.46579584]
 [  0.5200379 ]
 [ -1.58216593]]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant)

kstar [integer] Number of endogenous variables.

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments

z [array] nxk array of variables (combination of x and yend)

h [array] nxl array of instruments (combination of x and q)

robust [string] Adjustment for robust standard errors

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred)

utu [float] Sum of squared residuals

sig2 [float] Sigma squared used in computations

std_err [array] 1xk array of standard errors of the betas

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float

ak_test [tuple] Anselin-Kelejian test; tuple contains the pair (statistic, p-value)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

title [string] Name of the regression method used

sig2n [float] Sigma squared (computed with n in the denominator)

sig2n_k [float] Sigma squared (computed with n-k in the denominator)

hth [float] $H'H$

hthi [float] $(H'H)^{-1}$

varb [array] $(Z'H(H'H)^{-1}H'Z)^{-1}$

zthhthi [array] $Z'H(H'H)^{-1}$

pfora1a2 [array] $n(zthhthi)'varb$

__init__(*self*, *y*, *x*, *yend*, *q*, *w=None*, *robust=None*, *gwk=None*, *sig2n_k=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, yend, q[, w, robust, ...])	Initialize self.

Attributes

mean_y
pfora1a2
sig2n
sig2n_k
std_y
utu

Continued on next page

Table 29 – continued from previous page

vm

3.15 spreg.ThreeSLS

class spreg.**ThreeSLS**(*bigy*, *bigX*, *bigyend*, *bigq*, *regimes=None*, *nonspat_diag=True*, *name_bigy=None*, *name_bigX=None*, *name_bigyend=None*, *name_bigq=None*, *name_ds=None*, *name_regimes=None*)

User class for 3SLS estimation

Parameters

bigy [dictionary] with vector for dependent variable by equation

bigX [dictionary] with matrix of explanatory variables by equation (note, already includes constant term)

bigyend [dictionary] with matrix of endogenous variables by equation

bigq [dictionary] with matrix of instruments by equation

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

nonspat_diag: boolean flag for non-spatial diagnostics, default = True.

name_bigy [dictionary] with name of dependent variable for each equation. default = None, but should be specified. is done when sur_stackxy is used

name_bigX [dictionary] with names of explanatory variables for each equation. default = None, but should be specified. is done when sur_stackxy is used

name_bigyend [dictionary] with names of endogenous variables for each equation. default = None, but should be specified. is done when sur_stackZ is used

name_bigq [dictionary] with names of instrumental variables for each equation. default = None, but should be specified. is done when sur_stackZ is used.

name_ds [string] name for the data set.

name_regimes [string] name of regime variable for use in the output.

Examples

First import libpysal to load the spatial analysis tools.

```
>>> import libpysal
```

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

The specification of the model to be estimated can be provided as lists. Each equation should be listed separately. In this example, equation 1 has HR80 as dependent variable, PS80 and UE80 as exogenous regressors, RD80 as endogenous regressor and FP79 as additional instrument. For equation 2, HR90 is the dependent variable, PS90 and UE90 the exogenous regressors, RD90 as endogenous regressor and FP99 as additional instrument

```
>>> y_var = ['HR80','HR90']
>>> x_var = [['PS80','UE80'],['PS90','UE90']]
>>> yend_var = [['RD80'],['RD90']]
>>> q_var = [['FP79'],['FP89']]
```

The SUR method requires data to be provided as dictionaries. PySAL provides two tools to create these dictionaries from the list of variables: sur_dictxy and sur_dictZ. The tool sur_dictxy can be used to create the dictionaries for Y and X, and sur_dictZ for endogenous variables (yend) and additional instruments (q).

```
>>> bigy,bigX,bigyvars,bigXvars = pysal.spreg.sur_utils.sur_dictxy(db,y_var,x_var)
>>> bigyend,bigyendvars = pysal.spreg.sur_utils.sur_dictZ(db,yend_var)
>>> bigq,bigqvars = pysal.spreg.sur_utils.sur_dictZ(db,q_var)
```

We can now run the regression and then have a summary of the output by typing: print(reg.summary)

Alternatively, we can just check the betas and standard errors, asymptotic t and p-value of the parameters:

```
>>> reg = ThreeSLS(bigy,bigX,bigyend,bigq,name_bigy=bigyvars,name_bigX=bigXvars,
↪name_bigyend=bigyendvars,name_bigq=bigqvars,name_ds="NAT")
>>> reg.b3SLS
{0: array([[ 6.92426353],
       [ 1.42921826],
       [ 0.00049435],
       [ 3.5829275 ]]), 1: array([[ 7.62385875],
       [ 1.65031181],
       [-0.21682974],
       [ 3.91250428]])}
```

```
>>> reg.tsls_inf
{0: array([[  0.23220853,  29.81916157,   0.        ],
       [  0.10373417,  13.77770036,   0.        ],
       [  0.03086193,   0.01601807,   0.98721998],
       [  0.11131999,  32.18584124,   0.        ]]), 1: array([[  0.28739415,  26.
↪52753638,   0.        ],
       [  0.09597031,  17.19606554,   0.        ],
       [  0.04089547,  -5.30204786,   0.00000011],
       [  0.13586789,  28.79638723,   0.        ]])}
```

Attributes

bigy [dictionary] with y values

bigZ [dictionary] with matrix of exogenous and endogenous variables for each equation

bigZHZH [dictionary] with matrix of cross products Zhat_r'Zhat_s

bigZHy [dictionary] with matrix of cross products Zhat_r'y_end_s

n_eq [int] number of equations

n [int] number of observations in each cross-section

bigK [array] vector with number of explanatory variables (including constant, exogenous and endogenous) for each equation

b2SLS [dictionary] with 2SLS regression coefficients for each equation

tslsE [array] N x n_eq array with OLS residuals for each equation

b3SLS [dictionary] with 3SLS regression coefficients for each equation

varb [array] variance-covariance matrix

sig [array] Sigma matrix of inter-equation error covariances

bigE [array] n by n_eq array of residuals

corr [array] inter-equation 3SLS error correlation matrix

tsls_inf [dictionary] with standard error, asymptotic t and p-value, one for each equation

surchow [array] list with tuples for Chow test on regression coefficients each tuple contains test value, degrees of freedom, p-value

name_ds [string] name for the data set

name_bigy [dictionary] with name of dependent variable for each equation

name_bigX [dictionary] with names of explanatory variables for each equation

name_bigyend [dictionary] with names of endogenous variables for each equation

name_bigq [dictionary] with names of instrumental variables for each equations

name_regimes [string] name of regime variable for use in the output

__init__(*self*, *bigy*, *bigX*, *bigyend*, *bigq*, *regimes=None*, *nonspat_diag=True*, *name_bigy=None*, *name_bigX=None*, *name_bigyend=None*, *name_bigq=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, bigy, bigX, bigyend, bigq[, ...])	Initialize self.

3.16 Regimes Models

Regimes models are variants of spatial regression models which allow for structural instability in parameters. That means that these models allow different coefficient values in distinct subsets of the data.

spreg.OLS_Regimes(y, x, regimes[, w, ...])	Ordinary least squares with results and diagnostics.
spreg.ML_Lag_Regimes(y, x, regimes[, w, ...])	ML estimation of the spatial lag model with regimes (note no consistency checks, diagnostics or constants added) [Ans88].
spreg.ML_Error_Regimes(y, x, regimes[, w, ...])	ML estimation of the spatial error model with regimes (note no consistency checks, diagnostics or constants added); Anselin (1988) [Ans88]
spreg.GM_Lag_Regimes(y, x, regimes[, yend, ...])	Spatial two stage least squares (S2SLS) with regimes; [Ans88]
spreg.GM_Error_Regimes(y, x, regimes, w[, ...])	GMM method for a spatial error model with regimes, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].
spreg.GM_Error_Het_Regimes(y, x, regimes, w)	GMM method for a spatial error model with heteroskedasticity and regimes; based on Arraiz et al [ADKP10], following Anselin [Ans11].

Continued on next page

Table 31 – continued from previous page

spreg.GM_Error_Hom_Regimes(y, x, regimes, w)	GMM method for a spatial error model with homoskedasticity, with regimes, results and diagnostics; based on Drukker et al.
spreg.GM_Combo_Regimes(y, x, regimes[, ...])	GMM method for a spatial lag and error model with regimes and endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].
spreg.GM_Combo_Hom_Regimes(y, x, regimes[, ...])	GMM method for a spatial lag and error model with homoskedasticity, regimes and endogenous variables, with results and diagnostics; based on Drukker et al.
spreg.GM_Combo_Het_Regimes(y, x, regimes[, ...])	GMM method for a spatial lag and error model with heteroskedasticity, regimes and endogenous variables, with results and diagnostics; based on Arraiz et al [ADKP10], following Anselin [Ans11].
spreg.GM_Endog_Error_Regimes(y, x, yend, q, ...)	GMM method for a spatial error model with regimes and endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].
spreg.GM_Endog_Error_Hom_Regimes(y, x, yend, ...)	GMM method for a spatial error model with homoskedasticity, regimes and endogenous variables.
spreg.GM_Endog_Error_Het_Regimes(y, x, yend, ...)	GMM method for a spatial error model with heteroskedasticity, regimes and endogenous variables, with results and diagnostics; based on Arraiz et al [ADKP10], following Anselin [Ans11].

3.16.1 spreg.OLS_Regimes

class spreg.**OLS_Regimes**(*y*, *x*, *regimes*, *w=None*, *robust=None*, *gwk=None*, *sig2n_k=True*, *nonspat_diag=True*, *spat_diag=False*, *moran=False*, *white_test=False*, *vm=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=True*, *cores=False*, *name_y=None*, *name_x=None*, *name_regimes=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)

Ordinary least squares with results and diagnostics.

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object (required if running spatial diagnostics)

robust [string] If 'white', then a White consistent estimator of the variance-covariance matrix is given. If 'hac', then a HAC consistent estimator of the variance-covariance matrix is given. Default set to None.

gwk [pysal W object] Kernel spatial weights needed for HAC estimation. Note: matrix must have ones along the main diagonal.

sig2n_k [boolean] If True, then use n-k to estimate sigma^2. If False, use n.

nonspat_diag [boolean] If True, then compute non-spatial diagnostics on the regression.

spat_diag [boolean] If True, then compute Lagrange multiplier tests (requires w). Note: see moran for further tests.

moran [boolean] If True, compute Moran's I on the residuals. Note: requires spat_diag=True.

white_test [boolean] If True, compute White's specification robust test. (requires non-spat_diag=True)

vm [boolean] If True, include variance-covariance matrix in summary results

constant_regi: string, optional Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes
- 'many': a vector of ones is appended to x and considered different per regime (default)

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = db.by_col(y_var)
>>> y = np.array(y).reshape(len(y), 1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = ['Var1','Var2','...] Note that PySAL requires this to be an nxj numpy array, where j is the number of

independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

We can now run the regression and then have a summary of the output by typing: olsr.summary Alternatively, we can just check the betas and standard errors of the parameters:

```
>>> olsr = OLS_Regimes(y, x, regimes, nonspat_diag=False, name_y=y_var, name_x=[
↪'PS90','UE90'], name_regimes=r_var, name_ds='NAT')
>>> olsr.betas
array([[ 0.39642899],
       [ 0.65583299],
       [ 0.48703937],
       [ 5.59835   ],
       [ 1.16210453],
       [ 0.53163886]])
>>> np.sqrt(olsr.vm.diagonal())
array([ 0.24816345,  0.09662678,  0.03628629,  0.46894564,  0.21667395,
        0.05945651])
>>> olsr.cols2regi
'all'
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

robust [string] Adjustment for robust standard errors Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

r2 [float] R squared Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

ar2 [float] Adjusted R squared Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

utu [float] Sum of squared residuals

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2ML [float] Sigma squared (maximum likelihood) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

f_stat [tuple] Statistic (float), p-value (float) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

logll [float] Log likelihood Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

aic [float] Akaike information criterion Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

schwarz [float] Schwarz information criterion Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

t_stat [list of tuples] t statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mulColli [float] Multicollinearity condition number Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

jarque_bera [dictionary] 'jb': Jarque-Bera statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

breusch_pagan [dictionary] 'bp': Breusch-Pagan statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

koenker_bassett: dictionary 'kb': Koenker-Bassett statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int). Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details).

white [dictionary] 'wh': White statistic (float); 'pvalue': p-value (float); 'df': degrees of freedom (int). Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

lm_error [tuple] Lagrange multiplier test for spatial error model; tuple contains the pair (statistic, p-value), where each is a float. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

lm_lag [tuple] Lagrange multiplier test for spatial lag model; tuple contains the pair (statistic, p-value), where each is a float. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

rlm_error [tuple] Robust lagrange multiplier test for spatial error model; tuple contains the pair (statistic, p-value), where each is a float. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

rlm_lag [tuple] Robust lagrange multiplier test for spatial lag model; tuple contains the pair (statistic, p-value), where each is a float. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

lm_sarma [tuple] Lagrange multiplier test for spatial SARMA model; tuple contains the pair (statistic, p-value), where each is a float. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

moran_res [tuple] Moran's I for the residuals; tuple containing the triple (Moran's I, standardized Moran's I, p-value)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

title [string] Name of the regression method used. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2n [float] Sigma squared (computed with n in the denominator)

sig2n_k [float] Sigma squared (computed with n-k in the denominator)

xtx [float] $X'X$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

xtxi [float] $(X'X)^{-1}$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi [string] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime.

cols2regi [list] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate.

nr [int] Number of different regimes in the 'regimes' list.

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression.

__init__(*self*, *y*, *x*, *regimes*, *w=None*, *robust=None*, *gwk=None*, *sig2n_k=True*, *nonspat_diag=True*, *spat_diag=False*, *moran=False*, *white_test=False*, *vm=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=True*, *cores=False*, *name_y=None*, *name_x=None*, *name_regimes=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes[, w, robust, ...])	Initialize self.

Attributes

mean_y
sig2n
sig2n_k
std_y
utu
vm

3.16.2 spreg.ML_Lag_Regimes

class spreg.**ML_Lag_Regimes**(*y*, *x*, *regimes*, *w=None*, *constant_regi='many'*, *cols2regi='all'*, *method='full'*, *epsilon=1e-07*, *regime_lag_sep=False*, *regime_err_sep=False*, *cores=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
ML estimation of the spatial lag model with regimes (note no consistency checks, diagnostics or constants added) [Ans88].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes
- 'many': a vector of ones is appended to x and considered different per regime (default)

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

w [Sparse matrix] Spatial weights sparse matrix

method [string] if 'full', brute force calculation (full matrix expressions) if 'ord', Ord eigenvalue method if 'LU', LU sparse matrix decomposition

epsilon [float] tolerance criterion in mimimize_scalar function and inverse_product

regime_lag_sep: boolean If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed accross regimes.

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

spat_diag [boolean] if True, include spatial diagnostics (not implemented yet)

vm [boolean] if True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

Examples

Open data baltim.dbf using pysal and create the variables matrices and weights matrix.

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> db =  libpysal.io.open(examples.get_path("baltim.dbf"),'r')
>>> ds_name = "baltim.dbf"
>>> y_name = "PRICE"
>>> y = np.array(db.by_col(y_name)).T
>>> y.shape = (len(y),1)
>>> x_names = ["NROOM","AGE","SQFT"]
>>> x = np.array([db.by_col(var) for var in x_names]).T
>>> ww = ps.open(ps.examples.get_path("baltim_q.gal"))
>>> w = ww.read()
>>> ww.close()
>>> w_name = "baltim_q.gal"
>>> w.transform = 'r'
```

Since in this example we are interested in checking whether the results vary by regimes, we use CITCOU to define whether the location is in the city or outside the city (in the county):

```
>>> regimes = db.by_col("CITCOU")
```

Now we can run the regression with all parameters:

```
>>> mllag = ML_Lag_Regimes(y,x,regimes,w=w,name_y=y_name,name_x=x_names,
↪     name_w=w_name,name_ds=ds_name,name_regimes="CITCOU")
>>> np.around(mllag.betas, decimals=4)
array([[-15.0059],
       [  4.496 ],
       [ -0.0318],
       [  0.35  ],
       [ -4.5404],
       [  3.9219],
       [ -0.1702],
       [  0.8194],
```

(continues on next page)

(continued from previous page)

```
       [  0.5385]])
>>> "{0:.6f}".format(mllag.rho)
'0.538503'
>>> "{0:.6f}".format(mllag.mean_y)
'44.307180'
>>> "{0:.6f}".format(mllag.std_y)
'23.606077'
>>> np.around(np.diag(mllag.vm1), decimals=4)
array([  47.42  ,    2.3953,    0.0051,    0.0648,   69.6765,    3.2066,
          0.0116,    0.0486,    0.004 ,  390.7274])
>>> np.around(np.diag(mllag.vm), decimals=4)
array([ 47.42  ,   2.3953,   0.0051,   0.0648,  69.6765,   3.2066,
         0.0116,   0.0486,   0.004 ])
>>> "{0:.6f}".format(mllag.sig2)
'200.044334'
>>> "{0:.6f}".format(mllag.logll)
'-864.985056'
>>> "{0:.6f}".format(mllag.aic)
'1747.970112'
>>> "{0:.6f}".format(mllag.schwarz)
'1778.136835'
>>> mllag.title
'MAXIMUM LIKELIHOOD SPATIAL LAG - REGIMES (METHOD = full)'
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] (k+1)x1 array of estimated coefficients (rho first)

rho [float] estimate of spatial autoregressive coefficient Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

u [array] nx1 array of residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant, excluding the rho) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

method [string] log Jacobian method. if 'full': brute force (full matrix computations) if 'ord', Ord eigenvalue method if 'LU', LU sparse matrix decomposition

epsilon [float] tolerance criterion used in minimize_scalar function and inverse_product

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (k+1 x k+1), all coefficients

vm1 [array] Variance covariance matrix (k+2 x k+2), includes sig2 Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

logll [float] maximized log-likelihood (including constant terms) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

aic [float] Akaike information criterion Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

schwarz [float] Schwarz criterion Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

predy_e [array] predicted values from reduced form

e_pred [array] prediction errors using reduced form predicted values

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form)) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: ['one', 'many'] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes
- 'many': a vector of ones is appended to x and considered different per regime

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_lag_sep: boolean If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed accross regimes.

regime_err_sep: boolean always set to False - kept for compatibility with other regime models

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

Methods

ML_Lag_Regimes_Multi	

__init__(*self*, *y*, *x*, *regimes*, *w=None*, *constant_regi='many'*, *cols2regi='all'*, *method='full'*, *epsilon=1e-07*, *regime_lag_sep=False*, *regime_err_sep=False*, *cores=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

ML_Lag_Regimes_Multi(self, y, x, w_i, w, ...)	
__init__(self, y, x, regimes[, w, ...])	Initialize self.

Attributes

mean_y
sig2n
sig2n_k
std_y
utu
vm

3.16.3 spreg.ML_Error_Regimes

class spreg.**ML_Error_Regimes**(*y*, *x*, *regimes*, *w=None*, *constant_regi='many'*, *cols2regi='all'*, *method='full'*, *epsilon=1e-07*, *regime_err_sep=False*, *regime_lag_sep=False*, *cores=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
ML estimation of the spatial error model with regimes (note no consistency checks, diagnostics or constants added); Anselin (1988) [Ans88]

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous)

variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

w [Sparse matrix] Spatial weights sparse matrix

method [string] if 'full', brute force calculation (full matrix expressions) if 'ord', Ord eigenvalue computation if 'LU', LU sparse matrix decomposition

epsilon [float] tolerance criterion in mimimize_scalar function and inverse_product

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean Always False, kept for consistency in function call, ignored.

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

spat_diag [boolean] if True, include spatial diagnostics (not implemented yet)

vm [boolean] if True, include variance-covariance matrix in summary results

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

Examples

Open data baltim.dbf using pysal and create the variables matrices and weights matrix.

```
>>> import numpy as np
>>> import libpysal
>>> db = libpysal.io.open(libpysal.examples.get_path("baltim.dbf"),'r')
>>> ds_name = "baltim.dbf"
>>> y_name = "PRICE"
>>> y = np.array(db.by_col(y_name)).T
>>> y.shape = (len(y),1)
>>> x_names = ["NROOM","AGE","SQFT"]
>>> x = np.array([db.by_col(var) for var in x_names]).T
>>> ww = ps.open(ps.examples.get_path("baltim_q.gal"))
>>> w = ww.read()
>>> ww.close()
>>> w_name = "baltim_q.gal"
>>> w.transform = 'r'
```

Since in this example we are interested in checking whether the results vary by regimes, we use CITCOU to define whether the location is in the city or outside the city (in the county):

```
>>> regimes = db.by_col("CITCOU")
```

Now we can run the regression with all parameters:

```
>>> mlerr = ML_Error_Regimes(y,x,regimes,w=w,name_y=y_name,name_x=x_names,
↪       name_w=w_name,name_ds=ds_name,name_regimes="CITCOU")
>>> np.around(mlerr.betas, decimals=4)
array([[ -2.3949],
       [  4.8738],
       [ -0.0291],
       [  0.3328],
       [ 31.7962],
       [  2.981 ],
       [ -0.2371],
       [  0.8058],
       [  0.6177]])
>>> "{0:.6f}".format(mlerr.lam)
'0.617707'
>>> "{0:.6f}".format(mlerr.mean_y)
'44.307180'
>>> "{0:.6f}".format(mlerr.std_y)
'23.606077'
>>> np.around(mlerr.vm1, decimals=4)
array([[   0.005 ,   -0.3535],
       [  -0.3535,  441.3039]])
>>> np.around(np.diag(mlerr.vm), decimals=4)
array([ 58.5055,   2.4295,   0.0072,   0.0639,  80.5925,   3.161 ,
         0.012 ,   0.0499,   0.005 ])
>>> np.around(mlerr.sig2, decimals=4)
array([[ 209.6064]])
>>> "{0:.6f}".format(mlerr.logll)
'-870.333106'
>>> "{0:.6f}".format(mlerr.aic)
'1756.666212'
>>> "{0:.6f}".format(mlerr.schwarz)
'1783.481077'
>>> mlerr.title
'MAXIMUM LIKELIHOOD SPATIAL ERROR - REGIMES (METHOD = full)'
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] (k+1)x1 array of estimated coefficients (lambda last)

lam [float] estimate of spatial autoregressive coefficient Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant, excluding the rho) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

method [string] log Jacobian method. if 'full': brute force (full matrix computations) if 'ord', Ord eigenvalue computation if 'LU', LU sparse matrix decomposition

epsilon [float] tolerance criterion used in minimize_scalar function and inverse_product

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (k+1 x k+1), all coefficients

vm1 [array] variance covariance matrix for lambda, sigma (2 x 2) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

logll [float] maximized log-likelihood (including constant terms) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: string Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a

list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_lag_sep: boolean If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed accross regimes.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

Methods

get_x_lag	

__init__(*self*, *y*, *x*, *regimes*, *w=None*, *constant_regi='many'*, *cols2regi='all'*, *method='full'*, *epsilon=1e-07*, *regime_err_sep=False*, *regime_lag_sep=False*, *cores=False*, *spat_diag=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes[, w, ...])	Initialize self.
get_x_lag(self, w, regimes_att)	

Attributes

mean_y
sig2n
sig2n_k
std_y
utu
vm

3.16.4 spreg.GM_Lag_Regimes

class spreg.**GM_Lag_Regimes**(*y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *robust=None*, *gwk=None*, *sig2n_k=False*, *spat_diag=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_lag_sep=False*, *regime_err_sep=True*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_regimes=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)

Spatial two stage least squares (S2SLS) with regimes; [Ans88]

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x); cannot be used in combination with h

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

w [pysal W object] Spatial weights object

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

regime_lag_sep: boolean If True (default), the spatial parameter for spatial lag is also computed according to different regimes. If False, the spatial parameter is fixed accross regimes. Option valid only when regime_err_sep=True

regime_err_sep: boolean If True, a separate regression is run for each regime.

robust [string] If 'white', then a White consistent estimator of the variance-covariance matrix is given. If 'hac', then a HAC consistent estimator of the variance-covariance matrix is given. If 'ogmm', then Optimal GMM is used to estimate betas and the variance-covariance matrix. Default set to None.

gwk [pysal W object] Kernel spatial weights needed for HAC estimation. Note: matrix must have ones along the main diagonal.

sig2n_k [boolean] If True, then use n-k to estimate sigma^2. If False, use n.

spat_diag [boolean] If True, then compute Anselin-Kelejian test

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
```

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = [‘Var1’,’Var2’,’. . .] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial lag model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> from libpysal import weights
>>> w = weights.Rook.from_shapefile(examples.get_path("NAT.shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

This class runs a lag model, which means that includes the spatial lag of the dependent variable on the right-hand side of the equation. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> model=GM_Lag_Regimes(y, x, regimes, w=w, regime_lag_sep=False, regime_err_
↪sep=False, name_y=y_var, name_x=x_var, name_regimes=r_var, name_ds='NAT', name_
↪w='NAT.shp')
>>> model.betas
array([[ 1.28897623],
       [ 0.79777722],
       [ 0.56366891],
       [ 8.73327838],
       [ 1.30433406],
       [ 0.62418643],
       [-0.39993716]])
```

Once the model is run, we can have a summary of the output by typing: model.summary . Alternatively, we can obtain the standard error of the coefficient estimates by calling:

```
>>> model.std_err
array([ 0.44682888,  0.14358192,  0.05655124,  1.06044865,  0.20184548,
        0.06118262,  0.12387232])
```

In the example above, all coefficients but the spatial lag vary according to the regime. It is also possible to have the spatial lag varying according to the regime, which effective will result in an independent spatial lag model estimated for each regime. To run these models, the argument regime_lag_sep must be set to True:

```
>>> model=GM_Lag_Regimes(y, x, regimes, w=w, regime_lag_sep=True, name_y=y_var,␣
↪name_x=x_var, name_regimes=r_var, name_ds='NAT', name_w='NAT.shp')
>>> print np.hstack((np.array(model.name_z).reshape(8,1),model.betas,np.
↪sqrt(model.vm.diagonal().reshape(8,1))))
[['0_CONSTANT' '1.36584769' '0.39854720']
 ['0_PS90' '0.80875730' '0.11324884']
 ['0_UE90' '0.56946813' '0.04625087']
 ['0_W_HR90' '-0.4342438' '0.13350159']
 ['1_CONSTANT' '7.90731073' '1.63601874']
 ['1_PS90' '1.27465703' '0.24709870']
 ['1_UE90' '0.60167693' '0.07993322']
 ['1_W_HR90' '-0.2960338' '0.19934459']]
```

Alternatively, we can type: 'model.summary' to see the organized results output. The class is flexible enough to accomodate a spatial lag model that, besides the spatial lag of the dependent variable, includes other non-spatial endogenous regressors. As an example, we will add the endogenous variable RD90 (resource deprivation) and we decide to instrument for it with FP89 (families below poverty):

```
>>> yd_var = ['RD90']
>>> yd = np.array([db.by_col(name) for name in yd_var]).T
```

(continues on next page)

(continued from previous page)

```
>>> q_var = ['FP89']
>>> q = np.array([db.by_col(name) for name in q_var]).T
```

And we can run the model again:

```
>>> model = GM_Lag_Regimes(y, x, regimes, yend=yd, q=q, w=w, regime_lag_sep=False,
↪ regime_err_sep=False, name_y=y_var, name_x=x_var, name_yend=yd_var, name_q=q_
↪var, name_regimes=r_var, name_ds='NAT', name_w='NAT.shp')
>>> model.betas
array([[ 3.42195202],
       [ 1.03311878],
       [ 0.14308741],
       [ 8.99740066],
       [ 1.91877758],
       [-0.32084816],
       [ 2.38918212],
       [ 3.67243761],
       [ 0.06959139]])
```

Once the model is run, we can obtain the standard error of the coefficient estimates. Alternatively, we can have a summary of the output by typing: model.summary

```
>>> model.std_err
array([ 0.49163311,  0.12237382,  0.05633464,  0.72555909,  0.17250521,
        0.06749131,  0.27370369,  0.25106224,  0.05804213])
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

kstar [integer] Number of endogenous variables. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

yend [array] Two dimensional array with n rows and one column for each endogenous variable Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z [array] nxk array of variables (combination of x and yend) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

h [array] nxl array of instruments (combination of x and q) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

robust [string] Adjustment for robust standard errors Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form)) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

utu [float] Sum of squared residuals

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

ak_test [tuple] Anselin-Kelejian test; tuple contains the pair (statistic, p-value) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_gwk [string] Name of kernel weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2n [float] Sigma squared (computed with n in the denominator)

sig2n_k [float] Sigma squared (computed with n-k in the denominator)

hth [float] $H'H$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

hthi [float] $(H'H)^{-1}$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

varb [array] $(Z'H(H'H)^{-1}H'Z)^{-1}$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

zthhthi [array] $Z'H(H'H)^{-1}$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

pfora1a2 [array] n(zthhthi)'varb Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: string Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime.

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_lag_sep: boolean If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed accross regimes.

regime_err_sep: boolean If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

Methods

GM_Lag_Regimes_Multi	
sp_att_reg	

__init__(*self*, *y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *robust=None*, *gwk=None*, *sig2n_k=False*, *spat_diag=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_lag_sep=False*, *regime_err_sep=True*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_regimes=None*, *name_w=None*, *name_gwk=None*, *name_ds=None*)

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

Methods

GM_Lag_Regimes_Multi(self, y, x, w_i, w, ...)	
__init__(self, y, x, regimes[, yend, q, w, ...])	Initialize self.
sp_att_reg(self, w_i, regi_ids, wy)	

Attributes

mean_y
pfora1a2
sig2n
sig2n_k
std_y
utu
vm

3.16.5 spreg.GM_Error_Regimes

class spreg.**GM_Error_Regimes**(*y*, *x*, *regimes*, *w*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *cores=False*, *name_ds=None*, *name_regimes=None*)

GMM method for a spatial error model with regimes, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object

constant_regi: string, optional Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean Always False, kept for consistency, ignored.

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = ['Var1','Var2',...] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> model = GM_Error_Regimes(y, x, regimes, w=w, name_y=y_var, name_x=x_var, name_
↪regimes=r_var, name_ds='NAT.dbf')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. Note that because we are running the classical GMM error model from 1998/99, the spatial parameter is obtained as a point estimate, so although you get a value for it (there are for coefficients under model.betas), you cannot perform inference on it (there are only three values in model.se_betas). Alternatively, we can have a summary of the output by typing: model.summary

```
>>> print model.name_x
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', 'lambda']
>>> np.around(model.betas, decimals=6)
array([[ 0.074807],
       [ 0.786107],
       [ 0.538849],
       [ 5.103756],
       [ 1.196009],
       [ 0.600533],
       [ 0.364103]])
>>> np.around(model.std_err, decimals=6)
array([ 0.379864,  0.152316,  0.051942,  0.471285,  0.19867 ,  0.057252])
>>> np.around(model.z_stat, decimals=6)
array([[  0.196932,   0.843881],
       [  5.161042,   0.      ],
       [ 10.37397 ,   0.      ],
       [ 10.829455,   0.      ],
       [  6.02007 ,   0.      ],
       [ 10.489215,   0.      ]])
>>> np.around(model.sig2, decimals=6)
28.172732
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

vm [array] Variance covariance matrix (kxk)

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: string Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes
- 'many': a vector of ones is appended to x and considered different per regime

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *regimes*, *w*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *cores=False*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes, w[, vm, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.6 spreg.GM_Error_Het_Regimes

class spreg.**GM_Error_Het_Regimes**(*y*, *x*, *regimes*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
GMM method for a spatial error model with heteroskedasticity and regimes; based on Arraiz et al [ADKP10], following Anselin [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes
- 'many': a vector of ones is appended to x and considered different per regime (default)

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean Always False, kept for consistency, ignored.

max_iter [int] Maximum number of iterations of steps 2a and 2b from Arraiz et al. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from Arraiz et al. Note: max_iter provides an additional stop condition.

step1c [boolean] If True, then include Step 1c from Arraiz et al.

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = [‘Var1’,’Var2’,’. . .] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Error_Het_Regimes(y, x, regimes, w=w, step1c=True, name_y=y_var,␣
↪name_x=x_var, name_regimes=r_var, name_ds='NAT.dbf')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that explicitly accounts for heteroskedasticity and that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. Alternatively, we can have a summary of the output by typing: model.summary

```
>>> print reg.name_x
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', 'lambda']
>>> np.around(reg.betas, decimals=6)
array([[ 0.009121],
       [ 0.812973],
       [ 0.549355],
       [ 5.00279 ],
       [ 1.200929],
       [ 0.614681],
       [ 0.429277]])
>>> np.around(reg.std_err, decimals=6)
array([ 0.355844,  0.221743,  0.059276,  0.686764,  0.35843 ,  0.092788,
        0.02524 ])
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

vm [array] Variance covariance matrix (kxk)

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: string Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes
- 'many': a vector of ones is appended to x and considered different per regime

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *regimes*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes, w[, max_iter, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.7 spreg.GM_Error_Hom_Regimes

class spreg.**GM_Error_Hom_Regimes**(*y*, *x*, *regimes*, *w*, *max_iter=1*, *epsilon=1e-05*, *A1='het'*, *cores=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
GMM method for a spatial error model with homoskedasticity, with regimes, results and diagnostics; based on Drukker et al. (2013) [DEP13], following Anselin (2011) [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean Always False, kept for consistency, ignored.

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

A1 [string] If A1='het', then the matrix A1 is defined as in [ADKP10]. If A1='hom', then as in [Ans11]. If A1='hom_sc', then as in [DEP13] and [DPR13].

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = ['Var1','Var2',...] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial lag model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Error_Hom_Regimes(y, x, regimes, w=w, name_y=y_var, name_x=x_var,␣
↪name_ds='NAT')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that assumes homoskedasticity but that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. This is why you obtain as many coefficient estimates as standard errors, which you calculate taking the square root of the diagonal of the variance-covariance matrix of the parameters. Alternatively, we can have a summary of the output by typing: model.summary >>> print reg.name_x ['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', 'lambda']

```
>>> print np.around(reg.betas,4)
[[ 0.069 ]
 [ 0.7885]
 [ 0.5398]
 [ 5.0948]
 [ 1.1965]
 [ 0.6018]
 [ 0.4104]]
```

```
>>> print np.sqrt(reg.vm.diagonal())
[ 0.39105854  0.15664624  0.05254328  0.48379958  0.20018799  0.05834139
  0.01882401]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

vm [array] Variance covariance matrix (kxk)

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

xtx [float] $X'X$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi: string Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *regimes*, *w*, *max_iter=1*, *epsilon=1e-05*, *A1='het'*, *cores=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes, w[, max_iter, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.8 spreg.GM_Combo_Regimes

class spreg.**GM_Combo_Regimes**(*y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *cores=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
GMM method for a spatial lag and error model with regimes and endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

w [pysal W object] Spatial weights object (always needed)

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed accross regimes.

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = [‘Var1’,’Var2’,’...] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial lag model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

The Combo class runs an SARAR model, that is a spatial lag+error model. In this case we will run a simple version of that, where we have the spatial effects as well as exogenous variables. Since it is a spatial model, we have to pass in the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> model = GM_Combo_Regimes(y, x, regimes, w=w, name_y=y_var, name_x=x_var, name_
↪regimes=r_var, name_ds='NAT')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. Note that because we are running the classical GMM error model from 1998/99, the spatial parameter is obtained as a point estimate, so although you get a value for it (there are for coefficients under model.betas), you cannot perform inference on it (there are only three values in model.se_betas). Also, this regression uses a two stage least squares estimation method that accounts for the endogeneity created by the spatial lag of the dependent variable. We can have a summary of the output by typing: model.summary Alternatively, we can check the betas:

```
>>> print model.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '_Global_W_
↪HR90', 'lambda']
>>> print np.around(model.betas,4)
[[ 1.4607]
 [ 0.958 ]
 [ 0.5658]
 [ 9.113 ]
 [ 1.1338]
 [ 0.6517]
 [-0.4583]
 [ 0.6136]]
```

And lambda:

```
>>> print 'lambda: ', np.around(model.betas[-1], 4)
lambda:  [ 0.6136]
```

This class also allows the user to run a spatial lag+error model with the extra feature of including non-spatial endogenous regressors. This means that, in addition to the spatial lag and error, we consider some of the variables on the right-hand side of the equation as endogenous and we instrument for this. In this case we consider RD90 (resource deprivation) as an endogenous regressor. We use FP89 (families below poverty) for this and hence put it in the instruments parameter, 'q'.

```
>>> yd_var = ['RD90']
>>> yd = np.array([db.by_col(name) for name in yd_var]).T
>>> q_var = ['FP89']
>>> q = np.array([db.by_col(name) for name in q_var]).T
```

And then we can run and explore the model analogously to the previous combo:

```
>>> model = GM_Combo_Regimes(y, x, regimes, yd, q, w=w, name_y=y_var, name_x=x_
↪var, name_yend=yd_var, name_q=q_var, name_regimes=r_var, name_ds='NAT')
>>> print model.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '0_RD90', '1_
↪RD90', '_Global_W_HR90', 'lambda']
>>> print model.betas
[[ 3.41963782]
 [ 1.04065841]
 [ 0.16634393]
 [ 8.86544628]
 [ 1.85120528]
 [-0.24908469]
 [ 2.43014046]
 [ 3.61645481]
 [ 0.03308671]
 [ 0.18684992]]
>>> print np.sqrt(model.vm.diagonal())
[ 0.53067577  0.13271426  0.06058025  0.76406411  0.17969783  0.07167421
  0.28943121  0.25308326  0.06126529]
>>> print 'lambda: ', np.around(model.betas[-1], 4)
lambda:  [ 0.1868]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

yend [array] Two dimensional array with n rows and one column for each endogenous variable Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z [array] nxk array of variables (combination of x and yend) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form)) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2 [float] Sigma squared used in computations (based on filtered residuals) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi [string] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed accross regimes.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *cores=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes[, yend, q, w, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.9 spreg.GM_Combo_Hom_Regimes

class spreg.**GM_Combo_Hom_Regimes**(*y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *cores=False*, *max_iter=1*, *epsilon=1e-05*, *A1='het'*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
GMM method for a spatial lag and error model with homoskedasticity, regimes and endogenous variables, with results and diagnostics; based on Drukker et al. (2013) [DEP13], following Anselin (2011) [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object (always needed)

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

regime_lag_sep [boolean] If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed across regimes.

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

A1 [string] If A1='het', then the matrix A1 is defined as in [ADKP10]. If A1='hom', then as in [Ans11]. If A1='hom_sc', then as in [DEP13] and [DPR13].

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = ['Var1','Var2',...] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial combo model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

Example only with spatial lag

The Combo class runs an SARAR model, that is a spatial lag+error model. In this case we will run a simple version of that, where we have the spatial effects as well as exogenous variables. Since it is a spatial model, we have to pass in the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional. We can have a summary of the output by typing: model.summary Alternatively, we can check the betas:

```
>>> reg = GM_Combo_Hom_Regimes(y, x, regimes, w=w, A1='hom_sc', name_y=y_var,␣
↪name_x=x_var, name_regimes=r_var, name_ds='NAT')
>>> print reg.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '_Global_W_
↪HR90', 'lambda']
>>> print np.around(reg.betas,4)
[[ 1.4607]
 [ 0.9579]
```

(continues on next page)

(continued from previous page)

```
 [ 0.5658]
 [ 9.1129]
 [ 1.1339]
 [ 0.6517]
 [-0.4583]
 [ 0.6634]]
```

This class also allows the user to run a spatial lag+error model with the extra feature of including non-spatial endogenous regressors. This means that, in addition to the spatial lag and error, we consider some of the variables on the right-hand side of the equation as endogenous and we instrument for this. In this case we consider RD90 (resource deprivation) as an endogenous regressor. We use FP89 (families below poverty) for this and hence put it in the instruments parameter, 'q'.

```
>>> yd_var = ['RD90']
>>> yd = np.array([db.by_col(name) for name in yd_var]).T
>>> q_var = ['FP89']
>>> q = np.array([db.by_col(name) for name in q_var]).T
```

And then we can run and explore the model analogously to the previous combo:

```
>>> reg = GM_Combo_Hom_Regimes(y, x, regimes, yd, q, w=w, A1='hom_sc', name_y=y_
↪var, name_x=x_var, name_yend=yd_var, name_q=q_var, name_regimes=r_var, name_ds=
↪'NAT')
>>> print reg.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '0_RD90', '1_
↪RD90', '_Global_W_HR90', 'lambda']
>>> print reg.betas
[[ 3.4196478 ]
 [ 1.04065595]
 [ 0.16630304]
 [ 8.86570777]
 [ 1.85134286]
 [-0.24921597]
 [ 2.43007651]
 [ 3.61656899]
 [ 0.03315061]
 [ 0.22636055]]
>>> print np.sqrt(reg.vm.diagonal())
[ 0.53989913  0.13506086  0.06143434  0.77049956  0.18089997  0.07246848
  0.29218837  0.25378655  0.06184801  0.06323236]
>>> print 'lambda: ', np.around(reg.betas[-1], 4)
lambda:  [ 0.2264]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

yend [array] Two dimensional array with n rows and one column for each endogenous variable Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z [array] nxk array of variables (combination of x and yend) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

h [array] nxl array of instruments (combination of x and q) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form)) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2 [float] Sigma squared used in computations (based on filtered residuals) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi [string] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

regime_lag_sep [boolean] If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed across regimes.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *cores=False*, *max_iter=1*, *epsilon=1e-05*, *A1='het'*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes[, yend, q, w, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.10 spreg.GM_Combo_Het_Regimes

class spreg.**GM_Combo_Het_Regimes**(*y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *cores=False*, *inv_method='power_exp'*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)

GMM method for a spatial lag and error model with heteroskedasticity, regimes and endogenous variables, with results and diagnostics; based on Arraiz et al [ADKP10], following Anselin [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object (always needed)

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

regime_lag_sep [boolean] If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed across regimes.

w_lags [integer] Orders of W to include as instruments for the spatially lagged dependent variable. For example, w_lags=1, then instruments are WX; if w_lags=2, then WX, WWX; and so on.

lag_q [boolean] If True, then include spatial lags of the additional instruments (q).

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

step1c [boolean] If True, then include Step 1c from [ADKP10].

inv_method [string] If "power_exp", then compute inverse using the power expansion. If "true_inv", then compute the true inverse. Note that true_inv will fail for large n.

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = [‘Var1’,’Var2’,’. . .] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial combo model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

Example only with spatial lag

The Combo class runs an SARAR model, that is a spatial lag+error model. In this case we will run a simple version of that, where we have the spatial effects as well as exogenous variables. Since it is a spatial model, we have to pass in the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional. We can have a summary of the output by typing: model.summary Alternatively, we can check the betas:

```
>>> reg = GM_Combo_Het_Regimes(y, x, regimes, w=w, step1c=True, name_y=y_var,␣
↪name_x=x_var, name_regimes=r_var, name_ds='NAT')
>>> print reg.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '_Global_W_
↪HR90', 'lambda']
>>> print np.around(reg.betas,4)
[[ 1.4613]
 [ 0.9587]
 [ 0.5658]
 [ 9.1157]
 [ 1.1324]
 [ 0.6518]
 [-0.4587]
 [ 0.7174]]
```

This class also allows the user to run a spatial lag+error model with the extra feature of including non-spatial endogenous regressors. This means that, in addition to the spatial lag and error, we consider some of the variables on the right-hand side of the equation as endogenous and we instrument for this. In this case we consider RD90 (resource deprivation) as an endogenous regressor. We use FP89 (families below poverty) for this and hence put it in the instruments parameter, 'q'.

```
>>> yd_var = ['RD90']
>>> yd = np.array([db.by_col(name) for name in yd_var]).T
>>> q_var = ['FP89']
>>> q = np.array([db.by_col(name) for name in q_var]).T
```

And then we can run and explore the model analogously to the previous combo:

```
>>> reg = GM_Combo_Het_Regimes(y, x, regimes, yd, q, w=w, step1c=True, name_y=y_
↪var, name_x=x_var, name_yend=yd_var, name_q=q_var, name_regimes=r_var, name_ds=
↪'NAT')
>>> print reg.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '0_RD90', '1_
↪RD90', '_Global_W_HR90', 'lambda']
>>> print reg.betas
[[ 3.41936197]
 [ 1.04071048]
 [ 0.16747219]
 [ 8.85820215]
 [ 1.847382  ]
```

(continues on next page)

(continued from previous page)

```
 [-0.24545394]
 [ 2.43189808]
 [ 3.61328423]
 [ 0.03132164]
 [ 0.29544224]]
>>> print np.sqrt(reg.vm.diagonal())
[ 0.53103804  0.20835827  0.05755679  1.00496234  0.34332131  0.10259525
  0.3454436   0.37932794  0.07611667  0.07067059]
>>> print 'lambda: ', np.around(reg.betas[-1], 4)
lambda:  [ 0.2954]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

e_pred [array] nx1 array of residuals (using reduced form)

predy [array] nx1 array of predicted y values

predy_e [array] nx1 array of predicted y values (using reduced form)

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

yend [array] Two dimensional array with n rows and one column for each endogenous variable Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z [array] nxk array of variables (combination of x and yend) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

h [array] nxl array of instruments (combination of x and q) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

pr2_e [float] Pseudo R squared (squared correlation between y and ypred_e (using reduced form)) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi [string] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean If True, the spatial parameter for spatial lag is also computed according to different regimes. If False (default), the spatial parameter is fixed across regimes.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *regimes*, *yend=None*, *q=None*, *w=None*, *w_lags=1*, *lag_q=True*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *cores=False*, *inv_method='power_exp'*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, regimes[, yend, q, w, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.11 spreg.GM_Endog_Error_Regimes

class spreg.**GM_Endog_Error_Regimes**(*y*, *x*, *yend*, *q*, *regimes*, *w*, *cores=False*, *vm=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*, *summ=True*, *add_lag=False*)
GMM method for a spatial error model with regimes and endogenous variables, with results and diagnostics; based on Kelejian and Prucha (1998, 1999) [KP98] [KP99].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating

for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

regime_lag_sep: boolean Always False, kept for consistency, ignored.

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = ['Var1','Var2',...] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

For the endogenous models, we add the endogenous variable RD90 (resource deprivation) and we decide to instrument for it with FP89 (families below poverty):

```
>>> yd_var = ['RD90']
>>> yend = np.array([db.by_col(name) for name in yd_var]).T
>>> q_var = ['FP89']
>>> q = np.array([db.by_col(name) for name in q_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables (exogenous and endogenous), the instruments and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> model = GM_Endog_Error_Regimes(y, x, yend, q, regimes, w=w, name_y=y_var,␣
↪name_x=x_var, name_yend=yd_var, name_q=q_var, name_regimes=r_var, name_ds='NAT.
↪dbf')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. Note that because we are running the classical GMM error model from 1998/99, the spatial parameter is obtained as a point estimate, so although you get a value for it (there are for coefficients under model.betas), you cannot perform inference on it (there are only three values in model.se_betas). Also, this regression uses a two stage least squares estimation method that accounts for the endogeneity created by the endogenous variables included. Alternatively, we can have a summary of the output by typing: model.summary

```
>>> print model.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '0_RD90', '1_
↪RD90', 'lambda']
>>> np.around(model.betas, decimals=5)
array([[ 3.59718],
       [ 1.0652 ],
       [ 0.15822],
       [ 9.19754],
       [ 1.88082],
       [-0.24878],
       [ 2.46161],
       [ 3.57943],
       [ 0.25564]])
>>> np.around(model.std_err, decimals=6)
array([ 0.522633,  0.137555,  0.063054,  0.473654,  0.18335 ,  0.072786,
        0.300711,  0.240413])
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

yend [array] Two dimensional array with n rows and one column for each endogenous variable Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z [array] nxk array of variables (combination of x and yend) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2 [float] Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details) Sigma squared used in computations

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi [['one', 'many']] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep: boolean If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *yend*, *q*, *regimes*, *w*, *cores=False*, *vm=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*, *summ=True*, *add_lag=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, yend, q, regimes, w[, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.12 spreg.GM_Endog_Error_Hom_Regimes

class spreg.**GM_Endog_Error_Hom_Regimes**(*y*, *x*, *yend*, *q*, *regimes*, *w*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *max_iter=1*, *epsilon=1e-05*, *A1='het'*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*, *summ=True*, *add_lag=False*)

GMM method for a spatial error model with homoskedasticity, regimes and endogenous variables. Based on Drukker et al. (2013) [DEP13], following Anselin (2011) [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

regime_lag_sep [boolean] Always False, kept for consistency, ignored.

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

A1 [string] If A1='het', then the matrix A1 is defined as in [ADKP10]. If A1='hom', then as in [Ans11]. If A1='hom_sc', then as in [DEP13] and [DPR13].

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = ['Var1','Var2',...] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

For the endogenous models, we add the endogenous variable RD90 (resource deprivation) and we decide to instrument for it with FP89 (families below poverty):

```
>>> yd_var = ['RD90']
>>> yend = np.array([db.by_col(name) for name in yd_var]).T
>>> q_var = ['FP89']
>>> q = np.array([db.by_col(name) for name in q_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables (exogenous and endogenous), the instruments and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Endog_Error_Hom_Regimes(y, x, yend, q, regimes, w=w, A1='hom_sc',␣
↪name_y=y_var, name_x=x_var, name_yend=yd_var, name_q=q_var, name_regimes=r_var,␣
↪name_ds='NAT.dbf')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that assumes homoskedasticity but that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. Hence, we find the same number of betas as of standard errors, which we calculate taking the square root of the diagonal of the variance-covariance matrix. Alternatively, we can have a summary of the output by typing: model.summary

```
>>> print reg.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '0_RD90', '1_
↪RD90', 'lambda']
```

```
>>> print np.around(reg.betas,4)
[[ 3.5973]
 [ 1.0652]
 [ 0.1582]
 [ 9.198 ]
 [ 1.8809]
 [-0.2489]
 [ 2.4616]
 [ 3.5796]
 [ 0.2541]]
```

```
>>> print np.around(np.sqrt(reg.vm.diagonal()),4)
[ 0.5204  0.1371  0.0629  0.4721  0.1824  0.0725  0.2992  0.2395  0.024 ]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

yend [array] Two dimensional array with n rows and one column for each endogenous variable Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z [array] nxk array of variables (combination of x and yend) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

h [array] nxl array of instruments (combination of x and q) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

sig2 [float] Sigma squared used in computations Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

hth [float] $H'H$. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi [['one', 'many']] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *yend*, *q*, *regimes*, *w*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *max_iter=1*, *epsilon=1e-05*, *A1='het'*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*, *summ=True*, *add_lag=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, yend, q, regimes, w[, ...])	Initialize self.

Attributes

mean_y
std_y

3.16.13 spreg.GM_Endog_Error_Het_Regimes

class spreg.**GM_Endog_Error_Het_Regimes**(*y*, *x*, *yend*, *q*, *regimes*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *inv_method='power_exp'*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*, *summ=True*, *add_lag=False*)
GMM method for a spatial error model with heteroskedasticity, regimes and endogenous variables, with results

and diagnostics; based on Arraiz et al [ADKP10], following Anselin [Ans11].

Parameters

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, excluding the constant

yend [array] Two dimensional array with n rows and one column for each endogenous variable

q [array] Two dimensional array with n rows and one column for each external exogenous variable to use as instruments (note: this should not contain any variables from x)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

w [pysal W object] Spatial weights object

constant_regi: string Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all' (default), all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

regime_lag_sep [boolean] Always False, kept for consistency, ignored.

max_iter [int] Maximum number of iterations of steps 2a and 2b from [ADKP10]. Note: epsilon provides an additional stop condition.

epsilon [float] Minimum change in lambda required to stop iterations of steps 2a and 2b from [ADKP10]. Note: max_iter provides an additional stop condition.

step1c [boolean] If True, then include Step 1c from [ADKP10].

inv_method [string] If "power_exp", then compute inverse using the power expansion. If "true_inv", then compute the true inverse. Note that true_inv will fail for large n.

vm [boolean] If True, include variance-covariance matrix in summary results

cores [boolean] Specifies if multiprocessing is to be used Default: no multiprocessing, cores = False Note: Multiprocessing may not work on all platforms.

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_q [list of strings] Names of instruments for use in output

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regime variable for use in the output

Examples

We first need to import the needed modules, namely numpy to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis.

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

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

Extract the HR90 column (homicide rates in 1990) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y_var = 'HR90'
>>> y = np.array([db.by_col(y_var)]).reshape(3085,1)
```

Extract UE90 (unemployment rate) and PS90 (population structure) vectors from the DBF to be used as independent variables in the regression. Other variables can be inserted by adding their names to x_var, such as x_var = ['Var1','Var2',...] Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in.

```
>>> x_var = ['PS90','UE90']
>>> x = np.array([db.by_col(name) for name in x_var]).T
```

For the endogenous models, we add the endogenous variable RD90 (resource deprivation) and we decide to instrument for it with FP89 (families below poverty):

```
>>> yd_var = ['RD90']
>>> yend = np.array([db.by_col(name) for name in yd_var]).T
>>> q_var = ['FP89']
>>> q = np.array([db.by_col(name) for name in q_var]).T
```

The different regimes in this data are given according to the North and South dummy (SOUTH).

```
>>> r_var = 'SOUTH'
>>> regimes = db.by_col(r_var)
```

Since we want to run a spatial error model, we need to specify the spatial weights matrix that includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `NAT.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are all set with the preliminaries, we are good to run the model. In this case, we will need the variables (exogenous and endogenous), the instruments and the weights matrix. If we want to have the names of the variables printed in the output summary, we will have to pass them in as well, although this is optional.

```
>>> reg = GM_Endog_Error_Het_Regimes(y, x, yend, q, regimes, w=w, step1c=True,␣
↪name_y=y_var, name_x=x_var, name_yend=yd_var, name_q=q_var, name_regimes=r_var,␣
↪name_ds='NAT.dbf')
```

Once we have run the model, we can explore a little bit the output. The regression object we have created has many attributes so take your time to discover them. This class offers an error model that explicitly accounts for heteroskedasticity and that unlike the models from `spreg.error_sp`, it allows for inference on the spatial parameter. Hence, we find the same number of betas as of standard errors, which we calculate taking the square root of the diagonal of the variance-covariance matrix Alternatively, we can have a summary of the output by typing: model.summary

```
>>> print reg.name_z
['0_CONSTANT', '0_PS90', '0_UE90', '1_CONSTANT', '1_PS90', '1_UE90', '0_RD90', '1_
↪RD90', 'lambda']
```

```
>>> print np.around(reg.betas,4)
[[ 3.5944]
 [ 1.065 ]
 [ 0.1587]
 [ 9.184 ]
 [ 1.8784]
 [-0.2466]
 [ 2.4617]
 [ 3.5756]
 [ 0.2908]]
```

```
>>> print np.around(np.sqrt(reg.vm.diagonal()),4)
[ 0.5043  0.2132  0.0581  0.6681  0.3504  0.0999  0.3686  0.3402  0.028 ]
```

Attributes

summary [string] Summary of regression results and diagnostics (note: use in conjunction with the print command)

betas [array] kx1 array of estimated coefficients

u [array] nx1 array of residuals

e_filtered [array] nx1 array of spatially filtered residuals

predy [array] nx1 array of predicted y values

n [integer] Number of observations

k [integer] Number of variables for which coefficients are estimated (including the constant). Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

y [array] nx1 array for dependent variable

x [array] Two dimensional array with n rows and one column for each independent (exogenous) variable, including the constant Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

yend [array] Two dimensional array with n rows and one column for each endogenous variable Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

q [array] Two dimensional array with n rows and one column for each external exogenous variable used as instruments Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z [array] nxk array of variables (combination of x and yend) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

h [array] nxl array of instruments (combination of x and q) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iter_stop [string] Stop criterion reached during iteration of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

iteration [integer] Number of iterations of steps 2a and 2b from [ADKP10]. Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

mean_y [float] Mean of dependent variable

std_y [float] Standard deviation of dependent variable

vm [array] Variance covariance matrix (kxk)

pr2 [float] Pseudo R squared (squared correlation between y and ypred) Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

std_err [array] 1xk array of standard errors of the betas Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

z_stat [list of tuples] z statistic; each tuple contains the pair (statistic, p-value), where each is a float Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

name_y [string] Name of dependent variable for use in output

name_x [list of strings] Names of independent variables for use in output

name_yend [list of strings] Names of endogenous variables for use in output

name_z [list of strings] Names of exogenous and endogenous variables for use in output

name_q [list of strings] Names of external instruments

name_h [list of strings] Names of all instruments used in ouput

name_w [string] Name of weights matrix for use in output

name_ds [string] Name of dataset for use in output

name_regimes [string] Name of regimes variable for use in output

title [string] Name of the regression method used Only available in dictionary 'multi' when multiple regressions (see 'multi' below for details)

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

constant_regi [string] Ignored if regimes=False. Constant option for regimes. Switcher controlling the constant term setup. It may take the following values:

- 'one': a vector of ones is appended to x and held constant across regimes.
- 'many': a vector of ones is appended to x and considered different per regime (default).

cols2regi [list, 'all'] Ignored if regimes=False. Argument indicating whether each column of x should be considered as different per regime or held constant across regimes (False). If a

list, k booleans indicating for each variable the option (True if one per regime, False to be held constant). If 'all', all the variables vary by regime.

regime_err_sep [boolean] If True, a separate regression is run for each regime.

kr [int] Number of variables/columns to be "regimized" or subject to change by regime. These will result in one parameter estimate by regime for each variable (i.e. nr parameters per variable)

kf [int] Number of variables/columns to be considered fixed or global across regimes and hence only obtain one parameter estimate

nr [int] Number of different regimes in the 'regimes' list

multi [dictionary] Only available when multiple regressions are estimated, i.e. when regime_err_sep=True and no variable is fixed across regimes. Contains all attributes of each individual regression

__init__(*self*, *y*, *x*, *yend*, *q*, *regimes*, *w*, *max_iter=1*, *epsilon=1e-05*, *step1c=False*, *constant_regi='many'*, *cols2regi='all'*, *regime_err_sep=False*, *regime_lag_sep=False*, *inv_method='power_exp'*, *cores=False*, *vm=False*, *name_y=None*, *name_x=None*, *name_yend=None*, *name_q=None*, *name_w=None*, *name_ds=None*, *name_regimes=None*, *summ=True*, *add_lag=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, y, x, yend, q, regimes, w[, ...])	Initialize self.

Attributes

mean_y
std_y

3.17 Seemingly-Unrelated Regressions

Seeimingly-unrelated regression models are a generalization of linear regression. These models (and their spatial generalizations) allow for correlation in the residual terms between groups that use the same model. In spatial Seeimingly-Unrelated Regressions, the error terms across groups are allowed to exhibit a structured type of correlation: spatail correlation.

spreg.SUR(bigy, bigX[, w, regimes, ...])	User class for SUR estimation, both two step as well as iterated
spreg.SURerrorGM(bigy, bigX, w[, regimes, ...])	User class for SUR Error estimation by Maximum Likelihood
spreg.SURerrorML(bigy, bigX, w[, regimes, ...])	User class for SUR Error estimation by Maximum Likelihood
spreg.SURlagIV(bigy, bigX[, bigyend, bigq, ...])	User class for spatial lag estimation using IV
spreg.ThreeSLS(bigy, bigX, bigyend, bigq[, ...])	User class for 3SLS estimation

3.17.1 spreg.SUR

class spreg.**SUR**(*bigy*, *bigX*, *w=None*, *regimes=None*, *nonspat_diag=True*, *spat_diag=False*, *vm=False*, *iter=False*, *maxiter=5*, *epsilon=1e-05*, *verbose=False*, *name_bigy=None*, *name_bigX=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)

User class for SUR estimation, both two step as well as iterated

Parameters

bigy [dictionary] with vector for dependent variable by equation

bigX [dictionary] with matrix of explanatory variables by equation (note, already includes constant term)

w [spatial weights object] default = None

regimes [list] default = None. List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

nonspat_diag: boolean flag for non-spatial diagnostics, default = True

spat_diag [boolean] flag for spatial diagnostics, default = False

iter [boolean] whether or not to use iterated estimation. default = False

maxiter [int] maximum iterations; default = 5

epsilon [float] precision criterion to end iterations. default = 0.00001

verbose [boolean] flag to print out iteration number and value of log det(sig) at the beginning and the end of the iteration

name_bigy [dictionary] with name of dependent variable for each equation. default = None, but should be specified is done when sur_stackxy is used

name_bigX [dictionary] with names of explanatory variables for each equation. default = None, but should be specified is done when sur_stackxy is used

name_ds [string] name for the data set

name_w [string] name for the weights file

name_regimes [string] name of regime variable for use in the output

Examples

First import libpysal to load the spatial analysis tools.

```
>>> import libpysal
```

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

The specification of the model to be estimated can be provided as lists. Each equation should be listed separately. In this example, equation 1 has HR80 as dependent variable and PS80 and UE80 as exogenous regressors. For equation 2, HR90 is the dependent variable, and PS90 and UE90 the exogenous regressors.

```
>>> y_var = ['HR80','HR90']
>>> x_var = [['PS80','UE80'],['PS90','UE90']]
```

Although not required for this method, we can load a weights matrix file to allow for spatial diagnostics.

```
>>> w = libpysal.weights.Queen.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
>>> w.transform='r'
```

The SUR method requires data to be provided as dictionaries. PySAL provides the tool sur_dictxy to create these dictionaries from the list of variables. The line below will create four dictionaries containing respectively the dependent variables (bigy), the regressors (bigX), the dependent variables' names (bigyvars) and regressors' names (bigXvars). All these will be created from th database (db) and lists of variables (y_var and x_var) created above.

```
>>> bigy,bigX,bigyvars,bigXvars = pysal.spreg.sur_utils.sur_dictxy(db,y_var,x_var)
```

We can now run the regression and then have a summary of the output by typing: 'print(reg.summary)'

```
>>> reg = SUR(bigy,bigX,w=w,name_bigy=bigyvars,name_bigX=bigXvars,spat_diag=True,
↪name_ds="nat")
>>> print(reg.summary)
REGRESSION
----------
SUMMARY OF OUTPUT: SEEMINGLY UNRELATED REGRESSIONS (SUR)
--------------------------------------------------------
Data set            :         nat
Weights matrix      :     unknown
Number of Equations :           2                 Number of Observations:
↪3085
Log likelihood (SUR):  -19902.966                 Number of Iterations  :
↪ 1
----------

SUMMARY OF EQUATION 1
---------------------
Dependent Variable  :        HR80                 Number of Variables   :
↪ 3
Mean dependent var  :      6.9276                 Degrees of Freedom    :
↪3082
S.D. dependent var  :      6.8251

------------------------------------------------------------------------------
↪--
            Variable     Coefficient       Std.Error     z-Statistic
↪Probability
------------------------------------------------------------------------------
↪--
          Constant_1       5.1390718       0.2624673      19.5798587       0.
↪0000000
                PS80       0.6776481       0.1219578       5.5564132       0.
↪0000000
                UE80       0.2637240       0.0343184       7.6846277       0.
↪0000000
------------------------------------------------------------------------------
↪--

SUMMARY OF EQUATION 2
---------------------
Dependent Variable  :        HR90                 Number of Variables   :
↪ 3
```

(continues on next page)

(continued from previous page)

```
Mean dependent var  :      6.1829                Degrees of Freedom    :
↪3082
S.D. dependent var  :      6.6403

------------------------------------------------------------------------------
↪--
            Variable     Coefficient       Std.Error     z-Statistic
↪Probability
------------------------------------------------------------------------------
↪--
          Constant_2       3.6139403       0.2534996      14.2561949       0.
↪0000000
                PS90       1.0260715       0.1121662       9.1477755       0.
↪0000000
                UE90       0.3865499       0.0341996      11.3027760       0.
↪0000000
------------------------------------------------------------------------------
↪--


REGRESSION DIAGNOSTICS
                                    TEST         DF       VALUE           PROB
                          LM test on Sigma         1     680.168           0.0000
                          LR test on Sigma         1     768.385           0.0000

OTHER DIAGNOSTICS - CHOW TEST BETWEEN EQUATIONS
                                 VARIABLES         DF       VALUE           PROB
                     Constant_1, Constant_2         1      26.729           0.0000
                                 PS80, PS90         1       8.241           0.0041
                                 UE80, UE90         1       9.384           0.0022

DIAGNOSTICS FOR SPATIAL DEPENDENCE
TEST                               DF       VALUE           PROB
Lagrange Multiplier (error)        2        1333.586        0.0000
Lagrange Multiplier (lag)          2        1275.821        0.0000

ERROR CORRELATION MATRIX
  EQUATION 1  EQUATION 2
    1.000000    0.469548
    0.469548    1.000000
================================ END OF REPORT
↪====================================
```

Attributes

bigy [dictionary] with y values

bigX [dictionary] with X values

bigXX [dictionary] with $X_t'X_r$ cross-products

bigXy [dictionary] with $X_t'y_r$ cross-products

n_eq [int] number of equations

n [int] number of observations in each cross-section

bigK [array] vector with number of explanatory variables (including constant) for each equation

bOLS [dictionary] with OLS regression coefficients for each equation

olsE [array] N x n_eq array with OLS residuals for each equation

bSUR [dictionary] with SUR regression coefficients for each equation

varb [array] variance-covariance matrix

bigE [array] n by n_eq array of residuals

sig_ols [array] Sigma matrix for OLS residuals (diagonal)

ldetS0 [float] log det(Sigma) for null model (OLS by equation)

niter [int] number of iterations (=0 for iter=False)

corr [array] inter-equation error correlation matrix

llik [float] log-likelihood (including the constant pi)

sur_inf [dictionary] with standard error, asymptotic t and p-value, one for each equation

lrtest [tuple] Likelihood Ratio test on off-diagonal elements of sigma (tuple with test,df,p-value)

lmtest [tuple] Lagrange Multipler test on off-diagonal elements of sigma (tuple with test,df,p-value)

lmEtest [tuple] Lagrange Multiplier test on error spatial autocorrelation in SUR (tuple with test, df, p-value)

lmlagtest [tuple] Lagrange Multiplier test on spatial lag autocorrelation in SUR (tuple with test, df, p-value)

surchow [array] list with tuples for Chow test on regression coefficients. each tuple contains test value, degrees of freedom, p-value

name_bigy [dictionary] with name of dependent variable for each equation

name_bigX [dictionary] with names of explanatory variables for each equation

name_ds [string] name for the data set

name_w [string] name for the weights file

name_regimes [string] name of regime variable for use in the output

__init__(*self*, *bigy*, *bigX*, *w=None*, *regimes=None*, *nonspat_diag=True*, *spat_diag=False*, *vm=False*, *iter=False*, *maxiter=5*, *epsilon=1e-05*, *verbose=False*, *name_bigy=None*, *name_bigX=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, bigy, bigX[, w, regimes, ...])	Initialize self.

3.17.2 spreg.SURerrorGM

class spreg.**SURerrorGM**(*bigy*, *bigX*, *w*, *regimes=None*, *nonspat_diag=True*, *spat_diag=False*, *vm=False*, *name_bigy=None*, *name_bigX=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)
User class for SUR Error estimation by Maximum Likelihood

Parameters

bigy [dictionary] with vectors of dependent variable, one for each equation

bigX [dictionary] with matrices of explanatory variables, one for each equation

w [spatial weights object]

regimes [list] List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

nonspat_diag [boolean] flag for non-spatial diagnostics, default = False

spat_diag [boolean] flag for spatial diagnostics, default = False (to be implemented)

vm [boolean] flag for asymptotic variance for lambda and Sigma, default = False (to be implemented)

name_bigy [dictionary] with name of dependent variable for each equation. default = None, but should be specified is done when sur_stackxy is used

name_bigX [dictionary] with names of explanatory variables for each equation. default = None, but should be specified is done when sur_stackxy is used

name_ds [string] name for the data set

name_w [string] name for the weights file

name_regimes [string] name of regime variable for use in the output

Examples

First import pysal to load the spatial analysis tools.

```
>>> import pysal
```

Open data on NCOVR US County Homicides (3085 areas) using pysal.open(). This is the DBF associated with the NAT shapefile. Note that pysal.open() also reads data in CSV format.

```
>>> db = pysal.open(pysal.examples.get_path("NAT.dbf"),'r')
```

The specification of the model to be estimated can be provided as lists. Each equation should be listed separately. Equation 1 has HR80 as dependent variable, and PS80 and UE80 as exogenous regressors. For equation 2, HR90 is the dependent variable, and PS90 and UE90 the exogenous regressors.

```
>>> y_var = ['HR80','HR90']
>>> x_var = [['PS80','UE80'],['PS90','UE90']]
>>> yend_var = [['RD80'],['RD90']]
>>> q_var = [['FP79'],['FP89']]
```

The SUR method requires data to be provided as dictionaries. PySAL provides the tool sur_dictxy to create these dictionaries from the list of variables. The line below will create four dictionaries containing respectively the dependent variables (bigy), the regressors (bigX), the dependent variables' names (bigyvars) and regressors' names (bigXvars). All these will be created from th database (db) and lists of variables (y_var and x_var) created above.

```
>>> bigy,bigX,bigyvars,bigXvars = pysal.spreg.sur_utils.sur_dictxy(db,y_var,x_var)
```

To run a spatial error model, we need to specify the spatial weights matrix. To do that, we can open an already existing gal file or create a new one. In this example, we will create a new one from NAT.shp and transform it to row-standardized.

```
>>> w = pysal.queen_from_shapefile(pysal.examples.get_path("NAT.shp"))
>>> w.transform='r'
```

We can now run the regression and then have a summary of the output by typing: print(reg.summary)

Alternatively, we can just check the betas and standard errors, asymptotic t and p-value of the parameters:

```
>>> reg = SURerrorGM(bigy,bigX,w=w,name_bigy=bigyvars,name_bigX=bigXvars,name_ds=
↪"NAT",name_w="nat_queen")
>>> reg.bSUR
{0: array([[ 3.9774686 ],
       [ 0.8902122 ],
       [ 0.43050364]]), 1: array([[ 2.93679118],
       [ 1.11002827],
       [ 0.48761542]])}
>>> reg.sur_inf
{0: array([[  0.37251477,  10.67734473,   0.        ],
       [  0.14224297,   6.25839157,   0.        ],
       [  0.04322388,   9.95985619,   0.        ]]), 1: array([[  0.33694902,   8.
↪71583239,   0.        ],
       [  0.13413626,   8.27537784,   0.        ],
       [  0.04033105,  12.09032295,   0.        ]])}
```

Attributes

n [int] number of observations in each cross-section

n_eq [int] number of equations

bigy [dictionary] with vectors of dependent variable, one for each equation

bigX [dictionary] with matrices of explanatory variables, one for each equation

bigK [array] n_eq x 1 array with number of explanatory variables by equation

bigylag [dictionary] spatially lagged dependent variable

bigXlag [dictionary] spatially lagged explanatory variable

lamsur [float] spatial autoregressive coefficient in ML SUR Error

bSUR [array] beta coefficients in ML SUR Error

varb [array] variance of beta coefficients in ML SUR Error

sig [array] error variance-covariance matrix in ML SUR Error

bigE [array] n by n_eq matrix of vectors of residuals for each equation

sur_inf [array] inference for regression coefficients, stand. error, t, p

surchow [array] list with tuples for Chow test on regression coefficients. each tuple contains test value, degrees of freedom, p-value

name_bigy [dictionary] with name of dependent variable for each equation

name_bigX [dictionary] with names of explanatory variables for each equation

name_ds [string] name for the data set

name_w [string] name for the weights file

name_regimes [string] name of regime variable for use in the output

__init__(*self*, *bigy*, *bigX*, *w*, *regimes=None*, *nonspat_diag=True*, *spat_diag=False*, *vm=False*, *name_bigy=None*, *name_bigX=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, bigy, bigX, w[, regimes, ...])	Initialize self.

3.17.3 spreg.SURerrorML

class spreg.**SURerrorML**(*bigy*, *bigX*, *w*, *regimes=None*, *nonspat_diag=True*, *spat_diag=False*, *vm=False*, *epsilon=1e-07*, *name_bigy=None*, *name_bigX=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)
User class for SUR Error estimation by Maximum Likelihood

Parameters

bigy [dictionary] with vectors of dependent variable, one for each equation

bigX [dictionary] with matrices of explanatory variables, one for each equation

w [spatial weights object]

regimes [list] default = None. List of n values with the mapping of each observation to a regime. Assumed to be aligned with 'x'.

epsilon [float] convergence criterion for ML iterations. default 0.0000001

nonspat_diag [boolean] flag for non-spatial diagnostics, default = True

spat_diag [boolean] flag for spatial diagnostics, default = False

vm [boolean] flag for asymptotic variance for lambda and Sigma, default = False

name_bigy [dictionary] with name of dependent variable for each equation. default = None, but should be specified is done when sur_stackxy is used

name_bigX [dictionary] with names of explanatory variables for each equation. default = None, but should be specified is done when sur_stackxy is used

name_ds [string] name for the data set

name_w [string] name for the weights file

name_regimes [string] name of regime variable for use in the output

Examples

First import libpysal to load the spatial analysis tools.

```
>>> import libpysal
```

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

The specification of the model to be estimated can be provided as lists. Each equation should be listed separately. Equation 1 has HR80 as dependent variable, and PS80 and UE80 as exogenous regressors. For equation 2, HR90 is the dependent variable, and PS90 and UE90 the exogenous regressors.

```
>>> y_var = ['HR80','HR90']
>>> x_var = [['PS80','UE80'],['PS90','UE90']]
>>> yend_var = [['RD80'],['RD90']]
>>> q_var = [['FP79'],['FP89']]
```

The SUR method requires data to be provided as dictionaries. PySAL provides the tool sur_dictxy to create these dictionaries from the list of variables. The line below will create four dictionaries containing respectively the dependent variables (bigy), the regressors (bigX), the dependent variables' names (bigyvars) and regressors' names (bigXvars). All these will be created from th database (db) and lists of variables (y_var and x_var) created above.

```
>>> bigy,bigX,bigyvars,bigXvars = pysal.spreg.sur_utils.sur_dictxy(db,y_var,x_var)
```

To run a spatial error model, we need to specify the spatial weights matrix. To do that, we can open an already existing gal file or create a new one. In this example, we will create a new one from NAT.shp and transform it to row-standardized.

```
>>> w = libpysal.weights.Queen.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
>>> w.transform='r'
```

We can now run the regression and then have a summary of the output by typing: print(reg.summary)

Alternatively, we can just check the betas and standard errors, asymptotic t and p-value of the parameters:

```
>>> reg = SURerrorML(bigy,bigX,w=w,name_bigy=bigyvars,name_bigX=bigXvars,name_ds=
↪"NAT",name_w="nat_queen")
>>> reg.bSUR
{0: array([[ 4.0222855 ],
       [ 0.88489646],
       [ 0.42402853]]), 1: array([[ 3.04923009],
       [ 1.10972634],
       [ 0.47075682]])}
```

```
>>> reg.sur_inf
{0: array([[  0.36692181,  10.96224141,   0.        ],
       [  0.14129077,   6.26294579,   0.        ],
       [  0.04267954,   9.93517021,   0.        ]]), 1: array([[  0.33139969,   9.
↪20106497,   0.        ],
       [  0.13352591,   8.31094371,   0.        ],
       [  0.04004097,  11.756878  ,   0.        ]])}
```

Attributes

n [int] number of observations in each cross-section

n2 [int] n/2

n_eq [int] number of equations

bigy [dictionary] with vectors of dependent variable, one for each equation

bigX [dictionary] with matrices of explanatory variables, one for each equation

bigK [array] n_eq x 1 array with number of explanatory variables by equation

bigylag [dictionary] spatially lagged dependent variable

bigXlag [dictionary] spatially lagged explanatory variable

lamols [array] spatial autoregressive coefficients from equation by equation ML-Error estimation

clikerr [float] concentrated log-likelihood from equation by equation ML-Error estimation (no constant)

bSUR0 [array] SUR estimation for betas without spatial autocorrelation

llik [float] log-likelihood for classic SUR estimation (includes constant)

lamsur [float] spatial autoregressive coefficient in ML SUR Error

bSUR [array] beta coefficients in ML SUR Error

varb [array] variance of beta coefficients in ML SUR Error

sig [array] error variance-covariance matrix in ML SUR Error

bigE [array] n by n_eq matrix of vectors of residuals for each equation

cliksurerr [float] concentrated log-likelihood from ML SUR Error (no constant)

sur_inf [array] inference for regression coefficients, stand. error, t, p

errllik [float] log-likelihood for error model without SUR (with constant)

surerrllik [float] log-likelihood for SUR error model (with constant)

lrtest [tuple] likelihood ratio test for off-diagonal Sigma elements

likrlambda [tuple] likelihood ratio test on spatial autoregressive coefficients

vm [array] asymptotic variance matrix for lambda and Sigma (only for vm=True)

lamsetp [array] inference for lambda, stand. error, t, p (only for vm=True)

lamtest [tuple] with test for constancy of lambda across equations (test value, degrees of freedom, p-value)

joinlam [tuple] with test for joint significance of lambda across equations (test value, degrees of freedom, p-value)

surchow [list] with tuples for Chow test on regression coefficients. each tuple contains test value, degrees of freedom, p-value

name_bigy [dictionary] with name of dependent variable for each equation

name_bigX [dictionary] with names of explanatory variables for each equation

name_ds [string] name for the data set

name_w [string] name for the weights file

name_regimes [string] name of regime variable for use in the output

__init__(*self*, *bigy*, *bigX*, *w*, *regimes=None*, *nonspat_diag=True*, *spat_diag=False*, *vm=False*, *epsilon=1e-07*, *name_bigy=None*, *name_bigX=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)

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

Methods

`__init__`(self, bigy, bigX, w[, regimes, ...])	Initialize self.

3.17.4 spreg.SURlagIV

class spreg.**SURlagIV**(*bigy*, *bigX*, *bigyend=None*, *bigq=None*, *w=None*, *regimes=None*, *vm=False*, *regime_lag_sep=False*, *w_lags=1*, *lag_q=True*, *nonspat_diag=True*, *spat_diag=False*, *name_bigy=None*, *name_bigX=None*, *name_bigyend=None*, *name_bigq=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)

User class for spatial lag estimation using IV

Parameters

bigy [dictionary] with vector for dependent variable by equation

bigX [dictionary] with matrix of explanatory variables by equation (note, already includes constant term)

bigyend [dictionary] with matrix of endogenous variables by equation (optional)

bigq [dictionary] with matrix of instruments by equation (optional)

w [spatial weights object, required]

vm [boolean] listing of full variance-covariance matrix, default = False

w_lags [integer] order of spatial lags for WX instruments, default = 1

lag_q [boolean] flag to apply spatial lag to other instruments, default = True

nonspat_diag [boolean] flag for non-spatial diagnostics, default = True

spat_diag [boolean] flag for spatial diagnostics, default = False

name_bigy [dictionary] with name of dependent variable for each equation. default = None, but should be specified. is done when sur_stackxy is used.

name_bigX [dictionary] with names of explanatory variables for each equation. default = None, but should be specified. is done when sur_stackxy is used.

name_bigyend [dictionary] with names of endogenous variables for each equation. default = None, but should be specified. is done when sur_stackZ is used.

name_bigq [dictionary] with names of instrumental variables for each equations. default = None, but should be specified. is done when sur_stackZ is used.

name_ds [string] name for the data set

name_w [string] name for the spatial weights

Examples

First import libpysal to load the spatial analysis tools.

```
>>> import libpysal
```

Open data on NCOVR US County Homicides (3085 areas) using libpysal.io.open(). This is the DBF associated with the NAT shapefile. Note that libpysal.io.open() also reads data in CSV format.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("NAT.dbf"),'r')
```

The specification of the model to be estimated can be provided as lists. Each equation should be listed separately. Although not required, in this example we will specify additional endogenous regressors. Equation 1 has HR80 as dependent variable, PS80 and UE80 as exogenous regressors, RD80 as endogenous regressor and FP79 as additional instrument. For equation 2, HR90 is the dependent variable, PS90 and UE90 the exogenous regressors, RD90 as endogenous regressor and FP99 as additional instrument

```
>>> y_var = ['HR80','HR90']
>>> x_var = [['PS80','UE80'],['PS90','UE90']]
>>> yend_var = [['RD80'],['RD90']]
>>> q_var = [['FP79'],['FP89']]
```

The SUR method requires data to be provided as dictionaries. PySAL provides two tools to create these dictionaries from the list of variables: sur_dictxy and sur_dictZ. The tool sur_dictxy can be used to create the dictionaries for Y and X, and sur_dictZ for endogenous variables (yend) and additional instruments (q).

```
>>> bigy,bigX,bigyvars,bigXvars = pysal.spreg.sur_utils.sur_dictxy(db,y_var,x_var)
>>> bigyend,bigyendvars = pysal.spreg.sur_utils.sur_dictZ(db,yend_var)
>>> bigq,bigqvars = pysal.spreg.sur_utils.sur_dictZ(db,q_var)
```

To run a spatial lag model, we need to specify the spatial weights matrix. To do that, we can open an already existing gal file or create a new one. In this example, we will create a new one from NAT.shp and transform it to row-standardized.

```
>>> w = libpysal.weights.Queen.from_shapefile(libpysal.examples.get_path("NAT.shp
↪"))
>>> w.transform='r'
```

We can now run the regression and then have a summary of the output by typing: print(reg.summary)

Alternatively, we can just check the betas and standard errors, asymptotic t and p-value of the parameters:

```
>>> reg = SURlagIV(bigy,bigX,bigyend,bigq,w=w,name_bigy=bigyvars,name_
↪bigX=bigXvars,name_bigyend=bigyendvars,name_bigq=bigqvars,name_ds="NAT",name_w=
↪"nat_queen")
>>> reg.b3SLS
{0: array([[ 6.95472387],
       [ 1.44044301],
       [-0.00771893],
       [ 3.65051153],
       [ 0.00362663]]), 1: array([[ 5.61101925],
       [ 1.38716801],
       [-0.15512029],
       [ 3.1884457 ],
       [ 0.25832185]])}
```

```
>>> reg.tsls_inf
{0: array([[  0.49128435,  14.15620899,   0.        ],
       [  0.11516292,  12.50787151,   0.        ],
       [  0.03204088,  -0.2409087 ,   0.80962588],
       [  0.1876025 ,  19.45875745,   0.        ],
       [  0.05450628,   0.06653605,   0.94695106]]), 1: array([[  0.44969956,  12.
↪47726211,   0.        ],
       [  0.10440241,  13.28674277,   0.        ],
       [  0.04150243,  -3.73761961,   0.00018577],
```

(continues on next page)

(continued from previous page)

```
       [  0.19133145,  16.66451427,   0.        ],
       [  0.04394024,   5.87893596,   0.        ]])}
```

Attributes

w [spatial weights object]

bigy [dictionary] with y values

bigZ [dictionary] with matrix of exogenous and endogenous variables for each equation

bigyend [dictionary] with matrix of endogenous variables for each equation; contains Wy only if no other endogenous specified

bigq [dictionary] with matrix of instrumental variables for each equation; contains WX only if no other endogenous specified

bigZHZH [dictionary] with matrix of cross products Zhat_r'Zhat_s

bigZHy [dictionary] with matrix of cross products Zhat_r'y_end_s

n_eq [int] number of equations

n [int] number of observations in each cross-section

bigK [array] vector with number of explanatory variables (including constant, exogenous and endogenous) for each equation

b2SLS [dictionary] with 2SLS regression coefficients for each equation

tslsE [array] N x n_eq array with OLS residuals for each equation

b3SLS [dictionary] with 3SLS regression coefficients for each equation

varb [array] variance-covariance matrix

sig [array] Sigma matrix of inter-equation error covariances

resids [array] n by n_eq array of residuals

corr [array] inter-equation 3SLS error correlation matrix

tsls_inf [dictionary] with standard error, asymptotic t and p-value, one for each equation

joinrho [tuple] test on joint significance of spatial autoregressive coefficient. tuple with test statistic, degrees of freedom, p-value

surchow [array] list with tuples for Chow test on regression coefficients each tuple contains test value, degrees of freedom, p-value

name_w [string] name for the spatial weights

name_ds [string] name for the data set

name_bigy [dictionary] with name of dependent variable for each equation

name_bigX [dictionary] with names of explanatory variables for each equation

name_bigyend [dictionary] with names of endogenous variables for each equation

name_bigq [dictionary] with names of instrumental variables for each equations

__init__(*self*, *bigy*, *bigX*, *bigyend=None*, *bigq=None*, *w=None*, *regimes=None*, *vm=False*, *regime_lag_sep=False*, *w_lags=1*, *lag_q=True*, *nonspat_diag=True*, *spat_diag=False*, *name_bigy=None*, *name_bigX=None*, *name_bigyend=None*, *name_bigq=None*, *name_ds=None*, *name_w=None*, *name_regimes=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, bigy, bigX[, bigyend, bigq, ...])	Initialize self.

3.18 Diagnostics

Diagnostic tests are useful for identifying model fit, sufficiency, and specification correctness.

spreg.diagnostics.f_stat(reg)	Calculates the f-statistic and associated p-value of the regression.
spreg.diagnostics.t_stat(reg[, z_stat])	Calculates the t-statistics (or z-statistics) and associated p-values.
spreg.diagnostics.r2(reg)	Calculates the R^2 value for the regression.
spreg.diagnostics.ar2(reg)	Calculates the adjusted R^2 value for the regression.
spreg.diagnostics.se_betas(reg)	Calculates the standard error of the regression coefficients.
spreg.diagnostics.log_likelihood(reg)	Calculates the log-likelihood value for the regression.
spreg.diagnostics.akaike(reg)	Calculates the Akaike Information Criterion.
spreg.diagnostics.schwarz(reg)	Calculates the Schwarz Information Criterion.
spreg.diagnostics.condition_index(reg)	Calculates the multicollinearity condition index according to Belsey, Kuh and Welsh (1980) [BKW05].
spreg.diagnostics.jarque_bera(reg)	Jarque-Bera test for normality in the residuals.
spreg.diagnostics.breusch_pagan(reg[, z])	Calculates the Breusch-Pagan test statistic to check for heteroscedasticity.
spreg.diagnostics.white(reg)	Calculates the White test to check for heteroscedasticity.
spreg.diagnostics.koenker_bassett(reg[, z])	Calculates the Koenker-Bassett test statistic to check for heteroscedasticity.
spreg.diagnostics.vif(reg)	Calculates the variance inflation factor for each independent variable.
spreg.diagnostics.likratiotest(reg0, reg1)	Likelihood ratio test statistic [Gre03]
spreg.diagnostics_sp.LMtests(ols, w[, tests])	Lagrange Multiplier tests.
spreg.diagnostics_sp.MoranRes(ols, w[, z])	Moran's I for spatial autocorrelation in residuals from OLS regression
spreg.diagnostics_sp.AKtest(iv, w[, case])	Moran's I test of spatial autocorrelation for IV estimation.
spreg.diagnostics_sur.sur_setp(bigB, varb)	Utility to compute standard error, t and p-value
spreg.diagnostics_sur.sur_lrtest(n, n_eq, ...)	Likelihood Ratio test on off-diagonal elements of Sigma
spreg.diagnostics_sur.sur_lmtest(n, n_eq, sig)	Lagrange Multiplier test on off-diagonal elements of Sigma

Continued on next page

Table 63 – continued from previous page

spreg.diagnostics_sur.lam_setp(lam, vm)	Standard errors, t-test and p-value for lambda in SUR Error ML
spreg.diagnostics_sur.surLMe(n_eq, WS, bigE, sig)	Lagrange Multiplier test on error spatial autocorrelation in SUR
spreg.diagnostics_sur.surLMlag(n_eq, WS, ...)	Lagrange Multiplier test on lag spatial autocorrelation in SUR

3.18.1 spreg.diagnostics.f_stat

spreg.diagnostics.**f_stat**(*reg*)

Calculates the f-statistic and associated p-value of the regression. [Gre03]. (For two stage least squares see f_stat_tsls)

Parameters

reg [regression object] output instance from a regression model

Returns

fs_result [tuple] includes value of F statistic and associated p-value

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the F-statistic for the regression.

```
>>> testresult = diagnostics.f_stat(reg)
```

Print the results tuple, including the statistic and its significance.

```
>>> print("%12.12f"%testresult[0],"%12.12f"%testresult[1])
('28.385629224695', '0.000000009341')
```

3.18.2 spreg.diagnostics.t_stat

spreg.diagnostics.**t_stat**(*reg*, *z_stat=False*)

Calculates the t-statistics (or z-statistics) and associated p-values. [Gre03]

Parameters

reg [regression object] output instance from a regression model

z_stat [boolean] If True run z-stat instead of t-stat

Returns

ts_result [list of tuples] each tuple includes value of t statistic (or z statistic) and associated p-value

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.open(libpysal.examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate t-statistics for the regression coefficients.

```
>>> testresult = diagnostics.t_stat(reg)
```

Print the tuples that contain the t-statistics and their significances.

```
>>> print("%12.12f"%testresult[0][0], "%12.12f"%testresult[0][1], "%12.12f"
↪%testresult[1][0], "%12.12f"%testresult[1][1], "%12.12f"%testresult[2][0], "%12.
↪12f"%testresult[2][1])
('14.490373143689', '0.000000000000', '-4.780496191297', '0.000018289595', '-2.
↪654408642718', '0.010874504910')
```

3.18.3 spreg.diagnostics.r2

spreg.diagnostics.**r2**(*reg*)

Calculates the R^2 value for the regression. [Gre03]

Parameters

reg [regression object] output instance from a regression model

Returns

r2_result [float] value of the coefficient of determination for the regression

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the R^2 value for the regression.

```
>>> testresult = diagnostics.r2(reg)
```

Print the result.

```
>>> print("%1.8f"%testresult)
0.55240404
```

3.18.4 spreg.diagnostics.ar2

spreg.diagnostics.**ar2**(*reg*)

Calculates the adjusted R^2 value for the regression. [Gre03]

Parameters

reg [regression object] output instance from a regression model

Returns

ar2_result [float] value of R^2 adjusted for the number of explanatory variables.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the adjusted R^2 value for the regression. >>> testresult = diagnostics.ar2(reg)

Print the result.

```
>>> print("%1.8f"%testresult)
0.53294335
```

3.18.5 spreg.diagnostics.se_betas

spreg.diagnostics.**se_betas**(*reg*)

Calculates the standard error of the regression coefficients. [Gre03]

Parameters

reg [regression object] output instance from a regression model

Returns

se_result [array] includes standard errors of each coefficient (1 x k)

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the standard errors of the regression coefficients.

```
>>> testresult = diagnostics.se_betas(reg)
```

Print the vector of standard errors.

```
>>> testresult
array([ 4.73548613,  0.33413076,  0.10319868])
```

3.18.6 spreg.diagnostics.log_likelihood

spreg.diagnostics.**log_likelihood**(*reg*)

Calculates the log-likelihood value for the regression. [Gre03]

Parameters

reg [regression object] output instance from a regression model

Returns

ll_result [float] value for the log-likelihood of the regression.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the log-likelihood for the regression.

```
>>> testresult = diagnostics.log_likelihood(reg)
```

Print the result.

```
>>> testresult
-187.3772388121491
```

3.18.7 spreg.diagnostics.akaike

spreg.diagnostics.**akaike**(*reg*)

Calculates the Akaike Information Criterion. [Aka74]

Parameters

reg [regression object] output instance from a regression model

Returns

aic_result [scalar] value for Akaike Information Criterion of the regression.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
```

(continues on next page)

(continued from previous page)

```
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the Akaike Information Criterion (AIC).

```
>>> testresult = diagnostics.akaike(reg)
```

Print the result.

```
>>> testresult
380.7544776242982
```

3.18.8 spreg.diagnostics.schwarz

spreg.diagnostics.**schwarz**(*reg*)

Calculates the Schwarz Information Criterion. [S+78]

Parameters

reg [regression object] output instance from a regression model

Returns

bic_result [scalar] value for Schwarz (Bayesian) Information Criterion of the regression.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the Schwarz Information Criterion.

```
>>> testresult = diagnostics.schwarz(reg)
```

Print the results.

```
>>> testresult
386.42993851863008
```

3.18.9 spreg.diagnostics.condition_index

spreg.diagnostics.**condition_index**(*reg*)

Calculates the multicollinearity condition index according to Belsey, Kuh and Welsh (1980) [BKW05].

Parameters

reg [regression object] output instance from a regression model

Returns

ci_result [float] scalar value for the multicollinearity condition index.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the condition index to check for multicollinearity.

```
>>> testresult = diagnostics.condition_index(reg)
```

Print the result.

```
>>> print("%1.3f"%testresult)
6.542
```

3.18.10 spreg.diagnostics.jarque_bera

spreg.diagnostics.**jarque_bera**(*reg*)

Jarque-Bera test for normality in the residuals. [JB80]

Parameters

reg [regression object] output instance from a regression model

Returns

jb_result [dictionary] contains the statistic (jb) for the Jarque-Bera test and the associated p-value (p-value)

df [integer] degrees of freedom for the test (always 2)

jb [float] value of the test statistic

pvalue [float] p-value associated with the statistic (chi^2 distributed with 2 df)

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"), "r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the Jarque-Bera test for normality of residuals.

```
>>> testresult = diagnostics.jarque_bera(reg)
```

Print the degrees of freedom for the test.

```
>>> testresult['df']
2
```

Print the test statistic.

```
>>> print("%1.3f"%testresult['jb'])
1.836
```

Print the associated p-value.

```
>>> print("%1.4f"%testresult['pvalue'])
0.3994
```

3.18.11 spreg.diagnostics.breusch_pagan

spreg.diagnostics.**breusch_pagan**(*reg*, *z=None*)

Calculates the Breusch-Pagan test statistic to check for heteroscedasticity. [BP79]

Parameters

reg [regression object] output instance from a regression model

z [array] optional input for specifying an alternative set of variables (Z) to explain the observed variance. By default this is a matrix of the squared explanatory variables (X**2) with a constant added to the first column if not already present. In the default case, the explanatory variables are squared to eliminate negative values.

Returns

bp_result [dictionary] contains the statistic (bp) for the test and the associated p-value (p-value)

bp [float] scalar value for the Breusch-Pagan test statistic

df [integer] degrees of freedom associated with the test (k)

pvalue [float] p-value associated with the statistic (chi^2 distributed with k df)

Notes

x attribute in the reg object must have a constant term included. This is standard for spreg.OLS so no testing done to confirm constant.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"), "r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the Breusch-Pagan test for heteroscedasticity.

```
>>> testresult = diagnostics.breusch_pagan(reg)
```

Print the degrees of freedom for the test.

```
>>> testresult['df']
2
```

Print the test statistic.

```
>>> print("%1.3f"%testresult['bp'])
7.900
```

Print the associated p-value.

```
>>> print("%1.4f"%testresult['pvalue'])
0.0193
```

3.18.12 spreg.diagnostics.white

spreg.diagnostics.**white**(*reg*)

Calculates the White test to check for heteroscedasticity. [Whi80]

Parameters

reg [regression object] output instance from a regression model

Returns

white_result [dictionary] contains the statistic (white), degrees of freedom (df) and the associated p-value (pvalue) for the White test.

white [float] scalar value for the White test statistic.

df [integer] degrees of freedom associated with the test

pvalue [float] p-value associated with the statistic (chi^2 distributed with k df)

Notes

x attribute in the reg object must have a constant term included. This is standard for spreg.OLS so no testing done to confirm constant.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the White test for heteroscedasticity.

```
>>> testresult = diagnostics.white(reg)
```

Print the degrees of freedom for the test.

```
>>> print testresult['df']
5
```

Print the test statistic.

```
>>> print("%1.3f"%testresult['wh'])
19.946
```

Print the associated p-value.

```
>>> print("%1.4f"%testresult['pvalue'])
0.0013
```

3.18.13 spreg.diagnostics.koenker_bassett

spreg.diagnostics.**koenker_bassett**(*reg*, *z=None*)

Calculates the Koenker-Bassett test statistic to check for heteroscedasticity. [KBJ82][Gre03]

Parameters

reg [regression output] output from an instance of a regression class

z [array] optional input for specifying an alternative set of variables (Z) to explain the observed variance. By default this is a matrix of the squared explanatory variables (X**2) with a constant added to the first column if not already present. In the default case, the explanatory variables are squared to eliminate negative values.

Returns

kb_result [dictionary] contains the statistic (kb), degrees of freedom (df) and the associated p-value (pvalue) for the test.

kb [float] scalar value for the Koenker-Bassett test statistic.

df [integer] degrees of freedom associated with the test

pvalue [float] p-value associated with the statistic (chi^2 distributed)

Notes

x attribute in the reg object must have a constant term included. This is standard for spreg.OLS so no testing done to confirm constant.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the Koenker-Bassett test for heteroscedasticity.

```
>>> testresult = diagnostics.koenker_bassett(reg)
```

Print the degrees of freedom for the test.

```
>>> testresult['df']
2
```

Print the test statistic.

```
>>> print("%1.3f"%testresult['kb'])
5.694
```

Print the associated p-value.

```
>>> print("%1.4f"%testresult['pvalue'])
0.0580
```

3.18.14 spreg.diagnostics.vif

spreg.diagnostics.**vif**(*reg*)

Calculates the variance inflation factor for each independent variable. For the ease of indexing the results, the constant is currently included. This should be omitted when reporting the results to the output text. [Gre03]

Parameters

reg [regression object] output instance from a regression model

Returns

vif_result [list of tuples] each tuple includes the vif and the tolerance, the order of the variables corresponds to their order in the reg.x matrix

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import diagnostics
>>> from ols import OLS
```

Read the DBF associated with the Columbus data.

```
>>> db = libpysal.io.open(examples.get_path("columbus.dbf"),"r")
```

Create the dependent variable vector.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Create the matrix of independent variables.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X.append(db.by_col("HOVAL"))
>>> X = np.array(X).T
```

Run an OLS regression.

```
>>> reg = OLS(y,X)
```

Calculate the variance inflation factor (VIF). >>> testresult = diagnostics.vif(reg)

Select the tuple for the income variable.

```
>>> incvif = testresult[1]
```

Print the VIF for income.

```
>>> print("%12.12f"%incvif[0])
1.333117497189
```

Print the tolerance for income.

```
>>> print("%12.12f"%incvif[1])
0.750121427487
```

Repeat for the home value variable.

```
>>> hovalvif = testresult[2]
>>> print("%12.12f"%hovalvif[0])
1.333117497189
>>> print("%12.12f"%hovalvif[1])
0.750121427487
```

3.18.15 spreg.diagnostics.likratiotest

spreg.diagnostics.**likratiotest**(*reg0*, *reg1*)

Likelihood ratio test statistic [Gre03]

Parameters

reg0 [regression object] for constrained model (H0)

reg1 [regression object] for unconstrained model (H1)

Returns

likratio [dictionary] contains the statistic (likr), the degrees of freedom (df) and the p-value (pvalue)

likr [float] likelihood ratio statistic

df [integer] degrees of freedom

p-value [float] p-value

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from libpysal import examples
>>> import scipy.stats as stats
>>> import spreg.ml_lag as lag
```

Use the baltim sample data set

```
>>> db = libpysal.io.open(examples.get_path("baltim.dbf"),'r')
>>> y_name = "PRICE"
>>> y = np.array(db.by_col(y_name)).T
>>> y.shape = (len(y),1)
>>> x_names = ["NROOM","NBATH","PATIO","FIREPL","AC","GAR","AGE","LOTSZ","SQFT"]
>>> x = np.array([db.by_col(var) for var in x_names]).T
>>> ww = ps.open(ps.examples.get_path("baltim_q.gal"))
>>> w = ww.read()
>>> ww.close()
>>> w.transform = 'r'
```

OLS regression

```
>>> ols1 = ps.spreg.OLS(y,x)
```

ML Lag regression

```
>>> mllag1 = lag.ML_Lag(y,x,w)
```

```
>>> lr = likratiotest(ols1,mllag1)
```

```
>>> print "Likelihood Ratio Test: {0:.4f}       df: {1}        p-value: {2:.4f}".
↪format(lr["likr"],lr["df"],lr["p-value"])
Likelihood Ratio Test: 44.5721       df: 1        p-value: 0.0000
```

3.18.16 spreg.diagnostics_sp.LMtests

class spreg.diagnostics_sp.**LMtests**(*ols, w, tests=['all']*)

Lagrange Multiplier tests. Implemented as presented in [ABFY96]

Parameters

lme [tuple] (Only if 'lme' or 'all' was in tests). Pair of statistic and p-value for the LM error test.

lml [tuple] (Only if 'lml' or 'all' was in tests). Pair of statistic and p-value for the LM lag test.

rlme [tuple] (Only if 'rlme' or 'all' was in tests). Pair of statistic and p-value for the Robust LM error test.

rlml [tuple] (Only if 'rlml' or 'all' was in tests). Pair of statistic and p-value for the Robust LM lag test.

sarma [tuple] (Only if 'rlml' or 'all' was in tests). Pair of statistic and p-value for the SARMA test.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from ols import OLS
```

Open the csv file to access the data for analysis

```
>>> csv = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Pull out from the csv the files we need ('HOVAL' as dependent as well as 'INC' and 'CRIME' as independent) and directly transform them into nx1 and nx2 arrays, respectively

```
>>> y = np.array([csv.by_col('HOVAL')]).T
>>> x = np.array([csv.by_col('INC'), csv.by_col('CRIME')]).T
```

Create the weights object from existing .gal file

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

Row-standardize the weight object (not required although desirable in some cases)

```
>>> w.transform='r'
```

Run an OLS regression

```
>>> ols = OLS(y, x)
```

Run all the LM tests in the residuals. These diagnostics test for the presence of remaining spatial autocorrelation in the residuals of an OLS model and give indication about the type of spatial model. There are five types: presence of a spatial lag model (simple and robust version), presence of a spatial error model (simple and robust version) and joint presence of both a spatial lag as well as a spatial error model.

```
>>> lms = spreg.diagnostics_sp.LMtests(ols, w)
```

LM error test:

```
>>> print round(lms.lme[0],4), round(lms.lme[1],4)
3.0971 0.0784
```

LM lag test:

```
>>> print round(lms.lml[0],4), round(lms.lml[1],4)
0.9816 0.3218
```

Robust LM error test:

```
>>> print round(lms.rlme[0],4), round(lms.rlme[1],4)
3.2092 0.0732
```

Robust LM lag test:

```
>>> print round(lms.rlml[0],4), round(lms.rlml[1],4)
1.0936 0.2957
```

LM SARMA test:

```
>>> print round(lms.sarma[0],4), round(lms.sarma[1],4)
4.1907 0.123
```

Attributes

ols [OLS] OLS regression object

w [W] Spatial weights instance

tests [list] Lists of strings with the tests desired to be performed. Values may be:

- 'all': runs all the options (default)
- 'lme': LM error test
- 'rlme': Robust LM error test
- 'lml' : LM lag test
- 'rlml': Robust LM lag test

__init__(*self, ols, w, tests=['all']*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, ols, w[, tests])	Initialize self.

3.18.17 spreg.diagnostics_sp.MoranRes

class spreg.diagnostics_sp.**MoranRes**(*ols*, *w*, *z=False*)
Moran's I for spatial autocorrelation in residuals from OLS regression

Parameters

ols [OLS] OLS regression object

w [W] Spatial weights instance

z [boolean] If set to True computes attributes eI, vI and zI. Due to computational burden of vI, defaults to False.

Examples

```
>>> import numpy as np
>>> import libpysal
>>> from ols import OLS
```

Open the csv file to access the data for analysis

```
>>> csv = libpysal.io.open(libpysal.examples.get_path('columbus.dbf'),'r')
```

Pull out from the csv the files we need ('HOVAL' as dependent as well as 'INC' and 'CRIME' as independent) and directly transform them into nx1 and nx2 arrays, respectively

```
>>> y = np.array([csv.by_col('HOVAL')]).T
>>> x = np.array([csv.by_col('INC'), csv.by_col('CRIME')]).T
```

Create the weights object from existing .gal file

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

Row-standardize the weight object (not required although desirable in some cases)

```
>>> w.transform='r'
```

Run an OLS regression

```
>>> ols = OLS(y, x)
```

Run Moran's I test for residual spatial autocorrelation in an OLS model. This computes the traditional statistic applying a correction in the expectation and variance to account for the fact it comes from residuals instead of an independent variable

```
>>> m = spreg.diagnostics_sp.MoranRes(ols, w, z=True)
```

Value of the Moran's I statistic:

```
>>> print round(m.I,4)
0.1713
```

Value of the Moran's I expectation:

```
>>> print round(m.eI,4)
-0.0345
```

Value of the Moran's I variance:

```
>>> print round(m.vI,4)
0.0081
```

Value of the Moran's I standardized value. This is distributed as a standard Normal(0, 1)

```
>>> print round(m.zI,4)
2.2827
```

P-value of the standardized Moran's I value (z):

```
>>> print round(m.p_norm,4)
0.0224
```

Attributes

I [float] Moran's I statistic

eI [float] Moran's I expectation

vI [float] Moran's I variance

zI [float] Moran's I standardized value

__init__(*self*, *ols*, *w*, *z=False*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, ols, w[, z])	Initialize self.

3.18.18 spreg.diagnostics_sp.AKtest

class spreg.diagnostics_sp.**AKtest**(*iv*, *w*, *case='nosp'*)

Moran's I test of spatial autocorrelation for IV estimation. Implemented following the original reference [AK97]

Parameters

iv [TSLS] Regression object from TSLS class

w [W] Spatial weights instance

case [string] Flag for special cases (default to 'nosp'):

- 'nosp': Only NO spatial end. reg.
- 'gen': General case (spatial lag + end. reg.)

Examples

We first need to import the needed modules. Numpy is needed to convert the data we read into arrays that `spreg` understands and `pysal` to perform all the analysis. The TSLS is required to run the model on which we will perform the tests.

```
>>> import numpy as np
>>> import libpysal
>>> from twosls import TSLS
>>> from twosls_sp import GM_Lag
```

Open data on Columbus neighborhood crime (49 areas) using libpysal.io.open(). This is the DBF associated with the Columbus shapefile. Note that libpysal.io.open() also reads data in CSV format; since the actual class requires data to be passed in as numpy arrays, the user can read their data in using any method.

```
>>> db = libpysal.io.open(libpysal.examples.get_path("columbus.dbf"),'r')
```

Before being able to apply the diagnostics, we have to run a model and, for that, we need the input variables. Extract the CRIME column (crime rates) from the DBF file and make it the dependent variable for the regression. Note that PySAL requires this to be an numpy array of shape (n, 1) as opposed to the also common shape of (n,) that other packages accept.

```
>>> y = np.array(db.by_col("CRIME"))
>>> y = np.reshape(y, (49,1))
```

Extract INC (income) vector from the DBF to be used as independent variables in the regression. Note that PySAL requires this to be an nxj numpy array, where j is the number of independent variables (not including a constant). By default this model adds a vector of ones to the independent variables passed in, but this can be overridden by passing constant=False.

```
>>> X = []
>>> X.append(db.by_col("INC"))
>>> X = np.array(X).T
```

In this case, we consider HOVAL (home value) as an endogenous regressor, so we acknowledge that by reading it in a different category.

```
>>> yd = []
>>> yd.append(db.by_col("HOVAL"))
>>> yd = np.array(yd).T
```

In order to properly account for the endogeneity, we have to pass in the instruments. Let us consider DISCBD (distance to the CBD) is a good one:

```
>>> q = []
>>> q.append(db.by_col("DISCBD"))
>>> q = np.array(q).T
```

Now we are good to run the model. It is an easy one line task.

```
>>> reg = TSLS(y, X, yd, q=q)
```

Now we are concerned with whether our non-spatial model presents spatial autocorrelation in the residuals. To assess this possibility, we can run the Anselin-Kelejian test, which is a version of the classical LM error test adapted for the case of residuals from an instrumental variables (IV) regression. First we need an extra object, the weights matrix, which includes the spatial configuration of the observations into the error component of the model. To do that, we can open an already existing gal file or create a new one. In this case, we will create one from `columbus.shp`.

```
>>> w = libpysal.weights.Rook.from_shapefile(libpysal.examples.get_path("columbus.
↪shp"))
```

Unless there is a good reason not to do it, the weights have to be row-standardized so every row of the matrix sums to one. Among other things, this allows to interpret the spatial lag of a variable as the average value of the neighboring observations. In PySAL, this can be easily performed in the following way:

```
>>> w.transform = 'r'
```

We are good to run the test. It is a very simple task:

```
>>> ak = AKtest(reg, w)
```

And explore the information obtained:

```
>>> print('AK test: %f      P-value: %f'%(ak.ak, ak.p))
AK test: 4.642895      P-value: 0.031182
```

The test also accomodates the case when the residuals come from an IV regression that includes a spatial lag of the dependent variable. The only requirement needed is to modify the `case` parameter when we call `AKtest`. First, let us run a spatial lag model:

```
>>> reg_lag = GM_Lag(y, X, yd, q=q, w=w)
```

And now we can run the AK test and obtain similar information as in the non-spatial model.

```
>>> ak_sp = AKtest(reg, w, case='gen')
>>> print('AK test: %f      P-value: %f'%(ak_sp.ak, ak_sp.p))
AK test: 1.157593      P-value: 0.281965
```

Attributes

mi [float] Moran's I statistic for IV residuals

ak [float] Square of corrected Moran's I for residuals $ak = \frac{NimesI^*}{\phi^2}$. Note: if case='nosp' then it simplifies to the LMerror

p [float] P-value of the test

__init__(*self*, *iv*, *w*, *case='nosp'*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self, iv, w[, case])	Initialize self.

3.18.19 spreg.diagnostics_sur.sur_setp

spreg.diagnostics_sur.**sur_setp**(*bigB*, *varb*)
Utility to compute standard error, t and p-value

Parameters

bigB [dictionary] of regression coefficient estimates, one vector by equation

varb [array] variance-covariance matrix of coefficients

Returns

surinfdict [dictionary] with standard error, t-value, and p-value array, one for each equation

3.18.20 spreg.diagnostics_sur.sur_lrtest

spreg.diagnostics_sur.**sur_lrtest**(*n*, *n_eq*, *ldetS0*, *ldetS1*)
Likelihood Ratio test on off-diagonal elements of Sigma

Parameters

n [int] cross-sectional dimension (number of observations for an equation)

n_eq [int] number of equations

ldetS0 [float] log determinant of Sigma for OLS case

ldetS1 [float] log determinant of Sigma for SUR case (should be iterated)

Returns

(lrtest,M,pvalue) [tuple] with value of test statistic (lrtest), degrees of freedom (M, as an integer) p-value

3.18.21 spreg.diagnostics_sur.sur_lmtest

spreg.diagnostics_sur.**sur_lmtest**(*n*, *n_eq*, *sig*)
Lagrange Multiplier test on off-diagonal elements of Sigma

Parameters

n [int] cross-sectional dimension (number of observations for an equation)

n_eq [int] number of equations

sig [array] inter-equation covariance matrix for null model (OLS)

Returns

(lmtest,M,pvalue) [tuple] with value of test statistic (lmtest), degrees of freedom (M, as an integer) p-value

3.18.22 spreg.diagnostics_sur.lam_setp

spreg.diagnostics_sur.**lam_setp**(*lam*, *vm*)

Standard errors, t-test and p-value for lambda in SUR Error ML

Parameters

lam [array] n_eq x 1 array with ML estimates for spatial error autoregressive coefficient

vm [array] n_eq x n_eq subset of variance-covariance matrix for lambda and Sigma in SUR Error ML (needs to be subset from full vm)

Returns

: tuple with arrays for standard error, t-value and p-value (each element in the tuple is an n_eq x 1 array)

3.18.23 spreg.diagnostics_sur.surLMe

spreg.diagnostics_sur.**surLMe**(*n_eq*, *WS*, *bigE*, *sig*)

Lagrange Multiplier test on error spatial autocorrelation in SUR

Parameters

n_eq [int] number of equations

WS [array] spatial weights matrix in sparse form

bigE [array] n x n_eq matrix of residuals by equation

sig [array] cross-equation error covariance matrix

Returns

(LMe,n_eq,pvalue) [tuple] with value of statistic (LMe), degrees of freedom (n_eq) and p-value

3.18.24 spreg.diagnostics_sur.surLMlag

spreg.diagnostics_sur.**surLMlag**(*n_eq*, *WS*, *bigy*, *bigX*, *bigE*, *bigYP*, *sig*, *varb*)

Lagrange Multiplier test on lag spatial autocorrelation in SUR

Parameters

n_eq [int] number of equations

WS [spatial weights matrix in sparse form]

bigy [dictionary] with y values

bigX [dictionary] with X values

bigE [array] n x n_eq matrix of residuals by equation

bigYP [array] n x n_eq matrix of predicted values by equation

sig [array] cross-equation error covariance matrix

varb [array] variance-covariance matrix for b coefficients (inverse of Ibb)

Returns

(LMlag,n_eq,pvalue) [tuple] with value of statistic (LMlag), degrees of freedom (n_eq) and p-value

CHAPTER

FOUR

REFERENCES

BIBLIOGRAPHY

[Aka74] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[Ans88] Luc Anselin. *Spatial Econometrics: Methods and Models*. Kluwer, Dordrecht, 1988.

[Ans11] Luc Anselin. GMM estimation of spatial error autocorrelation with and without heteroskedasticity. Technical Report, GeoDa Center for Geospatial Analysis and Computation, 2011.

[ABFY96] Luc Anselin, Anil K Bera, Raymond Florax, and Mann J Yoon. Simple diagnostic tests for spatial dependence. *Regional science and urban economics*, 26(1):77–104, 1996.

[AK97] Luc Anselin and Harry H Kelejian. Testing for spatial error autocorrelation in the presence of endogenous regressors. *International Regional Science Review*, 20(1-2):153–182, 1997.

[ADKP10] Irani Arraiz, David M. Drukker, Harry H. Kelejian, and Ingmar R. Prucha. A spatial Cliff-Ord-type model with heteroskedastic innovations: Small and large sample results. *Journal of Regional Science*, 50(2):592–614, 2010. doi:10.1111/j.1467-9787.2009.00618.x.

[BKW05] David A Belsley, Edwin Kuh, and Roy E Welsch. *Regression diagnostics: Identifying influential data and sources of collinearity*. Volume 571. John Wiley & Sons, 2005.

[BP79] Trevor S Breusch and Adrian R Pagan. A simple test for heteroscedasticity and random coefficient variation. *Econometrica: Journal of the Econometric Society*, pages 1287–1294, 1979.

[DEP13] David M Drukker, Peter Egger, and Ingmar R Prucha. On two-step estimation of a spatial autoregressive model with autoregressive disturbances and endogenous regressors. *Econometric Reviews*, 32(5-6):686–733, 2013.

[DPR13] David M. Drukker, Ingmar R. Prucha, and Rafal Raciborski. A command for estimating spatial-autoregressive models with spatial-autoregressive disturbances and additional endogenous variables. *The Stata Journal*, 13(2):287–301, 2013. URL: https://journals.sagepub.com/doi/abs/10.1177/1536867X1301300203.

[Gre03] William H Greene. *Econometric analysis*. Pearson Education India, 2003.

[JB80] Carlos M Jarque and Anil K Bera. Efficient tests for normality, homoscedasticity and serial independence of regression residuals. *Economics letters*, 6(3):255–259, 1980.

[KP99] H H Kelejian and I R Prucha. A generalized moments estimator for the autoregressive parameter in a spatial model. *Int. Econ. Rev.*, 40:509–534, 1999.

[KP98] Harry H Kelejian and Ingmar R Prucha. A generalized spatial two-stage least squares procedure for estimating a spatial autoregressive model with autoregressive disturbances. *J. Real Estate Fin. Econ.*, 17(1):99–121, 1998.

[KBJ82] Roger Koenker and Gilbert Bassett Jr. Robust tests for heteroscedasticity based on regression quantiles. *Econometrica: Journal of the Econometric Society*, pages 43–61, 1982.

[S+78] Gideon Schwarz and others. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[Whi80] Halbert White. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica: Journal of the Econometric Society*, pages 817–838, 1980.

INDEX

Symbols

A

B

C

F

G