# `pysal.spreg`: "spatial regression like a Pro"

**Author: Daniel Arribas-Bel (May, 2012) <darribas@asu.edu>**
In this session, we will walk through most of the functionality that `pysal` has to offer when it comes to spatial regression. We will repeat the exercise we have previously seen in GeoDaSpace using PySAL entirely. In the process, we will discover some of the details that GeoDaSpace misses in order to have an intuitive interface and, by showing some extra tricks, will hopefully convince you of the extra power and flexibility that the command line offers at the cost of a slightly steeper learning curve. The three parts in which we will split the session are the following:

- Load up the data into Python
- Replication of the regression analysis in GeoDaSpace
- Some extra tricks you can access if you use `pysal.spreg`

## Loading your data into Python

This is the very first step you need to go through in order to perform any data analysis with Python. It is the equivalent to load up a dbf of a csv file in GeoDaSpace, onlyt that instead of clicking on a GUI, we will run commands. Before anything, we have to bring `pysal` to our sesion. To make it shorter to type, we will import it with the alias `ps`. Because ultimately, `pysal` takes NumPy arrays, we will also import numpy, shortening with the common `np`:

In [1]:
```
import pysal as ps
import numpy as np
```

Let us imagine that our data are in a `csv` file under the path `../data_workshop/phx.csv`. Make sure that you have navigated to the right folder or pass the full path of the file (alternatively, you can use relative paths as well, as we are doing in this example). We will load them in memory by calling the command open in `pysal`, which mirrors the general file I/O in Python:

In [2]:
```
db = ps.open('../workshop_data/phx.csv')
```

This creates the object `db` which contains all our data. We can explore it a little bit to get a feel of what it contains:

In [3]:
```
len(db) #How many observations
```

Out[3]:
```
985
```

In [4]:
```
db.header #Names of the columns
```

Out[4]:
```
['id',
 'ALAND10',
 'AWATER10',
 'GEOID10',
 'NAMELSAD10',
 'pop_dens',
 'inc',
 'inc_error',
 'pct_error',
 'renter_rt',
 'pop',
 'white_rt',
 'black_rt',
 'hisp_rt',
 'fem_nh_rt',
```

```
 'vac_hsu_rt',
 'hsu',
 'l_pct_err']
```

In [5]:
```
db[0, :] #First row
```

Out[5]:
```
[['g04021000803',
  2197.12461,
  527824,
  4021000803,
  'Census Tract 8.03',
  5.52949975832,
  4462.0,
  766.0,
  17.1671896011,
  0.625565890197,
  100.0,
  54.9839492962,
  7.78664910692,
  27.0639558811,
  0.304551814964,
  17.8477690289,
  381.0,
  2.84299998074]]
```

In [6]:
```
db[0:5, 0] #First six elements of the first column
```

Out[6]:
```
['g04021000803',
 'g04021001701',
 'g04021001403',
 'g04021001406',
 'g04021001303']
```

If we want to extract a full column, `pysal` has a very handy utility for the task, which will return a list with the elements of the columns. We can also get some basic statistics using NumPy:

In [7]:
```
pdens = db.by_col('pop_dens')
pdens[0:5] #First six elements of the population density column
```

Out[7]:
```
[5.52949975832, 1.10832310582, 170.575010113, 27.2587305868, 3.04072345619]
```

In [8]:
```
min(pdens) #Minimum value
```

Out[8]:
```
0.00471158964005
```

In [9]:
```
max(pdens) #Maximum value
```

Out[9]:
```
904.675278538
```

In [10]:
```
np.mean(pdens) #Average value
```

Out[10]:
```
163.48580448940245
```

In [11]:
```
np.var(pdens) #Variance
```

Out[11]:
```
13592.733720318472
```

To be able to use `spreg`, we need to convert the data we will be using in our models into NumPy arrays, which are efficient data structures that allow very good performance for matrix operations. Also, these arrays need to be 2D so we will reshape them when necessary. Let us use the log of the `pct_error` as the dependent variable (y) and the same list of variables as in the GeoDaSpace example for the regressors:

In [12]:
```
y = np.array(db.by_col('l_pct_err')).reshape((len(db), 1))
x_names = ['hsu', 'pop_dens', 'white_rt', 'black_rt', 'hisp_rt', \
           'fem_nh_rt', 'renter_rt', 'vac_hsu_rt']
x = np.array([db.by_col(var) for var in x_names]).T
```

The last bit of data house-keeping we need to perform is to load the spatial weights. Although we will use more in the last section, let us begin with the `knn` weights with six neighbors (note you might get a warning because it does not find IDs matching in the shapefile in the folder). We also row-standardize the matrix so every row sums to one and the spatial lag can be interpreted as the average value of the neighbors.

In [13]:
```
w = ps.open('../workshop_data/phx_knn06.gwt').read()
w.transform = 'R'
```

```
/Users/dani/code/pysal/pysal/core/IOHandlers/gwt.py:141: RuntimeWarning: DBF
relating to GWT was not found, proceeding with unordered string ids.
  warn("DBF relating to GWT was not found, proceeding with unordered string ids.",
RuntimeWarning)
```

## Replication of the regression analysis in GeoDaSpace

### Non-spatial model

We are all set to start the regression analysis. As a benchmark, we will begin with a base model using the command `OLS`. We will also pass in a weights object and set the flag so we obtain also spatial diagnostics about the residuals.

In [14]:
```
from pysal.spreg import OLS
model = OLS(y, x, w=w, name_x=x_names, spat_diag=True)
print model.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: ORDINARY LEAST SQUARES ESTIMATION
----------------------------------------------------
Data set            :     unknown
Weights matrix      :     unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :           9
S.D. dependent var  :      0.3505  Degrees of Freedom    :         976

R-squared           :    0.322025
Adjusted R-squared  :      0.3165
Sum squared residual:      81.944  F-statistic           :     57.9476
Sigma-square        :       0.084  Prob(F-statistic)     :2.844783e-77
S.E. of regression  :       0.290  Log likelihood        :    -173.002
Sigma-square ML     :       0.083  Akaike info criterion :     364.003
S.E of regression ML:      0.2884  Schwarz criterion     :     408.037

----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     t-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       3.2602807       0.1246450      26.1565321    1.039377e-114
```

```
          hsu      -0.0002086       0.0000134     -15.6037433    3.581137e-49
     pop_dens      -0.0004243       0.0000988      -4.2944144    1.926527e-05
     white_rt      -0.0034062       0.0012541      -2.7160276     0.006723808
     black_rt       0.0038674       0.0028898       1.3382790       0.1811172
      hisp_rt       0.0003542       0.0007873       0.4499107       0.6528747
    fem_nh_rt      -0.0236822       0.0074328      -3.1861867      0.00148738
    renter_rt       0.0079963       0.0013002       6.1501153    1.127742e-09
   vac_hsu_rt       0.0088448       0.0012917       6.8475856    1.326764e-11
----------------------------------------------------------------------------

REGRESSION DIAGNOSTICS
MULTICOLLINEARITY CONDITION NUMBER   45.873984
TEST ON NORMALITY OF ERRORS
TEST                      DF           VALUE              PROB
Jarque-Bera                2           75.866541          0.0000000

DIAGNOSTICS FOR HETEROSKEDASTICITY
RANDOM COEFFICIENTS
TEST                      DF           VALUE              PROB
Breusch-Pagan test         8           31.433268          0.0001176
Koenker-Bassett test       8           20.528622          0.0085108

SPECIFICATION ROBUST TEST
Not computed due to multicollinearity.

DIAGNOSTICS FOR SPATIAL DEPENDENCE
TEST                             MI/DF       VALUE           PROB
Lagrange Multiplier (lag)          1        24.978573        0.0000006
Robust LM (lag)                    1         8.479697        0.0035913
Lagrange Multiplier (error)        1        16.931274        0.0000388
Robust LM (error)                  1         0.432398        0.5108145
Lagrange Multiplier (SARMA)        2        25.410972        0.0000030

========================= END OF REPORT ==============================
```

At first sight, it is not too bad: we are explaining over 30% of the variation in the dependent variable and the F test reveals some significance of the coefficients overall. However, before we consider the coefficients, it is important to check that the model we have just run is correct and that OLS is a good estimation procedure for this dataset. There are several parts in this output report that should rise a red flag when we look at them.

In first place, the `SPECIFICATION ROBUST TEST` (White test) section has not been computed due to multicolinearity. If we look at the multicolinearity condition number we see it is almost 46. This is high, in fact beyond 30 which is the limit at which `pysal.spreg` stops computing the White test. However, it is not terribly high (although the rule of thumb is 30, below 100 may still be feasible) so for the time being, we will let it be.

The assumption of normality of the residuals is one of the most important ones in linear regression when the size of your sample is limited. However, as `n` grows, this is less relevant due to the central limit theorem (CLM). Since we have almost 1000 observations, we assume we can rely on the CLT.

Two other problems still remain, heteroskedasticity and spatial dependence, and it is on them that we will focus the most. Both diagnostics against heteroskedasticity clearly point to the presence of the problem. One traditional way to tackle this issue is by estimating a robust VC matrix following White's procedure. Let us do just that before we get our hands dirty with any spatial issue. Since we have already seen them, let us skip the computation of the diagnostics in this run:

In [15]:
```
from pysal.spreg import OLS
model = OLS(y, x, w=w, name_x=x_names, spat_diag=False, nonspat_diag=False, robust='w
print model.summary
```

```
REGRESSION
----------
```

```
SUMMARY OF OUTPUT: ORDINARY LEAST SQUARES ESTIMATION
-----------------------------------------------------
Data set            :     unknown
Weights matrix      :     unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables    :           9
S.D. dependent var  :      0.3505  Degrees of Freedom     :         976

R-squared           :    0.322025
Adjusted R-squared  :      0.3165

White Standard Errors
----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     t-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       3.2602807       0.1307919      24.9272298     1.5903e-106
         hsu      -0.0002086       0.0000157     -13.2902064    3.621625e-37
    pop_dens      -0.0004243       0.0001018      -4.1666504    3.365117e-05
    white_rt      -0.0034062       0.0012197      -2.7926344     0.005330259
    black_rt       0.0038674       0.0025722       1.5035688       0.1330159
     hisp_rt       0.0003542       0.0006928       0.5113044       0.6092535
   fem_nh_rt      -0.0236822       0.0097645      -2.4253461      0.01547431
   renter_rt       0.0079963       0.0015382       5.1983427    2.448857e-07
  vac_hsu_rt       0.0088448       0.0014780       5.9843749    3.047318e-09
----------------------------------------------------------------------------
========================= END OF REPORT ==============================
```

We can see some changes in the standard errors of the coefficients, and one variable (`fem_nh_rt`) becomes insignificant at the 1% level.

## Spatial diagnostics

At this point, we are ready to consider the issue of spatial spatial dependence and explicitly incorporate space into our model. In order to better focus on them, we will print the part of the regression output that concerns only to the spatial diagnostics. This is a somewhat hidden feature, but it can prove very useful in some contexts, for instance when you are considering several types of weights on the same model. Mind that this trick requires you to re-run the OLS every time so it is not very efficient in terms of speed up (for that, see below in the extra tricks section). However, it comes in very handy to inspect the results.

In [16]:
```
from pysal.spreg.user_output import summary_spat_diag
model = OLS(y, x, w=w, name_x=x_names, spat_diag=True)
spd = summary_spat_diag(model, None, None)
print spd
```

```
DIAGNOSTICS FOR SPATIAL DEPENDENCE
TEST                           MI/DF       VALUE           PROB
Lagrange Multiplier (lag)        1       24.978573        0.0000006
Robust LM (lag)                  1        8.479697        0.0035913
Lagrange Multiplier (error)      1       16.931274        0.0000388
Robust LM (error)                1        0.432398        0.5108145
Lagrange Multiplier (SARMA)      2       25.410972        0.0000030
```

The first thing we can observe is that the non-robust versions of the LM tests all point to severe presence of spatial autocorrelation. The problem of having both tests (error and lag) rejecting the null is that they can be misleading at the presence of the other specification. In other words, if the true DGP contains say an error structure, the result of the LM-Lag is affected and viceversa. For that reason, we have to turn to the robust versions of the LM which, as they name reads, are robust to the presence of the other model. At this point, things become clearer: while the robust version of the LM-Lag is still significant, the error counterpart is not.

## Spatial error models

Before we move into the right direction, let us assume that we did not look at the robust versions and decided to try first with one of the simplest spatial specifications: the error model. This is very easy to run in PySAL thanks to the method `GM_Error`, which implements the traditional Kelejan and Prucha 1998/99 papers:

In [17]:
```
from pysal.spreg import GM_Error
error_kp98 = GM_Error(y, x, w, name_x=x_names)
print error_kp98.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIALLY WEIGHTED LEAST SQUARES ESTIMATION
--------------------------------------------------------------
Data set            :     unknown
Weights matrix      :     unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :           9
S.D. dependent var  :      0.3505  Degrees of Freedom    :         976

Pseudo R-squared    :    0.321335

----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       3.2542359       0.1265294      25.7192144    7.127017e-146
         hsu      -0.0002062       0.0000133     -15.4802880    4.713582e-54
    pop_dens      -0.0003921       0.0001024      -3.8280762    0.0001291488
    white_rt      -0.0032943       0.0012636      -2.6069925     0.009134137
    black_rt       0.0042299       0.0030457       1.3887930       0.1648957
     hisp_rt       0.0004473       0.0008225       0.5437729       0.5865978
   fem_nh_rt      -0.0223199       0.0077180      -2.8919223     0.003828926
   renter_rt       0.0069856       0.0013690       5.1026101    3.350008e-07
  vac_hsu_rt       0.0083237       0.0013581       6.1288540    8.851432e-10
      lambda       0.2009658
----------------------------------------------------------------------------
========================= END OF REPORT ==============================
```

Note there is a change in the output print with respect to the OLS. Instead of R squared, we now have the *pseudo* R squared. This is simply the correlation coefficient between the dependent variable (`y`) and the predicted values of our model. Although it is not strictly comparable to the traditional one, it is still a good guidance of *how right* our model performs.

One inconvenient of this procedure is that it only obtains a point estimate of the spatial parameter and hence does not allow to do inference on it. In a recent paper, Drukker et al. (2010) present an improvement on the techinique that allows to obtain inference even on the spatial parameter. PySAL implements it in the method `GM_Error_Hom`, where the `Hom` is named after the assumption of homoskedasticity made in the model.

In [18]:
```
from pysal.spreg import GM_Error_Hom
error_hom = GM_Error_Hom(y, x, w, name_x=x_names)
print error_hom.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIALLY WEIGHTED LEAST SQUARES (HOM) ESTIMATION
--------------------------------------------------------------------
Data set            :     unknown
Weights matrix      :     unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :           9
S.D. dependent var  :      0.3505  Degrees of Freedom    :         976
```

```
Pseudo R-squared    :    0.321329

----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       3.2541994       0.1267546      25.6732252    2.32761e-145
         hsu      -0.0002062       0.0000133     -15.4782228     4.867341e-54
    pop_dens      -0.0003919       0.0001028      -3.8131977    0.0001371803
    white_rt      -0.0032937       0.0012651      -2.6034443     0.009229225
    black_rt       0.0042317       0.0030615       1.3822208       0.1669039
     hisp_rt       0.0004479       0.0008266       0.5418071       0.5879514
   fem_nh_rt      -0.0223135       0.0077447      -2.8811329     0.003962486
   renter_rt       0.0069811       0.0013755       5.0753519    3.867796e-07
  vac_hsu_rt       0.0083214       0.0013643       6.0993920    1.064727e-09
      lambda       0.2176842       0.0463056       4.7010301    2.588524e-06
----------------------------------------------------------------------------
========================= END OF REPORT ==============================
```

At this point, we can evaluate the significance of `lambda`. Apparently in contrast with what we would expect from the LM tests on spatial autocorrelation, the coefficient appears highly significant. We will provide an explanation for this below when we delve into more complicated models but, before that, let us present a third estimation procedure implemented in PySAL for a spatial autoregressive error process. If we remember from the OLS output print, there were significant signs of heteroskedasticity, and the error models we have fit so far assume homoskedasticity. In a very recent reference, Arraiz et al. (2010) propose a procedure to estimate error models that is consistent to the presence of heteroskedasticity. PySAL implements this in the `spreg` module `error_sp_het` and it is as simple to use as the rest we have seen:

In [19]:
```python
from pysal.spreg import GM_Error_Het
error_het = GM_Error_Het(y, x, w, name_x=x_names)
print error_het.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIALLY WEIGHTED LEAST SQUARES (HET) ESTIMATION
--------------------------------------------------------------------
Data set            :      unknown
Weights matrix      :      unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :           9
S.D. dependent var  :      0.3505  Degrees of Freedom    :         976

Pseudo R-squared    :    0.321329

Heteroskedastic Corrected Standard Errors
----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       3.2541994       0.1399596      23.2509897    1.390113e-119
         hsu      -0.0002062       0.0000158     -13.0265120     8.647101e-39
    pop_dens      -0.0003919       0.0001034      -3.7907494    0.0001501934
    white_rt      -0.0032937       0.0012963      -2.5408348      0.01105882
    black_rt       0.0042317       0.0027850       1.5194830        0.128641
     hisp_rt       0.0004479       0.0007325       0.6113715       0.5409537
   fem_nh_rt      -0.0223135       0.0104736      -2.1304427      0.03313508
   renter_rt       0.0069811       0.0016474       4.2376742    2.258471e-05
  vac_hsu_rt       0.0083214       0.0015693       5.3027086     1.14097e-07
      lambda       0.2142769       0.0469120       4.5676366    4.932542e-06
----------------------------------------------------------------------------
========================= END OF REPORT ==============================
```

This procedure only differs from `GM_Error_Hom` in the moments and VC matrix, so all the coefficients except for the spatial parameter remain unchanged. The standard errors however slightly change.

## Spatial lag model

Imagine that, after examining the LM tests, we had decided for a lag model instead of the error. Running a spatial lag model is equally simple in PySAL thanks to the method GM_Lag. Note how, in this case, `w` has to be referenced. This is because we might be passing in additional instruments and hence it is not clear that the third place is for the weights object.

In [20]:
```
from pysal.spreg import GM_Lag
lag_model = GM_Lag(y, x, w=w, name_x=x_names, spat_diag=True)
print lag_model.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIAL TWO STAGE LEAST SQUARES ESTIMATION
-------------------------------------------------------------
Data set            :     unknown
Weights matrix      :     unknown
Dependent Variable  :     dep_var  Number of Observations:        985
Mean dependent var  :      2.7077  Number of Variables   :         10
S.D. dependent var  :      0.3505  Degrees of Freedom    :        975

Pseudo R-squared    :    0.341497
Spatial Pseudo R-squared:    0.328653

----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       2.4970736       0.2632379       9.4859966    2.400779e-21
         hsu      -0.0002039       0.0000132     -15.4584424    6.617946e-54
    pop_dens      -0.0003663       0.0000985      -3.7180632    0.0002007561
    white_rt      -0.0028577       0.0012417      -2.3015501      0.02136056
    black_rt       0.0026414       0.0028596       0.9236830       0.3556514
     hisp_rt      -0.0001681       0.0007887      -0.2131477       0.8312118
   fem_nh_rt      -0.0202109       0.0073684      -2.7429067     0.006089798
   renter_rt       0.0068149       0.0013256       5.1410192    2.732521e-07
  vac_hsu_rt       0.0078722       0.0013015       6.0484365    1.462583e-09
   W_dep_var       0.2740141       0.0836946       3.2739746     0.001060461
----------------------------------------------------------------------------
Instruments: W_hsu, W_pop_dens, W_white_rt, W_black_rt, W_hisp_rt,
             W_fem_nh_rt, W_renter_rt, W_vac_hsu_rt

DIAGNOSTICS FOR SPATIAL DEPENDENCE
TEST                           MI/DF       VALUE           PROB
Anselin-Kelejian Test            1        1.209171       0.2714964
========================= END OF REPORT ==============================
```

As we can see, the parameter for the lag of the dependent variable is positive and significant, similar to the error case. A new addition to the output print is the *spatial pseudo R squared*. In the lag model, we can obtain two different predicted values: the *naive* ones and the complete ones. The former, used for the pseudo R squared, include only the first lag of the dependent variable to predict the value of `y`; the latter, used in the spatial pseudo R squared, employs the reduced form of the model, thus incorporating all the feedback effects implicit in the model and offering a more correct estimate. In either case, we can see how the lag model seems to do a better job with this dataset.

Note also that we asked for spatial diagnostics of the residuals. PySAL implements the Anselin-Kelejian (AK) test for residuals of an IV estimation. The test is a modification of the LM-Error and is used to check for remaining spatial autocorrelation in the residuals of a regresion with instruments. Consistent with the initial LM tests from above, the AK does not find any significant spatial correlation, pointing again to the lag model as the preferred one.

The estimation procedure that `GM_Lag` employs is a particular case of the traditional Instrumental Variables (IV) approach, in which the endogeneity of the spatial lag of the variable is dealt with by using instruments. Kelejian and Prucha (1998-99) show that the optimal instruments for a lag model are the spatial lag of the exogenous variables, and `GM_Lag` follows their guidance. A less settled debate is how many orders of lags are optimal: just the first one (`WX`)? two (`WX` and `WWX`)? Very much like GeoDaSpace, PySAL uses only the first lag by default, but it allows you to modify it if you wish. Suppose we preferred to include two lags, the call would then be:

In [21]:
```
lag_model2lags = GM_Lag(y, x, w=w, name_x=x_names, w_lags=2)
print lag_model2lags.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIAL TWO STAGE LEAST SQUARES ESTIMATION
-------------------------------------------------------------
Data set            :     unknown
Weights matrix      :     unknown
Dependent Variable  :    dep_var  Number of Observations:         985
Mean dependent var  :     2.7077  Number of Variables   :          10
S.D. dependent var  :     0.3505  Degrees of Freedom    :         975

Pseudo R-squared    :   0.341153
Spatial Pseudo R-squared:   0.328877

----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       2.3211310       0.2579456       8.9985291    2.287619e-19
         hsu      -0.0002028       0.0000132     -15.3771109    2.331201e-53
    pop_dens      -0.0003530       0.0000985      -3.5844272    0.0003378187
    white_rt      -0.0027313       0.0012414      -2.2001304      0.02779765
    black_rt       0.0023588       0.0028592       0.8249764        0.409385
     hisp_rt      -0.0002885       0.0007881      -0.3661051       0.7142867
   fem_nh_rt      -0.0194107       0.0073666      -2.6349576     0.008414779
   renter_rt       0.0065425       0.0013234       4.9435842    7.669921e-07
  vac_hsu_rt       0.0076479       0.0013001       5.8823768    4.044165e-09
   W_dep_var       0.3371827       0.0815362       4.1353753     3.54375e-05
----------------------------------------------------------------------------
Instruments: W_hsu, W_pop_dens, W_white_rt, W_black_rt, W_hisp_rt,
             W_fem_nh_rt, W_renter_rt, W_vac_hsu_rt, W2_hsu, W2_pop_dens,
             W2_white_rt, W2_black_rt, W2_hisp_rt, W2_fem_nh_rt,
             W2_renter_rt, W2_vac_hsu_rt
========================= END OF REPORT ==============================
```

As mentioned before, this procedure is a particular case of IV estimation where the instruments are spatial. To demonstrate how they are created, we will replicate what `GM_Lag` builds internally and will pass it to the generic two stages least squares method in `spreg`. This will allow us to discover some more functionality in PySAL and, at the same time, to check that we actually obtain the same numbers as with the default.

We will try to match the default settings, so we will only need to compute the first lag of `X`. PySAL has a generic function (`lag_spatial`) that will do the heavy lifting in a very efficient way for us, and it works both with single vectors or with matrices:

In [22]:
```
from pysal import lag_spatial
from pysal.spreg import TSLS
wy = lag_spatial(w, y) #Get the spatial lag of y
wx = lag_spatial(w, x) #Get the lag of x to use as instruments
iv_model = TSLS(y, x, yend=wy, q=wx, name_x=x_names)
print iv_model.summary
```

```
REGRESSION
```

```
----------
SUMMARY OF OUTPUT: TWO STAGE LEAST SQUARES ESTIMATION
-----------------------------------------------------
Data set            :     unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :          10
S.D. dependent var  :      0.3505  Degrees of Freedom    :         975

Pseudo R-squared    :    0.341497

----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       2.4970736       0.2632379       9.4859966    2.400779e-21
         hsu      -0.0002039       0.0000132     -15.4584424    6.617946e-54
    pop_dens      -0.0003663       0.0000985      -3.7180632    0.0002007561
    white_rt      -0.0028577       0.0012417      -2.3015501      0.02136056
    black_rt       0.0026414       0.0028596       0.9236830       0.3556514
     hisp_rt      -0.0001681       0.0007887      -0.2131477       0.8312118
   fem_nh_rt      -0.0202109       0.0073684      -2.7429067     0.006089798
   renter_rt       0.0068149       0.0013256       5.1410192    2.732521e-07
  vac_hsu_rt       0.0078722       0.0013015       6.0484365    1.462583e-09
endogenous_1       0.2740141       0.0836946       3.2739746     0.001060461
----------------------------------------------------------------------------
Instruments: instrument_1, instrument_2, instrument_3, instrument_4,
             instrument_5, instrument_6, instrument_7, instrument_8
========================= END OF REPORT ==============================
```

As you can see for yourself, the results of `iv_model` exactly match those from `lag_model`.

### Full models

At this point, we have run a non-spatial model and found significant evidence of spatial autocorrelation; we have then run lag and error models separately to try account for it. Although the robust LM tests pointed to a lag model, we found significant the `lambda` coefficient in the error model, so it would not be unrealistic to think that a full model with both spatial lag and error would be a good fit. In this section we will show what options are available in PySAL to construct more complicated models that include spatial effects in the dependent variable as well as in the error term. In particular, we will use two main specifications: a SARAR model that models autoregressive terms in both the lag and the error; and a combination of a lag model with the Spatial Heteroskedasticity and Autocorrelation Consistent (SHAC) variance covariance matrix.

Our first option is to fit a model that introduces autoregressive parameters for the lag and the error. PySAL offers combinations of the lag model we have seen with all the error models presented before. For the sake of simplicity, and without loss of generality, we will stick to our last choice (`GM_Error_Het`) to blend it with the lag, since it offers the possibility of inference in the spatial parameter and accounts for heteroskedasticity, which we have present in the dataset. These models are called `Combo` in PySAL, to allude to the fact they combine a lag and an error.

In [23]:
```
from pysal.spreg import GM_Combo_Het
sarar_het = GM_Combo_Het(y, x, w=w, name_x=x_names)
print sarar_het.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIALLY WEIGHTED TWO STAGE LEAST SQUARES (HET) ESTIMATION
------------------------------------------------------------------------------
Data set            :     unknown
Weights matrix      :     unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :          10
S.D. dependent var  :      0.3505  Degrees of Freedom    :         975
```

```
Pseudo R-squared    :    0.341478
Spatial Pseudo R-squared:    0.329460

Heteroskedastic Corrected Standard Errors
----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       2.4533311       0.2807015       8.7399985    2.331135e-18
         hsu      -0.0002023       0.0000153     -13.2302169    5.871765e-40
    pop_dens      -0.0003740       0.0000979      -3.8197206    0.0001336029
    white_rt      -0.0028910       0.0011420      -2.5315327      0.01135652
    black_rt       0.0023357       0.0022767       1.0259095       0.3049342
     hisp_rt      -0.0002446       0.0006538      -0.3741339       0.7083047
   fem_nh_rt      -0.0202743       0.0090433      -2.2419124      0.02496704
   renter_rt       0.0070361       0.0015542       4.5270082    5.982462e-06
  vac_hsu_rt       0.0079395       0.0013681       5.8031888    6.506547e-09
   W_dep_var       0.2903322       0.0888340       3.2682557     0.001082126
      lambda      -0.1720565       0.1272811      -1.3517838       0.1764445
----------------------------------------------------------------------------
Instruments: W_hsu, W_pop_dens, W_white_rt, W_black_rt, W_hisp_rt,
             W_fem_nh_rt, W_renter_rt, W_vac_hsu_rt
========================= END OF REPORT ==============================
```

Note how, once the spatial lag is introduced, the spatial parameter in the error term becomes insignificant. We can now close the argument we started with the LM tests and seemed to counter-argue with the error model: the best specification for this dataset appears to be a lag model; however, if a spatial error parameter is introduced instead of a lag parameter, part of the spatial effects in the model are pushed to the error and picked up by `lambda`, which becomes significant for the lack of a better structure to model space (i.e. lag). When such preferable structure is introduced and the spatial effects modelled properly, the error does not have remaining spatial autocorrelation and the parameter becomes not significant.

The problem of the straight lag model is that it does not account for heteroskedasticity, which is a problem with our data. To solve the issue, we can use the White correction in the same way we did with OLS:

In [24]:
```
lag_white = GM_Lag(y, x, w=w, name_x=x_names, robust='white')
print lag_white.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIAL TWO STAGE LEAST SQUARES ESTIMATION
----------------------------------------------------------------
Data set            :      unknown
Weights matrix      :      unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :          10
S.D. dependent var  :      0.3505  Degrees of Freedom    :         975

Pseudo R-squared    :    0.341497
Spatial Pseudo R-squared:    0.328653

White Standard Errors
----------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
----------------------------------------------------------------------------
    CONSTANT       2.4970736       0.2988176       8.3565147     6.45984e-17
         hsu      -0.0002039       0.0000155     -13.1234882    2.415553e-39
    pop_dens      -0.0003663       0.0001009      -3.6291256    0.0002843828
    white_rt      -0.0028577       0.0012154      -2.3512323      0.01871135
    black_rt       0.0026414       0.0024471       1.0793995       0.2804097
     hisp_rt      -0.0001681       0.0006862      -0.2449773        0.806474
   fem_nh_rt      -0.0202109       0.0097566      -2.0715185      0.03831037
```

```
    renter_rt       0.0068149       0.0016322       4.1751404    2.978023e-05
   vac_hsu_rt       0.0078722       0.0014515       5.4232996     5.85088e-08
    W_dep_var       0.2740141       0.0941695       2.9097969     0.003616637
------------------------------------------------------------------------------------
Instruments: W_hsu, W_pop_dens, W_white_rt, W_black_rt, W_hisp_rt,
             W_fem_nh_rt, W_renter_rt, W_vac_hsu_rt
========================= END OF REPORT ==============================
```

Since White is a VC matrix correction, only the standard errors will be affected. Compared to the initial lag model, standard errors are in general larger, which results in `fem_nh_rt` becoming insignificant at the 1% level.

Now that we have our best model, we can compare the coefficients with the ones obtained previously to assess the effect of the spatial effects present in the data on our estimations and conclusions. If we look at the estimates of `lag_white` in comparison with either the OLS or error models, we find they tend to be smaller in the former. This means that the effect of space makes models that do not account for it overestimate the importance of our regressors in explaining the dependent variable. Just by looking at the actual numbers, it might seem like the changes are not that large, but keep in mind the scale issues and the fact that our dependent variable is expressed in logs. If you count both in, depending on the motivation of the model, the changes may not seem that tiny.

As a final note for completeness, we will showcase one more model available in PySAL. As mentioned before, Arraiz et al. (2010) present a nonparametric procedure to estimate the VC matrix that accounts for both heteroskedasticity *and* spatial autocorrelation. Since it is a correction of the VC matrix and hence only affects the standard errors, we can use it with either non-spatial models or with a spatial lag model. The call we need to make is not very different from that for White, but before we can do that we need to create a kernel weights object, for which we will use 10 neighbors and, for this example, a quadratic kernel function.

In [25]:
```
wk = ps.open('../workshop_data/phx_k_triangular.kwt', dataFormat='gwt').read()
lag_hac = GM_Lag(y, x, w=w, name_x=x_names, robust='hac', gwk=wk)
print lag_hac.summary
```

```
REGRESSION
----------
SUMMARY OF OUTPUT: SPATIAL TWO STAGE LEAST SQUARES ESTIMATION
--------------------------------------------------------------
Data set            :      unknown
Weights matrix      :      unknown
Dependent Variable  :     dep_var  Number of Observations:         985
Mean dependent var  :      2.7077  Number of Variables   :          10
S.D. dependent var  :      0.3505  Degrees of Freedom    :         975

Pseudo R-squared    :    0.341497
Spatial Pseudo R-squared:    0.328653

HAC Standard Errors; Kernel Weights: unknown
------------------------------------------------------------------------------------
    Variable     Coefficient       Std.Error     z-Statistic     Probability
------------------------------------------------------------------------------------
    CONSTANT       2.4970736       0.3072108       8.1282081    4.356827e-16
         hsu      -0.0002039       0.0000163     -12.5274161    5.285681e-36
    pop_dens      -0.0003663       0.0001001      -3.6590947    0.0002531078
    white_rt      -0.0028577       0.0011902      -2.4010425      0.01634844
    black_rt       0.0026414       0.0023127       1.1421190       0.2534045
     hisp_rt      -0.0001681       0.0006744      -0.2492576       0.8031616
   fem_nh_rt      -0.0202109       0.0095914      -2.1071828      0.03510174
   renter_rt       0.0068149       0.0016346       4.1692471    3.056075e-05
  vac_hsu_rt       0.0078722       0.0014090       5.5872065    2.307513e-08
   W_dep_var       0.2740141       0.0978080       2.8015507     0.005085764
------------------------------------------------------------------------------------
Instruments: W_hsu, W_pop_dens, W_white_rt, W_black_rt, W_hisp_rt,
             W_fem_nh_rt, W_renter_rt, W_vac_hsu_rt
```

```
========================= END OF REPORT ==============================
```

As it can be seen, the standard errors do not differ much from those obtained in `lag_white`. This is because the SHAC does as good of a job as the White at correcting for heteroskedasticity and, since residual spatial autocorrelation is not an issue once the spatial lag is included, there is very little room left for the SHAC to improve the White.

## Non-spatial endogenous variables

In order to complete the overview of `pysal.spreg`, we have one more substantive feature to show. Each of the models we have just used (non-spatial, spatial error, lag and combo) allow the user to pass not only exogenous regressors, but also endogenous variables. This is implemented again using IV estimation, so any of the models we have just seen may be turned into a two-stage least squares procedure in which you can allow some of you explanatory variables to be endogenous and use other variables to instrument for them.

As a mere example, we will consider now that the variable population density (`pop_dens`) is endogenous and that we want to instrument for it using the inverse of land area. This means we have to re-factor our matrix `x` and create two new ones (which in this case will be of dimension n by 1): one for endogenous variables (`yend`) and one for the instruments (`q`). We also can use this example to show a bit more how to manipulate data in a *pythonic* way. It takes a bit of house-keeping, but it is not complicated:

In [26]:
```
#Pull out the X variables without including population density
x_names = ['hsu', 'white_rt', 'black_rt', 'hisp_rt', \
           'fem_nh_rt', 'renter_rt', 'vac_hsu_rt']
x = np.array([db.by_col(var) for var in x_names]).T
#Create inverse of land area
area = np.array(db.by_col('ALAND10')).reshape((x.shape[0], 1))
inv_area = 1. / area
#Create the matrix for the instruments
q_names = ['inv_area']
q = inv_area
#Create the matrix for the endogenous variable
yend_names = ['pop_dens']
yend = np.array([db.by_col(var) for var in yend_names]).T
```

Once we have all the pieces ready, we can run the models very much like we have done before. Since the main purpose now is to show you how to call the functions when you have endogenous variables, we will not print the summary outputs to save space. Also, in order not to import every method one by one, we will call them directly from `pysal.spreg`, showing you yet another way to access the code (note that some of the models include now the name `Endog` for endogeneity).

In [27]:
```
#Non-spatial model (w only required for spatial diagnostics
model = ps.spreg.TSLS(y, x, w=w, yend=yend, q=q, \
                      name_x=x_names, name_yend=yend_names, name_q=q_names)
#KP98-99 Error model
model = ps.spreg.GM_Endog_Error(y, x, w=w, yend=yend, q=q, \
                      name_x=x_names, name_yend=yend_names, name_q=q_names)
#Drukker et al. error model (Hom)
model = ps.spreg.GM_Endog_Error_Hom(y, x, w=w, yend=yend, q=q, \
                      name_x=x_names, name_yend=yend_names, name_q=q_names)
#Arraiz. et al. error model (Het)
model = ps.spreg.GM_Endog_Error_Het(y, x, w=w, yend=yend, q=q, \
                      name_x=x_names, name_yend=yend_names, name_q=q_names)
#Lag model
model = ps.spreg.GM_Lag(y, x, w=w, yend=yend, q=q, \
                      name_x=x_names, name_yend=yend_names, name_q=q_names)
#Combo model
model = ps.spreg.GM_Combo_Het(y, x, w=w, yend=yend, q=q, \
                      name_x=x_names, name_yend=yend_names, name_q=q_names)
```

# GeoDaSpace Vs. pysal.spreg

As we have just shown, pysal.spreg can do everything that GeoDaSpace does; in fact, all the core functionality from GeoDaSpace is actually powered by the same code. At this point, you might be wondering why then make the investment to learn it this way, when you can click through the dialogs in GeoDaSpace. In this section, we will discuss some of the features that you can enjoy only if you use the command line and will demonstrate it using the spatial diagnostics we saw before as an example.

## Extra parameters and tweaks

GeoDaSpace aims at being a simple, easy to use interface that puts all the advanced spatial econometrics implemented in PySAL one click away. This is partly done at the cost of setting some pysal.spreg options to reasonable defaults. In cases when we do need to change those defaults because the problem at hand requires some customization of estimation procedures, the command line is the way to go. In this document, we will not detail one by one all these options; instead, we will review how they can be found out and accessed.

PySAL takes documentation very seriously, and the spreg module is no exception. The inline help provided is so good that, with some basic Python knowledge at hand, it allows to explore all the functionality from the command line, right in place. The documentation has a more or less fixed structure that repeats throughout the modules:

- Basic description of the functionality.
- Parameters to be passed to the method.
- Attributes / output returned by the method.
- Examples of how to use the code.
- References to works cited before.

Accessing this information will detail every parameter and tweak that the code allows and it will tell us how to use it, so it is the best way to discover whether the customization we are after is implemented in PySAL. There are several ways to access the documentation, but two are the most preferred ones:

- Inline help from the Python interpreter: simply type `help(pysal.function)` and the documentation for `pysal.function` will be displayed. Let us exemplify it with OLS:

In [28]:
```
help(ps.spreg.OLS)
```

```
Help on class OLS in module pysal.spreg.ols:

class OLS(BaseOLS, pysal.spreg.user_output.DiagnosticBuilder)
 |  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.
 |  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
 |
 |
 |  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
 |
 |
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> import pysal
 |
 |  Open data on Columbus neighborhood crime (49 areas) using pysal.open().
 |  This is the DBF associated with the Columbus shapefile.  Note that
 |  pysal.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 = pysal.open(pysal.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). pysal.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')
 |
 |  pysal.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. All of these results can be independently
 |  accessed as attributes of the regression object created by running
 |  pysal.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 ols.t_stat[2][0]
|  -2.65440864272
|  >>> print ols.t_stat[2][1]
|  0.0108745049098
|  >>> ols.r2
|  0.34951437785126105
|  >>> print ols.summary
|  REGRESSION
|  ----------
|  SUMMARY OF OUTPUT: ORDINARY LEAST SQUARES ESTIMATION
|  ----------------------------------------------------
|  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
|  <BLANKLINE>
|  R-squared           :    0.349514
|  Adjusted R-squared  :      0.3212
|  Sum squared residual:   10647.015  F-statistic           :     12.3582
|  Sigma-square        :     231.457  Prob(F-statistic)     : 5.06369e-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
|  <BLANKLINE>
|  ----------------------------------------------------------------------------
|      Variable     Coefficient       Std.Error     t-Statistic     Probability
|  ----------------------------------------------------------------------------
|      CONSTANT      46.4281827      13.1917570       3.5194844    0.0009866767
|        income       0.6289840       0.5359104       1.1736736       0.2465669
|         crime      -0.4848885       0.1826729      -2.6544086       0.0108745
|  ----------------------------------------------------------------------------
|  <BLANKLINE>
|  REGRESSION DIAGNOSTICS
|  MULTICOLLINEARITY CONDITION NUMBER   12.537555
|  TEST ON NORMALITY OF ERRORS
|  TEST                     DF          VALUE            PROB
|  Jarque-Bera               2          39.706155        0.0000000
|  <BLANKLINE>
|  DIAGNOSTICS FOR HETEROSKEDASTICITY
|  RANDOM COEFFICIENTS
|  TEST                     DF          VALUE            PROB
|  Breusch-Pagan test        2           5.766791        0.0559445
|  Koenker-Bassett test      2           2.270038        0.3214160
|  <BLANKLINE>
|  SPECIFICATION ROBUST TEST
|  TEST                     DF          VALUE            PROB
|  White                     5           2.906067        0.7144648
|  ========================= END OF REPORT ==============================
|
|  If the optional parameters w and spat_diag are passed to pysal.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 0.2037 with a standardized value of 2.5918 and a p-value of
 |  0.009547.
 |
 |  >>> w =
pysal.weights.rook_from_shapefile(pysal.examples.get_path("columbus.shp"))
 |  >>> 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 ols.moran_res[0]
 |  0.20373540938
 |  >>> print ols.moran_res[1]
 |  2.59180452208
 |  >>> print ols.moran_res[2]
 |  0.00954740031251
 |
 |  Method resolution order:
 |      OLS
 |      BaseOLS
 |      pysal.spreg.utils.RegressionPropsY
 |      pysal.spreg.utils.RegressionPropsVM
 |      pysal.spreg.user_output.DiagnosticBuilder
 |
 |  Methods defined here:
 |
 |  __init__(self, y, x, w=None, robust=None, gwk=None, sig2n_k=True,
nonspat_diag=True, spat_diag=False, moran=False, vm=False, name_y=None, name_x=None,
name_w=None, name_gwk=None, name_ds=None)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from pysal.spreg.utils.RegressionPropsY:
 |
 |  mean_y
 |
 |  std_y
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from pysal.spreg.utils.RegressionPropsVM:
 |
 |  sig2n
 |
 |  sig2n_k
 |
 |  utu
 |
 |  vm
```

As you can see, the output is significant and it details all the parameters you can tweak (e.g. sig2n_k).

- Online help: pysal.org hosts not only the code but an HTML version of the documentation. Very handy if we want to quickly google some function.

## Programatic access to the functionality and batch processing

This may seem obvious but is not: the command line gives us programatic access to the code. This means that every action we perform can be saved (e.g. in a script) and that we no longer have to rely on what we clicked or did not click to know where the output comes from. Also, programatic access to the core of GeoDaSpace opens new options that the GUI does not offer: reproducibility (an important aspect of an open science) is much easier; we can use part of our code in bigger projects or re-use it over and over without re-doing the work for ourselves; we can now run it from a remote server (with more power than our desktop) and then use it for larger datasets, for example.

Probably one of the most useful options that the command line has to offer is the possibility to run the models we have seen in loops so we can use them as part of simulations or to quickly and efficiently try several specifications (e.g. different weights, variable combinations, etc.). As a very simple example of how to implement these ideas, we will walk through the following case: imagine that we want to see how different weights make results change in the spatial diagnostics we have shown before. To do that, we only need to run the OLS once because it is aspatial, but we have to compute the LM tests once for each weights specification. This is a perfect case to use a for loop, let us see how we would implement it (in order to see the results, we will also setup a little printout for ourselves):

In [29]:
```
#Import the LM tests method
from pysal.spreg import LMtests
#Specify the files for the weights we want to try as a list
w_files = ['../workshop_data/phx_knn06.gwt', \
           '../workshop_data/phx_knn10.gwt', \
           '../workshop_data/phx_rook.gal', \
           '../workshop_data/phx_queen.gal']
#Run the OLS
model = ps.spreg.OLS(y, x, spat_diag=False, nonspat_diag=False)
#Setup the loop over the weights files
for w_file in w_files:
    print 'File: ', w_file
    w = ps.open(w_file).read()
    lms = LMtests(model, w)
    print '\tLM error: %.4f\t(%.4f)'%lms.lme
    print '\tLM lag:   %.4f\t(%.4f)'%lms.lml
    print '\tSARMA:    %.4f\t(%.4f)'%lms.sarma
    print '\tRobust LM error:   %.4f\t(%.4f)'%lms.rlme
    print '\tRobust LM lag:     %.4f\t(%.4f)'%lms.rlml
```

```
File:  ../workshop_data/phx_knn06.gwt
        LM error: 22.1933       (0.0000)
        LM lag:   29.1600       (0.0000)
        SARMA:    29.1979       (0.0000)
        Robust LM error:   0.0379       (0.8456)
        Robust LM lag:     7.0046       (0.0081)
File:  ../workshop_data/phx_knn10.gwt
        LM error: 31.0525       (0.0000)
        LM lag:   48.2486       (0.0000)
        SARMA:    49.1130       (0.0000)
        Robust LM error:   0.8644       (0.3525)
        Robust LM lag:     18.0604      (0.0000)
File:  ../workshop_data/phx_rook.gal
        LM error: 16.2499       (0.0001)
        LM lag:   21.7033       (0.0000)
        SARMA:    31.7729       (0.0000)
        Robust LM error:   10.0696      (0.0015)
        Robust LM lag:     15.5230      (0.0001)
File:  ../workshop_data/phx_queen.gal
        LM error: 22.9589       (0.0000)
        LM lag:   9.3013        (0.0023)
        SARMA:    27.5552       (0.0000)
        Robust LM error:   18.2539      (0.0000)
        Robust LM lag:     4.5963       (0.0320)
```

Note that to perform this, we only had to run the OLS model once, gaining some speed up that becomes very important if the size of the dataset is very large.

This is only a very simple example of how scripting and programatic access can be a powerful element to help us in our analysis. With these tools, the possibilities of building fairly complicated analysis or systems become not only feasible, but reachable. Feel free to tinker and hack with pysal.spreg as much as you want, the limit is the sky!