lab_05

October 28, 2015

1 Spatial weights

```
In [1]: %matplotlib inline

        import seaborn as sns
        import pandas as pd
        import pysal as ps
        import geopandas as gpd
        import numpy as np
        import matplotlib.pyplot as plt
```

1.1 Data

For this tutorial, we will use again the recently released 2015 Index of Multiple Deprivation (IMD) for England and Wales. This dataset can be most easily downloaded from the CDRC data store (link) and, since it already comes both in tabular as well as spatial data format (shapefile), it does not need merging or joining to additional geometries.

In addition, we will be using the lookup between LSOAs and Medium Super Output Areas (MSOAs), which can be downloaded on this link. This connects each LSOA polygon to the MSOA they belong to. MSOAs are a coarser geographic delineation from the Office of National Statistics (ONS), within which LSOAs are nested. That is, no LSOA boundary crosses any of an MSOA.

As usual, let us set the paths to the folders containing the files before anything so we can then focus on data analysis exclusively (keep in mind the specific paths will probably be different for your computer):

```
In [94]: # This will be different on your computer and will depend on where
         # you have downloaded the files
         imd_shp = '../../../../data/E08000012_IMD/shapefiles/E08000012.shp'
         lookup_path = '../../../../data/Output_areas_(2011)_to_lower_layer_super_output_areas_(2011)_t
```

Let us load the IMD data first:

```
In [120]: # Read the file in
          imd = gpd.read_file(imd_shp)
          # Index it on the LSOA ID
          imd = imd.set_index('LSOA11CD')
          # Display summary
          imd.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
Index: 298 entries, E01006512 to E01033768
Data columns (total 12 columns):
crime         298 non-null float64
education     298 non-null float64
employment    298 non-null float64
```

```
geometry      298 non-null object
health        298 non-null float64
housing       298 non-null float64
idaci         298 non-null float64
idaopi        298 non-null float64
imd_rank      298 non-null int64
imd_score     298 non-null float64
income        298 non-null float64
living_env    298 non-null float64
dtypes: float64(10), int64(1), object(1)
memory usage: 30.3+ KB
```

1.2 Building spatial weights in PySAL

1.2.1 Contiguity

- **Queen**

```
In [16]: w_queen = ps.queen_from_shapefile(imd_shp, idVariable='LSOA11CD')
         w_queen
```

```
Out[16]: <pysal.weights.weights.W at 0x7f1d26655790>
```

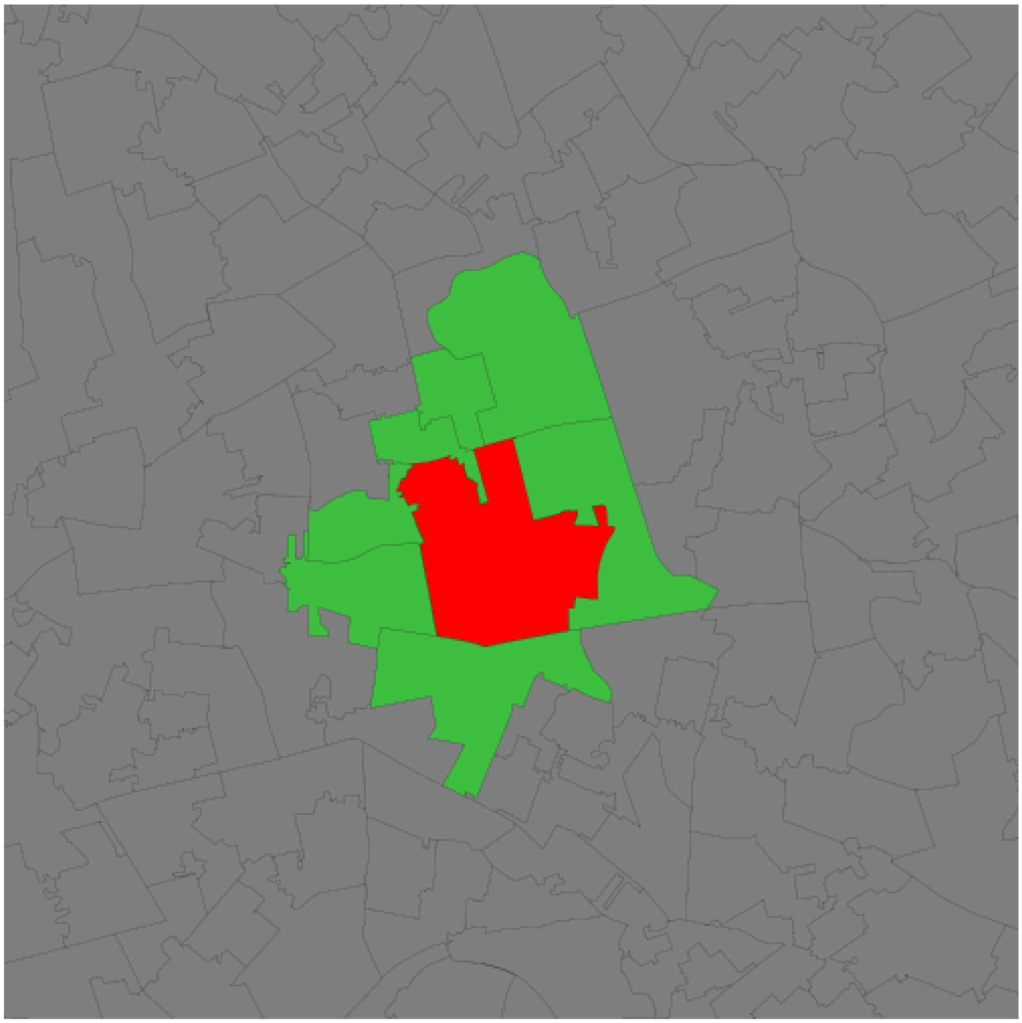
```
In [18]: w_queen['E01006690']
```

```
Out[18]: {'E01006691': 1.0,
          'E01006692': 1.0,
          'E01006695': 1.0,
          'E01006697': 1.0,
          'E01006720': 1.0,
          'E01006759': 1.0,
          'E01033763': 1.0}
```

Check Weights tutorial.

```
In [170]: # Setup figure
          f, ax = plt.subplots(1, figsize=(9, 9))
          # Plot base layer of polygons
          for poly in imd['geometry']:
              gpd.plotting.plot_multipolygon(ax, poly, facecolor='k', linewidth=0.1)
          # Select focal polygon
          focus = imd.loc['E01006690', 'geometry']
          # Plot focal polygon
          gpd.plotting.plot_multipolygon(ax, focus, facecolor='red', alpha=1, linewidth=0)
          # Plot neighbors
          for nei in w_queen['E01006690']:
              nei_poly = imd.loc[nei, 'geometry']
              gpd.plotting.plot_multipolygon(ax, nei_poly, facecolor='lime', linewidth=0)
          # Title
          f.suptitle("Queen neighbors of `E01006690`")
          # Style and display on screen
          ax.set_axis_off()
          plt.axis('equal')
          ax.set_ylim(388000, 393500)
          ax.set_xlim(336000, 339500)
          plt.show()
```

Queen neighbors of `E01006690`



- **Rook**

```
In [23]: w_rook = ps.rook_from_shapefile(imd_shp, idVariable='LSOA11CD')
         w_rook

Out[23]: <pysal.weights.weights.W at 0x7f1d26ad6090>
```

[Optional exercise]

Create a similar map for the rook neighbors of polygon `E01006580`. How would it differ if the spatial weights were created based on the queen criterion?

1.2.2 Distance

- **K-Nearest Neighbors**
- **Inverse distance**

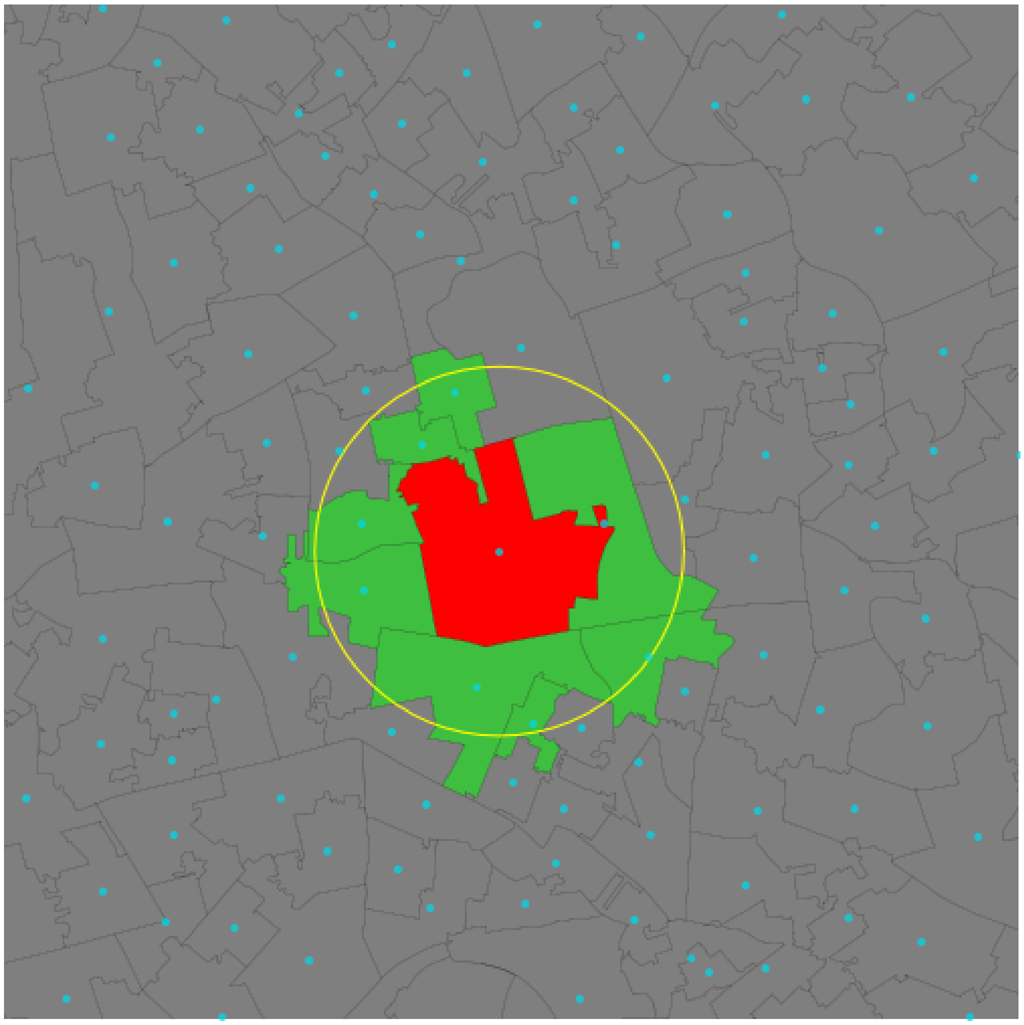
```
In [31]: w_dist1km = ps.threshold_binaryW_from_shapefile(imd_shp, 1000, idVariable='LSOA11CD')
```

[Optional extension. Lecture figure]

Below is how to build a visualization for distance-based weights that displays the polygons, highlighting the focus and its neighbors, and then overlays the centroids and the buffer used to decide whether a polygon is a neighbor or not. Since this is distance-based weights, there needs to be a way to establish distance between two polygons and, in this case, the distance between their centroids is used.

```
In [171]: # Setup figure
          f, ax = plt.subplots(1, figsize=(9, 9))
          # Plot base layer of polygons
          for poly in imd['geometry']:
              gpd.plotting.plot_multipolygon(ax, poly, facecolor='k', linewidth=0.1)
          # Select focal polygon
          focus = imd.loc['E01006690', 'geometry']
          # Plot focal polygon
          gpd.plotting.plot_multipolygon(ax, focus, facecolor='red', alpha=1, linewidth=0)
          # Plot neighbors
          for nei in w_dist1km['E01006690']:
              nei_poly = imd.loc[nei, 'geometry']
              gpd.plotting.plot_multipolygon(ax, nei_poly, facecolor='lime', linewidth=0)
          # Plot 1km buffer
          buf = focus.centroid.buffer(1000)
          gpd.plotting.plot_multipolygon(ax, buf, edgecolor='yellow', alpha=0)
          # Plot centroids of neighbor
          pts = np.array([(pt.x, pt.y) for pt in imd.centroid])
          ax.plot(pts[:, 0], pts[:, 1], color='#00d8ea', linewidth=0, alpha=0.75, marker='o', markersize
          #imd.centroid.plot(axes=ax)
          # Title
          f.suptitle("Neighbors within 1km of `E01006690`")
          # Style, zoom and display on screen
          ax.set_axis_off()
          plt.axis('equal')
          ax.set_ylim(388000, 393500)
          ax.set_xlim(336000, 339500)
          plt.show()
```

Neighbors within 1km of `E01006690`



1.2.3 Block weights

For this case, we will build a spatial weights matrix that connects every LSOA with all the other ones in the same MSOA. To do this, we first need a lookup list that connects both kinds of geographies:

In [144]:
```
lookup = pd.read_csv(lookup_path+'OA11_LSOA11_MSOA11_LAD11_EW_LUv2.csv')
lookup = lookup[['LSOA11CD', 'MSOA11CD']].drop_duplicates(take_last=True)\
                                         .set_index('LSOA11CD')['MSOA11CD']
lookup.head()
```

Out[144]:
```
LSOA11CD
E01000002    E02000001
```

```
E01032740    E02000001
E01000005    E02000001
E01000009    E02000017
E01000008    E02000016
Name: MSOA11CD, dtype: object
```

Since the original file contains much more information than we need for this exercise, note how in line 2 we limit the columns we keep to only two, `LSOA11CD` and `MSOA11CD`. We also add an additional command, `drop_duplicates`, which removes elements whose index is repeated more than once, as is the case in this dataset (every LSOA has more than one row in this table). By adding the `take_last` argument, we make sure that one and only one element of each index value is retained. For ease of use later on, we set the index, that is the name of the rows, to `LSOA11CD`. This will allow us to perform easy lookups without having to perform full `DataFrame` queries, and it is also a more computationally efficient way to select observations.

For example, if we want to know in which MSOA the polygon `E01000003` is, we just need to type:

```
In [147]: lookup.loc['E01000003']
```

```
Out[147]: 'E02000001'
```

With the lookup in hand, let us append it to the IMD table to keep all the necessary pieces in one place only:

```
In [149]: imd['MSOA11CD'] = lookup
```

Now we are ready to build a block spatial weights matrix that connects as neighbors all the LSOAs in the same MSOA. Using `PySAL`, this is a one-line task:

```
In [154]: w_block = ps.block_weights(imd['MSOA11CD'])
```

In this case, `PySAL` does not allow to pass the argument `idVariable` as above. As a result, observations are named after the the order the occupy in the list:

```
In [156]: w_block[0]
```

```
Out[156]: {218: 1.0, 219: 1.0, 220: 1.0, 292: 1.0}
```

The first element is neighbor of observations 218, 129, 220, and 292, all of them with an assigned weight of 1. However, it is easy enough to correct this by using the additional method `remap_ids`:

```
In [161]: w_block.remap_ids(imd.index)
```

Now if you try `w_bloc[0]`, it will return an error. But if you query for the neighbors of an observation by its LSOA id, it will work:

```
In [167]: w_block['E01006512']
```

```
Out[167]: {u'E01006747': 1.0, u'E01006748': 1.0, u'E01006751': 1.0, u'E01033763': 1.0}
```

[Optional exercise]
For block weights, create a similar map to that of queen neighbors of polygon `E01006690`.

1.3 Standardizing W matrices

In [176]:
```
w_queen['E01006690']
```

Out[176]:
```
{'E01006691': 1.0,
 'E01006692': 1.0,
 'E01006695': 1.0,
 'E01006697': 1.0,
 'E01006720': 1.0,
 'E01006759': 1.0,
 'E01033763': 1.0}
```

In [178]:
```
w_queen.transform
```

Out[178]:
```
'O'
```

In [179]:
```
w_queen.transform = 'R'
```

In [180]:
```
w_queen['E01006690']
```

Out[180]:
```
{'E01006691': 0.14285714285714285,
 'E01006692': 0.14285714285714285,
 'E01006695': 0.14285714285714285,
 'E01006697': 0.14285714285714285,
 'E01006720': 0.14285714285714285,
 'E01006759': 0.14285714285714285,
 'E01033763': 0.14285714285714285}
```

In [182]:
```
pd.Series(w_queen['E01006690']).sum()
```

Out[182]:
```
0.99999999999999978
```

In [183]:
```
w_queen.transform = 'O'
```

In [184]:
```
w_queen['E01006690']
```

Out[184]:
```
{'E01006691': 1.0,
 'E01006692': 1.0,
 'E01006695': 1.0,
 'E01006697': 1.0,
 'E01006720': 1.0,
 'E01006759': 1.0,
 'E01033763': 1.0}
```

1.4 Spatial Lag

In [191]:
```
w_queen.transform = 'R'
w_queen_score = ps.lag_spatial(w_queen, imd['imd_score'])
w_queen_score
```

Out[191]:
```
array([ 48.27833333,  34.96777778,  46.538     ,  40.02375   ,
        63.738     ,  15.186     ,  44.95714286,  28.94833333,
        30.52428571,  31.63      ,  19.00666667,  13.07857143,
        24.07285714,  11.91      ,  19.08333333,  20.872     ,
        20.588     ,  20.8225    ,  22.61833333,  20.18      ,
        26.54      ,  26.27666667,  29.52166667,  34.0325    ,
        36.095     ,  12.25285714,  73.82      ,  50.20444444,
```

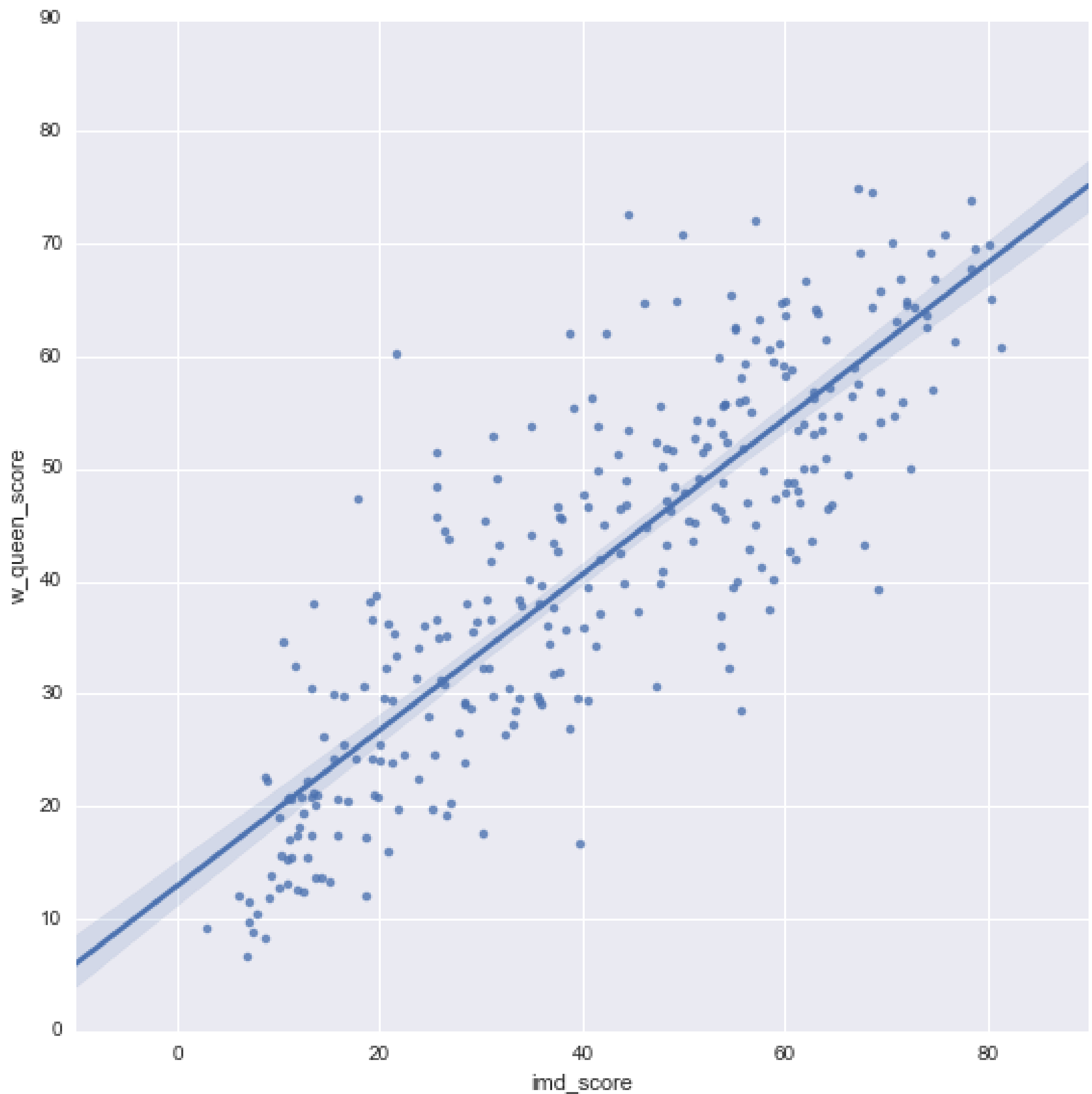
```
63.29      , 51.466     , 53.36333333, 61.93      ,
66.624     , 67.7725    , 74.78833333, 44.68714286,
24.04      , 26.9       , 62.3825    , 20.87428571,
31.19      , 49.96888889, 36.54166667, 70.8275    ,
70.08285714, 69.16166667, 69.56142857, 72.504     ,
42.474     , 45.63833333, 37.17571429, 38.31428571,
32.295     , 35.3       , 46.99333333, 54.71      ,
43.072     , 59.825     , 28.53625   , 20.354     ,
20.082     , 20.53      , 15.34285714, 17.05857143,
 8.2225    ,  9.64      ,  9.02166667, 28.005     ,
34.164     , 28.90833333, 43.77571429, 27.24      ,
20.698     , 20.666     , 15.45375   ,  8.8025    ,
15.92142857,  6.618     , 13.21666667, 13.58333333,
10.30333333, 57.162     , 48.68      , 51.895     ,
54.20125   , 43.43142857, 45.20375   , 47.09428571,
52.96666667, 56.295     , 52.668     , 58.88285714,
60.642     , 50.89857143, 47.8275    , 61.458     ,
64.71666667, 48.85571429, 31.36666667, 41.91833333,
22.41      , 19.628     , 29.86555556, 22.222     ,
30.4225    , 29.31      , 38.635     , 35.96333333,
23.86545455, 33.31666667, 28.555     , 54.67285714,
65.364     , 49.468     , 46.626     , 39.282     ,
44.442     , 29.31571429, 60.235     , 46.468     ,
52.238     , 39.705     , 43.19666667, 54.71285714,
56.93      , 45.31166667, 34.538     , 29.25      ,
37.0225    , 36.60428571, 37.66      , 29.606     ,
34.15      , 42.74      , 41.98285714, 51.30375   ,
39.31      , 49.955     , 53.80333333, 42.61      ,
30.6325    , 17.39      , 37.83571429, 20.76      ,
38.138     , 25.35142857, 37.994     , 45.42125   ,
56.3       , 64.515     , 60.5325    , 64.31      ,
62.5375    , 59.18375   , 55.85857143, 12.70166667,
19.68125   , 12.02857143, 13.762     , 15.594     ,
13.53666667, 21.096     , 29.48833333, 23.832     ,
12.576     , 63.58142857, 56.82      , 49.87375   ,
74.47166667, 64.83166667, 63.67      , 55.996     ,
64.058     , 70.672     , 63.12      , 65.71666667,
71.982     , 69.17      , 66.72714286, 61.40333333,
52.84      , 50.05333333, 46.43666667, 39.7775    ,
61.94      , 48.68125   , 43.264     , 45.405     ,
46.2675    , 41.755     , 37.9125    , 35.38166667,
53.282     , 37.33666667, 52.92      , 42.63428571,
29.74571429, 49.78714286, 47.5775    , 40.0675    ,
47.834     , 55.44666667, 51.73833333, 55.62333333,
52.984     , 56.71      , 48.018     , 55.57833333,
51.706     , 48.31333333, 41.20714286, 39.854     ,
49.11428571, 29.68857143, 43.07142857, 29.4       ,
35.836     , 51.344     , 45.2575    , 53.695     ,
46.825     , 62.47      , 58.21      , 55.875     ,
56.48      , 59.28666667, 57.95      , 59.47333333,
64.78      , 65.04      , 53.83142857, 54.076     ,
53.442     , 58.786     , 52.34666667, 64.372     ,
61.02142857, 36.92166667, 28.47714286, 51.58      ,
46.715     , 39.4925    , 17.395     , 32.382     ,
```

```
43.47      , 64.695     , 66.768     , 64.86666667,
32.31      , 32.2275    , 31.854     , 35.97      ,
32.2725    , 39.442     , 36.3925    , 46.148     ,
49.1425    , 57.502     , 46.89      , 11.706     ,
24.592     , 26.13142857, 17.27375   , 11.47333333,
16.63333333, 22.1725    , 17.502     , 19.26714286,
18.0775    , 38.38166667, 45.588     , 36.59      ,
47.235     , 55.01      , 55.43      , 46.48428571,
69.845     , 47.272     , 20.53833333, 37.902     ,
35.67285714, 25.33166667, 24.15333333, 30.4925    ,
24.2025    , 24.505     , 34.305     , 44.02      ,
55.67571429, 29.66142857, 35.11      , 30.874     ,
54.074     , 61.251     , 44.948     , 17.15      ,
48.748     , 40.745     ])
```

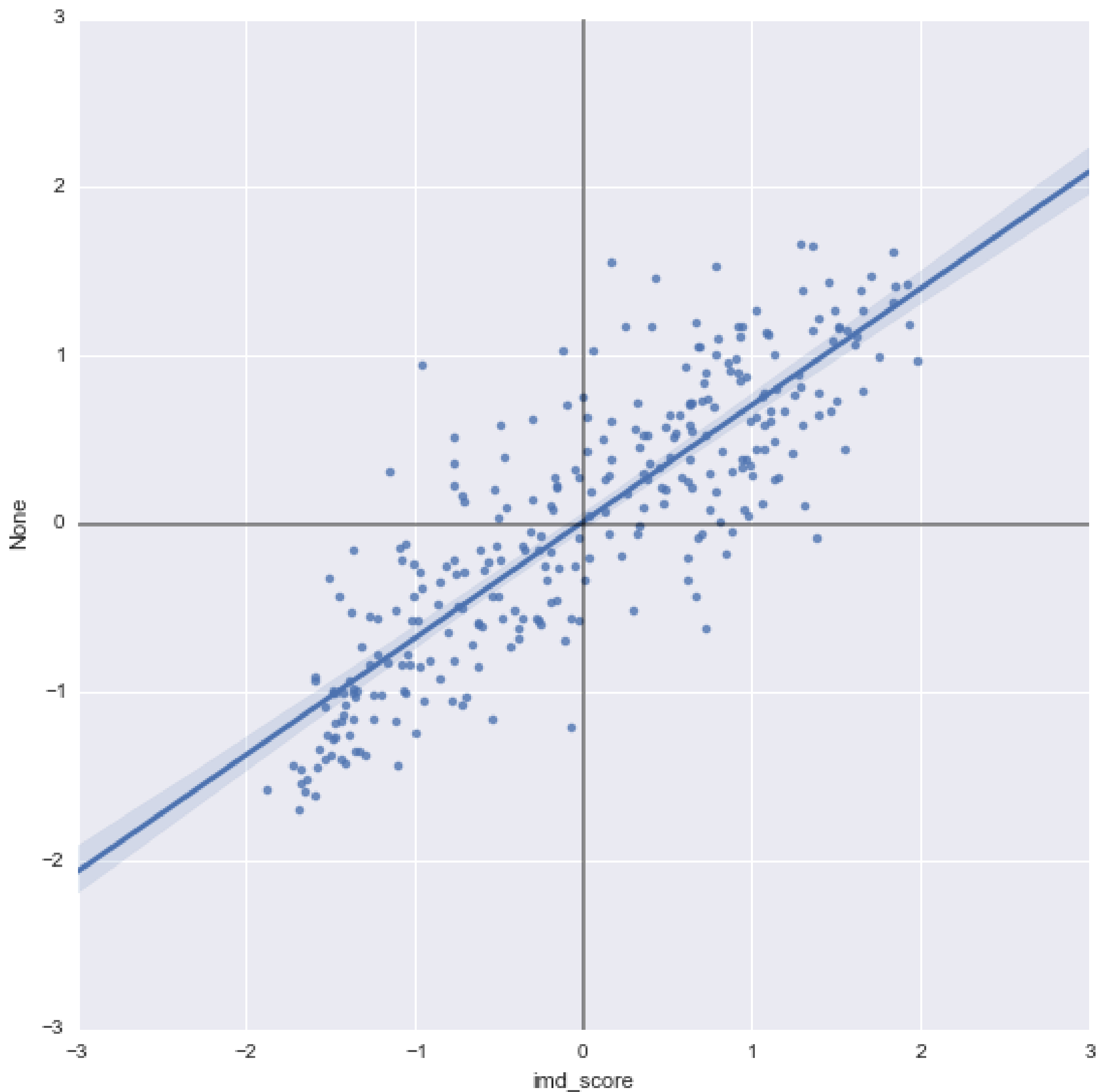
[Optional exercise]
Explore the spatial lag of `w_queen_score` by constructing a density/histogram plot similar to those created in Lab 2. Compare these with one for `imd_score`. What differences can you tell?

1.5 Moran Plot

In [235]:
```
# Setup the figure and axis
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot values
sns.regplot(x="imd_score", y="w_queen_score", data=imd)
# Display
plt.show()
```



In [236]:
```
# Standardize the IMD scores
std_imd = (imd['imd_score'] - imd['imd_score'].mean()) / imd['imd_score'].std()
# Compute the spatial lag of the standardized version and save is as a
# Series indexed as the original variable
std_w_imd = pd.Series(ps.lag_spatial(w_queen, std_imd), index=std_imd.index)
# Setup the figure and axis
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot values
sns.regplot(x=std_imd, y=std_w_imd)
# Add vertical and horizontal lines
plt.axvline(0, c='k', alpha=0.5)
plt.axhline(0, c='k', alpha=0.5)
# Display
plt.show()
```



[Optional exercise]
Create a standardized Moran Plot for each of the components of the IMD:

- Crime
- Education
- Employment
- Health
- Housing
- Income
- Living environment

Bonus if you can generate all the plots with a `for` loop.
Bonus-II if you explore the functionality of Seaborn's `jointplot` (link and link) to create a richer Moran plot.