lab_06

November 2, 2015

1 Spatial autocorrelation and Exploratory Spatial Data Analysis

Spatial autocorrelation has to do with the degree to which the similarity in values between observations in a dataset is related to the similarity in locations of such observations. Not completely unlike the traditional correlation between two variables -which informs us about how the values in one variable change as a function of those in the other- and analogous to its time-series counterpart -which relates the value of a variable at a given point in time with those in previous periods-, spatial autocorrelation relates the value of the variable of interest in a given location, with values of the same variable in surrounding locations.

A key idea in this context is that of spatial randomness: a situation in which the location of an observation gives no information whatsoever about its value. In other words, a variable is spatially random if it is distributed following no discernible pattern over space. Spatial autocorrelation can thus be formally defined as the "absence of spatial randomness", which gives room for two main classes of autocorrelation, similar to the traditional case: *positive* spatial autocorrelation, when similar values tend to group together in similar locations; and *negative* spatial autocorrelation, in cases where similar values tend to be dispersed and further apart from each other.

In this session we will learn how to explore spatial autocorrelation in a given dataset, interrogating the data about its presence, nature, and strength. To do this, we will use a set of tools collectively known as Exploratory Spatial Data Analysis (ESDA), specifically designed for this purpose. The range of ESDA methods is very wide and spans from simpler approaches like choropleths and general table querying, to more advanced and robust methodologies that include statistical inference and an explicit recognition of the geographical dimension of the data. The purpose of this session is to dip our toes into the latter group.

ESDA techniques are usually divided into two main groups: tools to analyze *global*, and *local* spatial autocorrelation. The former consider the overall trend that the location of values follows, and makes possible statements about the degree of *clustering* in the dataset. *Do values generally follow a particular pattern in their geographical distribution*? *Are similar values closer to other similar values than we would expect from pure chance?* These are some of the questions that tools for global spatial autocorrelation allow to answer. We will practice with global spatial autocorrelation by using Moran's I statistic.

Tools for *local* spatial autocorrelation instead focus on spatial instability: the departure of parts of a map from the general trend. The idea here is that, even though there is a given trend for the data in terms of the nature and strength of spatial association, some particular areas can diverege quite substantially from the general pattern. Regardless of the overall degree of concentration in the values, we can observe pockets of unusually high (low) values close to other high (low) values, in what we will call hot(cold)spots. Additionally, it is also possible to observe some high (low) values surrounded by low (high) values, and we will name these "spatial outliers". The main technique we will review in this session to explore local spatial autocorrelation is the Local Indicators of Spatial Association (LISA).

```
In [50]: %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 session, we will use a classic dataset in the history of spatial analysis: the cholera map by Dr. John Snow. His story is well known: thanks to his mapping exercise of the location of cholera deaths in XIXth. Century London, he was able to prove that the disease is in fact transmitted through contaminated water, as opposed to the conventional thinking of the day, which stated that transmission occured through the air.

Although the original data was locations of deaths at the point level, in this case we will access an aggregated version at the street level. This will allow us to try calculating spatial weights matrices for a different but also very common type of data, lines. It is also the spatial unit at which the process we are looking is probably best characterized: since we do not have individual data on house units, but only the location of those who died, aggregating at a unit like the street segment provides a good approximation of the scale at which the disease was occuring and spreading.

All the necessary data are available as a single download from the course website on the following link:

http://darribas.org/gds15/content/labs/data/john_snow.zip

The folder contains the street network, point data for the location of the pumps -one of which was contaminated with cholera- and a polygon file with building blocks from the Ordnance Survey (OS data © Crown copyright and database right, 2015). An explanation of the data sources is provided in the companion text file `README.txt`.

Once you download it and unpack it into your computer, set the path to its location as we have been doing on the previous sessions:

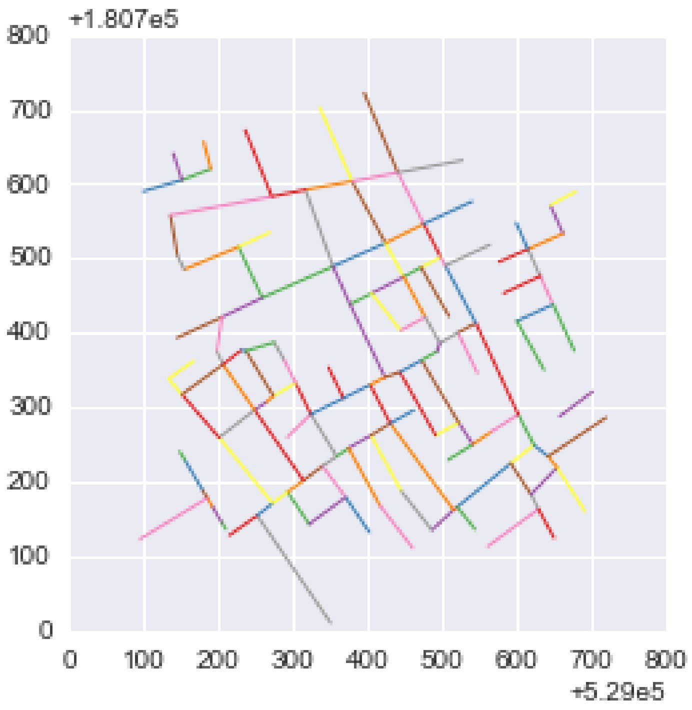
```
In [79]: # This will be different on your computer and will depend on where
         # you have downloaded the files
         js_path = '../../../../data/john_snow/'
```

1.1.1 Loading and exploring the data

Although the data are lines instead polygons, we can load and manipulate them in exactly the same terms as with polygon files:

```
In [80]: # Load point data
         pumps = gpd.read_file(js_path+'Pumps.shp')
         # Load building blocks
         blocks = gpd.read_file(js_path+'polys.shp')
         # Load street network
         js = gpd.read_file(js_path+'streets_js.shp')
         # Quickly plot the streets
         js.plot(colormap='Set1')

Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x112db9d50>
```



And, since it is a full-fledge `GeoDataFrame`, we can also inspect its top rows the same way as with polygons:

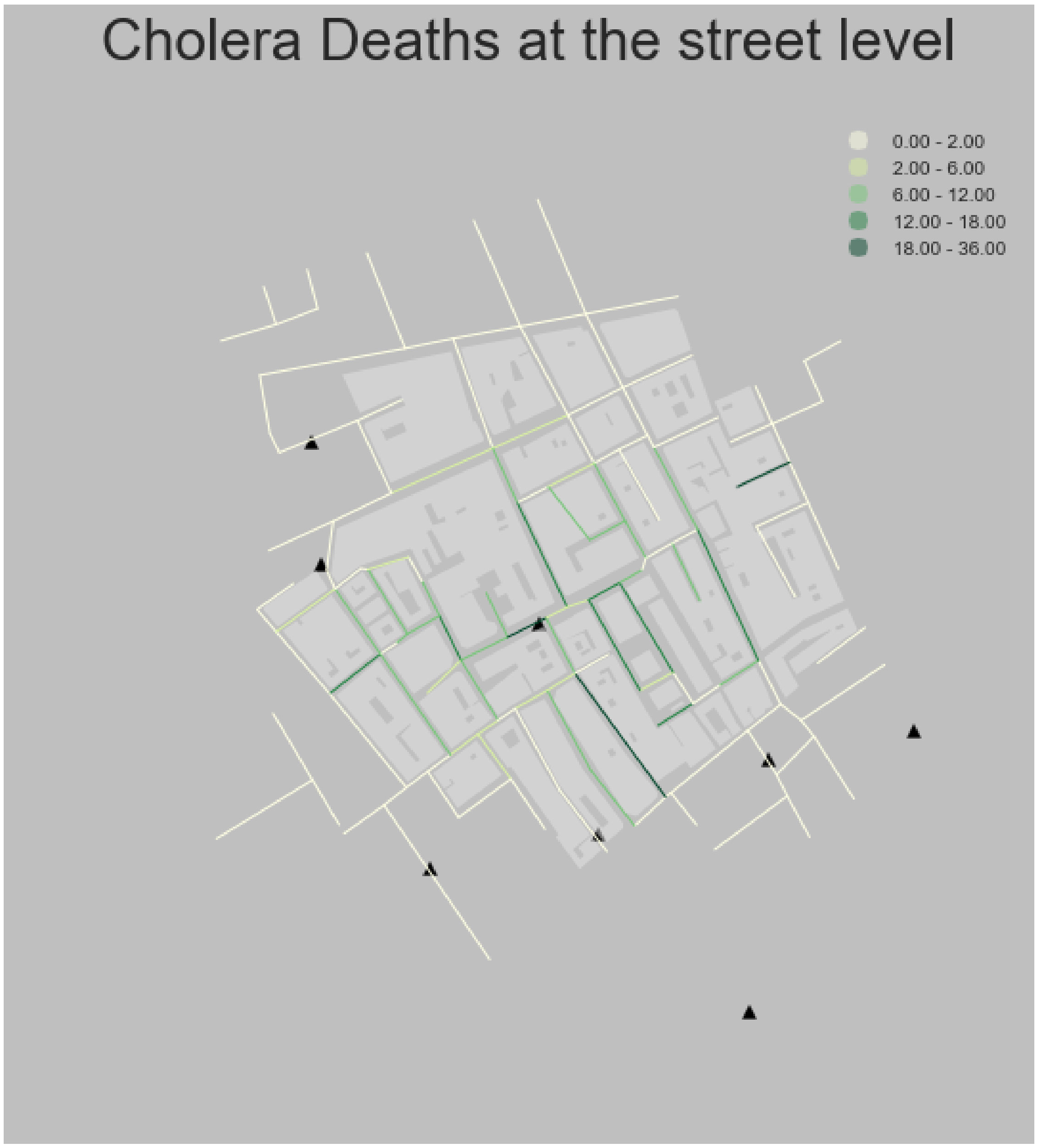
```
In [53]: js.head()
```

```
Out[53]:    Deaths                                            geometry segIdStr
         0       0         LINESTRING (529521 180866, 529516 180862)     s0-1
         1       1  LINESTRING (529521 180866, 529592.98 180924.53)     s0-2
         2       0         LINESTRING (529521 180866, 529545 180836)     s0-3
         3       0         LINESTRING (529516 180862, 529487 180835)    s1-25
         4      26         LINESTRING (529516 180862, 529431 180978)    s1-27
```

Before we move on to the analytical part, we can also create choropleth maps for line data in very much the same way as we have with polygons. In the following code snippet, we build a choropleth using the Fisher-Jenks classification for the count of cholera deaths in each street segment, and style it by adding a background color, building blocks and the location of the water pumps:

```
In [73]: # Set up figure and axis
         f, ax = plt.subplots(1, figsize=(9, 9))
         # Plot building blocks
         for poly in blocks['geometry']:
             gpd.plotting.plot_multipolygon(ax, poly, facecolor='0.9', linewidth=0)
         # Quantile choropleth of deaths at the street level
         js.plot(column='Deaths', scheme='fisher_jenks', axes=ax, \
                 colormap='YlGn', legend=True)
         # Plot pumps
         xys = np.array([(pt.x, pt.y) for pt in pumps.geometry])
         ax.scatter(xys[:, 0], xys[:, 1], marker='^', color='k', s=50)
         # Remove axis frame
```

```
ax.set_axis_off()
# Change background color of the figure
f.set_facecolor('0.75')
# Keep axes proportionate
plt.axis('equal')
# Title
f.suptitle('Cholera Deaths at the street level', size=30)
# Draw
plt.show()
```



[Optional exercise]

Create a similar map as above but using a quantile classification and an equal interval one. How do the maps differ? How do you think the distribution of values is for this dataset? Confirm your hunch by generating a density/histogram plot.

1.1.2 Spatial weights matrix

Spatial weights matrix for street network:

```
In [55]: ntw = ps.Network(js_path+'streets_js.shp')
         w = ntw.contiguityweights(graph=False)
         w.remap_ids(js['segIdStr'])
         w.transform = 'R'
```

```
WARNING: there is one disconnected observation (no neighbors)
Island id:  [(73, 74)]
WARNING:  s70-90  is an island (no neighbors)
```

Spatial lag:

```
In [56]: js['w_Deaths'] = ps.lag_spatial(w, js['Deaths'])
```

Standardized death count:

```
In [57]: js['Deaths_std'] = (js['Deaths'] - js['Deaths'].mean()) / js['Deaths'].std()
```

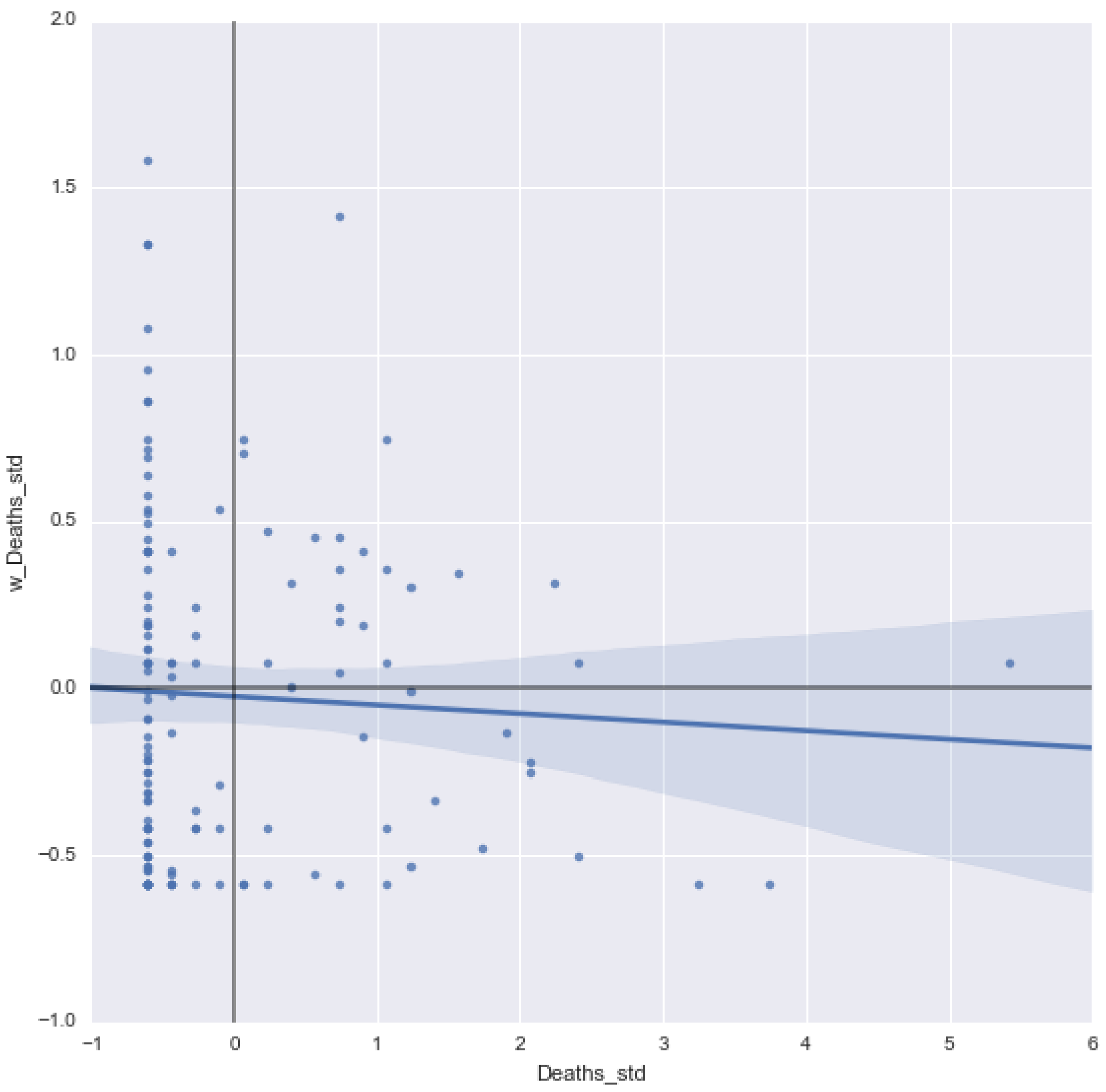
Spatial lag of standardized death count:

```
In [58]: js['w_Deaths_std'] = ps.lag_spatial(w, js['Deaths_std'])
```

1.2 Global Spatial autocorrelation

- Moran Plot

```
In [59]: # Setup the figure and axis
         f, ax = plt.subplots(1, figsize=(9, 9))
         # Plot values
         sns.regplot(x='Deaths_std', y='w_Deaths_std', data=js)
         # Add vertical and horizontal lines
         plt.axvline(0, c='k', alpha=0.5)
         plt.axhline(0, c='k', alpha=0.5)
         # Display
         plt.show()
```



- Moran's I

```
In [60]: mi = ps.Moran(js['Deaths'], w)
```

```
In [61]: mi.I
```

```
Out[61]: -0.026225831288716479
```

- Inference on Moran's I (seaborn band)

```
In [62]: mi.p_sim
```

```
Out[62]: 0.439
```

1.3 Local Spatial autocorrelation

- Moran Plot and Quadrants

- LISAs

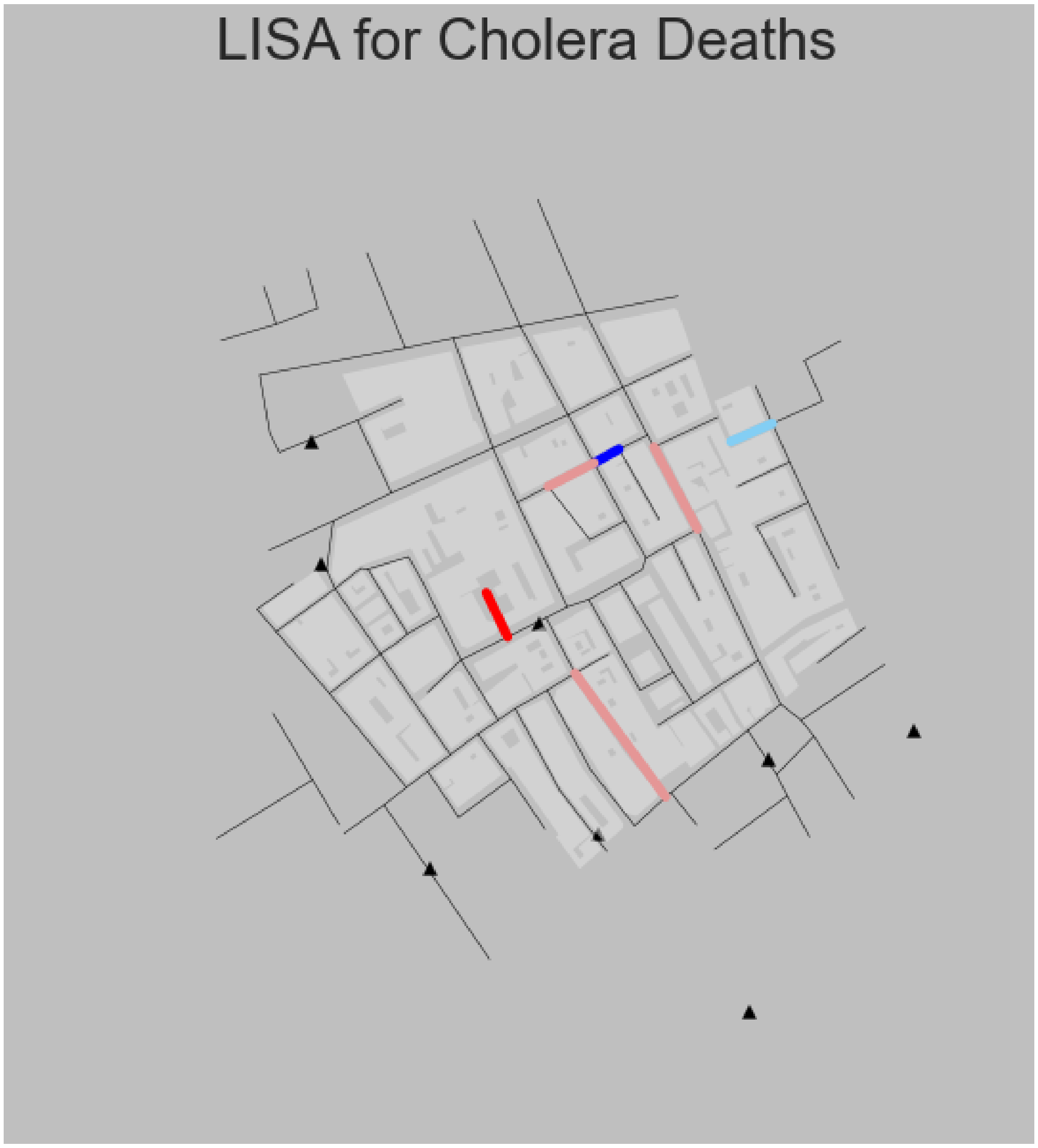
```
In [63]: lisa = ps.Moran_Local(js['Deaths'].values, w)
```

```
In [64]: js['significant'] = lisa.p_sim < 0.05
         js['quadrant'] = lisa.q
```

From PySAL documentation: * 1 HH * 2 LH * 3 LL * 4 HL

- LISA cluster maps

```
In [78]: # Setup the figure and axis
         f, ax = plt.subplots(1, figsize=(9, 9))
         # Plot building blocks
         for poly in blocks['geometry']:
             gpd.plotting.plot_multipolygon(ax, poly, facecolor='0.9', linewidth=0)
         # Plot baseline street network
         for line in js['geometry']:
             gpd.plotting.plot_multilinestring(ax, line, color='k', linewidth=0.5)
         # Plot HH clusters
         hh = js.loc[(js['quadrant']==1) & (js['significant']==True), 'geometry']
         for line in hh:
             gpd.plotting.plot_multilinestring(ax, line, color='red', linewidth=5)
         # Plot LL clusters
         ll = js.loc[(js['quadrant']==3) & (js['significant']==True), 'geometry']
         for line in ll:
             gpd.plotting.plot_multilinestring(ax, line, color='blue', linewidth=5)
         # Plot LH clusters
         lh = js.loc[(js['quadrant']==2) & (js['significant']==True), 'geometry']
         for line in lh:
             gpd.plotting.plot_multilinestring(ax, line, color='#83cef4', linewidth=5)
         # Plot HL clusters
         hl = js.loc[(js['quadrant']==4) & (js['significant']==True), 'geometry']
         for line in hl:
             gpd.plotting.plot_multilinestring(ax, line, color='#e59696', linewidth=5)
         # Plot pumps
         xys = np.array([(pt.x, pt.y) for pt in pumps.geometry])
         ax.scatter(xys[:, 0], xys[:, 1], marker='^', color='k', s=50)
         # Style and draw
         f.suptitle('LISA for Cholera Deaths', size=30)
         f.set_facecolor('0.75')
         ax.set_axis_off()
         plt.axis('equal')
         plt.savefig('../lectures/figs/l06_js_lisa.png')
         plt.show()
```



1.4 Main exercise

- Use IMD data for Liverpool and perform the same analysis

1.5 `[Extension]` Spatial autocorrelation interactive visualization

To do:

- Finish this tutorial
- Add images to lecture slides
- Complete a quick version of main exercise in separate notebook (to send demonstrators).