MeiBau_sheet07

Unknown Author

December 04, 2013

Exercise Sheet 7: K-means Clustering (100 P)

Intro: In this scenario we will simulate finding good locations for warehouses of a multi-national company and some scenarios which can occur. The company is expanding to Europe and surrounding countries and is looking for locations which would maximize its profits. We will attempt to find the number of warehouses and their locations by using k-means clustering. We will look at several scenarios and how they impact the proposed model.

In this exercise we will using data available at: http://sedac.ciesin.columbia.edu/ . The data contains a number of (x,y) coordinates on the map, the number of inhabitants of that section and the corresponding country.

In [1]:
```
import pickle,numpy
import scipy
import scipy.spatial.distance
import scipy.ndimage.filters as flt
import os
import random
import io,IPython,Image
import matplotlib.pyplot as plt
import math
import matplotlib
%matplotlib inline
```

The following helper function reads the data provided. It returns:

- **xnid** : an array of integers indicating the country at each latitude/longitude on the map
- **xpop** : an array of integers counting the population size at each latitude/longitude on the map
- **nx** : the number of latitudes considered
- **ny** : the number of longitudes considered

In [2]:
```
def getData():

    def reading(filename):
        with open(filename, 'rb') as st:
            source = pickle.loads(st.read())
        return source

    xpop = reading('data/pop.pkl')
    xpop = numpy.array([xpop[i::5,j::5] for i in range(5) for j in range(5)]).sum(axis
    nx,ny = xpop.shape


    xnid = reading('data/nid.pkl')
    xnid = xnid[2::5,2::5]
    return xnid, xpop, nx,ny
```

Part 1: Examine the Data and Generate Various Maps (10 P)

In this section we will utilize the data to show 4 different maps:

- A map containing the borders between countries in our data (already provided for you)
- A map where every country is colored differently (or at least assignes different colors to neighbouring countries)

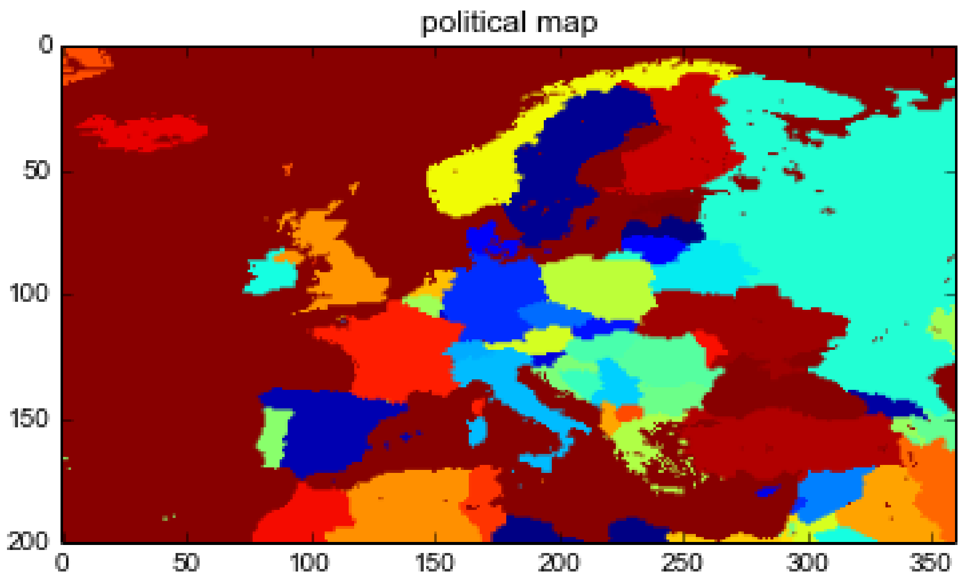
- A map which shows population density in our data
- A map that combines border indicators and population density

Sample maps are given in the folder /maps. Your maps can be generated using the function imshow of the matplotlib library. The last hybrid map will be used in the rest of the exercise sheet.

In [3]:
```
# 1a: Political map
xnid,xpop,nx,ny = getData()
plt.imshow(xnid*(1029743.9384958475)%1)
plt.title('political map')
```

```
<matplotlib.text.Text at 0x4874990>
```

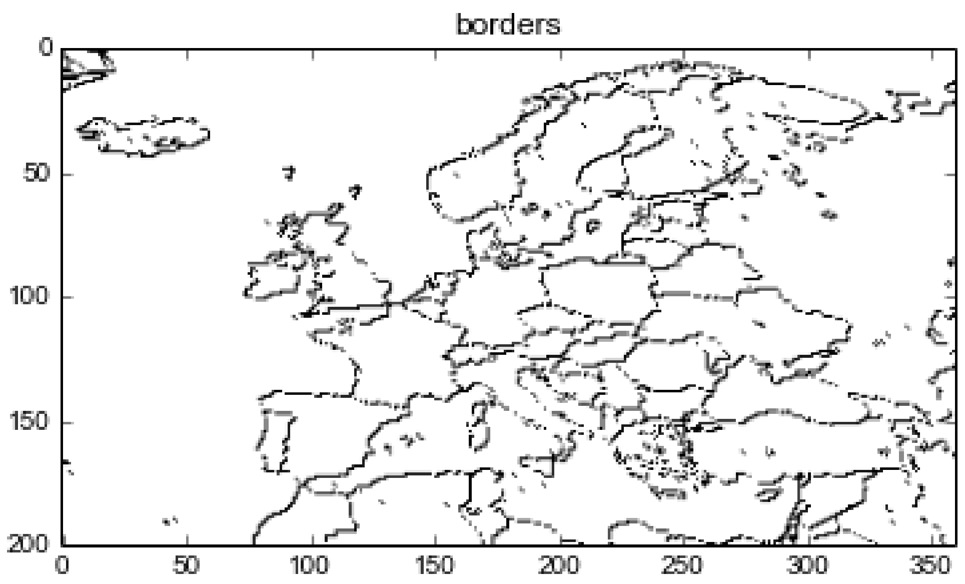
Out [3]:



In [4]:
```
# 1b: Map showing the borders between countries
xnid,xpop,nx,ny = getData()
border_map = numpy.zeros(xnid.shape)
border_map[1:nx,1:ny] = xnid[0:nx-1,0:ny-1]
border_map = border_map - xnid
border_map = border_map != 0
plt.title('borders')
plt.imshow(border_map,cmap='Greys')
```

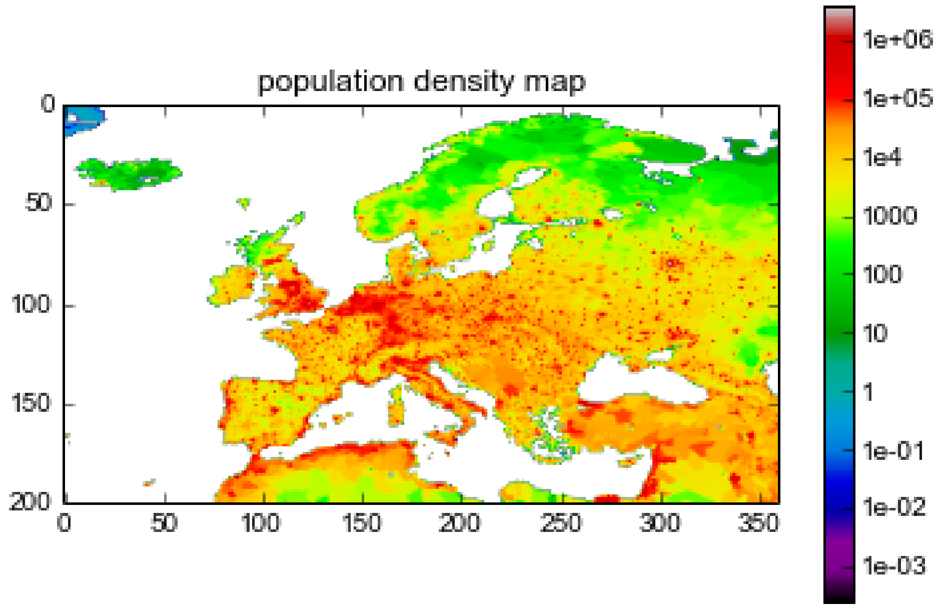
```
<matplotlib.image.AxesImage at 0x4888690>
```

Out [4]:



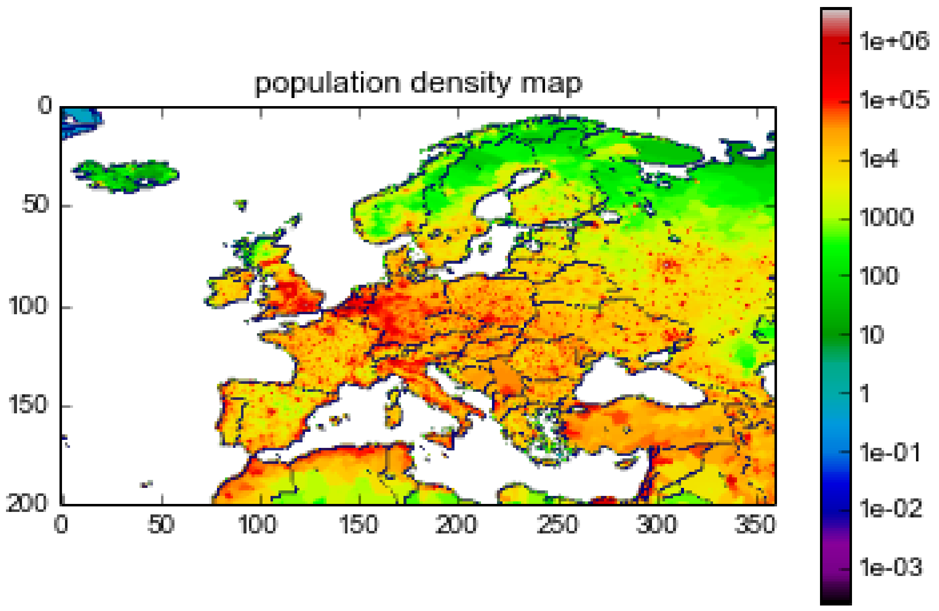
In [5]:
```
# 1c: Map showing population density
xnid,xpop,nx,ny = getData()
imgplot = plt.imshow(xpop, norm=matplotlib.colors.LogNorm())
imgplot.set_cmap('spectral')
plt.colorbar(format=matplotlib.ticker.LogFormatter())
plt.title('population density map')
```

Out [5]:
```
<matplotlib.text.Text at 0x4b51850>
```



In [6]:
```
# 1c: Map superposing population density and borders
xnid,xpop,nx,ny = getData()

imgplot = plt.imshow(xpop, norm=matplotlib.colors.LogNorm())
imgplot.set_cmap('spectral')
plt.colorbar(format=matplotlib.ticker.LogFormatter())

# use masked_where() with already generated border_map to overlay the image
```

```
plt.imshow(numpy.ma.masked_where(border_map == 0, numpy.zeros(border_map.shape)))

plt.title('population density map')
plt.show()
```



Part 2: Implementing one step of weighted k-means (20 P)

Your task is to implement the k-means algorithm using weighted data. Here, instead of considering each inhabitant as a separate point, we consider each geographic coordinate as an inhabitant with a different weight corresponding to the number of people living at this coordinate.

- **initial centroids**: the x and y coordinates of initial centroids
- **points** : x and y coordinates of points with number of inhabitants > 0
- **weights** : Containing the number of people living in every section

The function should output the following:

- **final centroids** : the final centroids after many iterations
- **cost function** : the value of the cost function $J(c) = \sum_{x_i \in \text{points}} w_i ||x_i - c(x_i)||_2^2$ where $c(x_i)$ denotes the centroid associated to x_i and w_i is the weight of each data point x_i.

Do not restrict your solution to only two dimensions. Avoid loops when possible.

In [15]:
```
def wkmeans(centroids, points, weights):
    """
    It is given a dataset points where each row is a single data point, a vector
    idx of centroid assignments (i.e. each entry in range [1..K]) for each
    example, and the number of centroids. You should return a matrix
    centroids, where each row of centroids is the mean of the data points
    assigned to it.
    """
    J = 0
    # <points, points>
    p_v = points[:,0]**2 + points[:,1]**2
    # <centroids, centroids>
    c_v = centroids[:,0]**2 + centroids[:,1]**2

    # <points, centroids>
    pc = points.dot(centroids.T)
    error_matrix = numpy.outer(p_v, numpy.ones(c_v.shape)) - 2 * pc + numpy.outer(nump

    idx = numpy.argmin(error_matrix, axis=1)

    J = sum(numpy.amin(error_matrix, axis=1) * weights)

    for c_i,c in enumerate(centroids):
        inter = idx == c_i
        if inter.any():
            centroids[c_i] = numpy.average(points[inter], axis=0, weights=weights[inte


    return centroids,J
```

Part 3: Building an initialization heuristic (20 P)

Here, you are requested to implement an heuristic for choosing initial centroids. The idea is the following: We assign a random score $s(i, j)$ to each geographical coordinate (i, j). The score is computed as $s(i, j) = w(i, j) - N$ where $w(i, j)$ is the number of inhabitants at a certain location and N is a random variable drawn from an exponential distribution of scale parameter λ to be determined. Then, the k locations with largest score are assigned a centroid.

(a) Implement the initialization heuristic

(b) Generate a map superposing the centroids to the population density map

(c) Test different parameters λ and k

Comment:

The convention is that $\lambda = \frac{1}{\beta}$ is named the rate parameter which is the inverse of the scale Parameter β. But because you explicitly called λ to be the scale parameter, we will not calculate any β or something but use λ directly

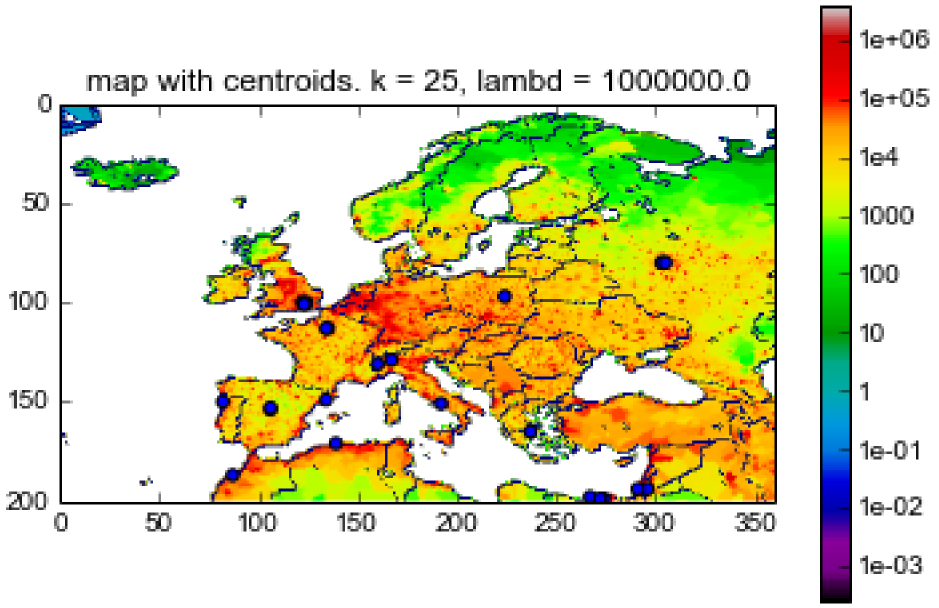
discribed here

In [8]:
```
def initialize(pop, k, lambd):
    dim = pop.shape
    N = numpy.random.exponential(lambd,dim)
    score = pop - N
    centroids = numpy.zeros((k,2))
    for l in range(k):
        i,j = numpy.unravel_index(score.argmax(), dim)
        centroids[l] = numpy.array([i,j])
        score[i,j] = -99999999999999
    return centroids
```

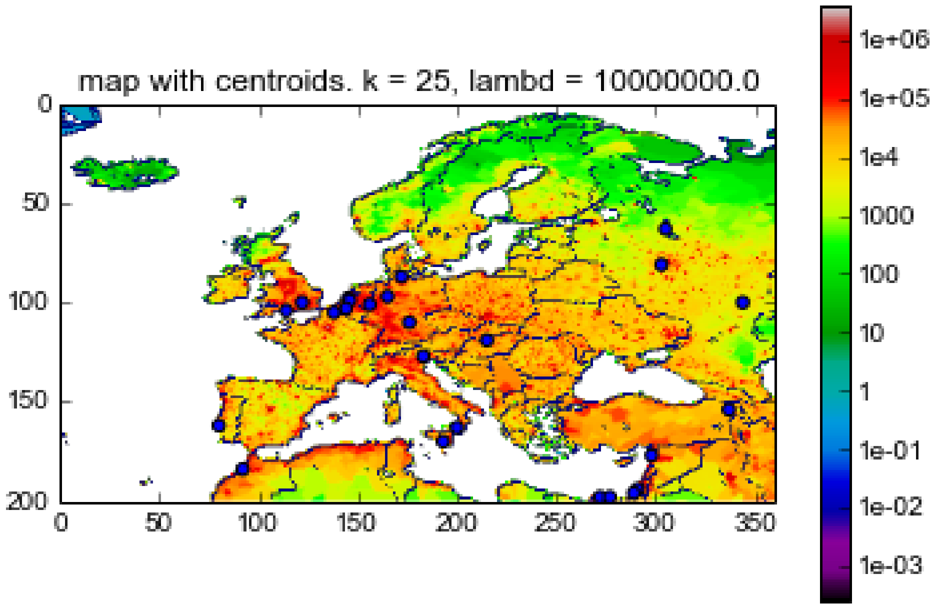
In [9]:
```
def draw_centroids_map(centroids):

    # new figure
    plt.figure()

    # plot population
    imgplot = plt.imshow(xpop, norm=matplotlib.colors.LogNorm())
    imgplot.set_cmap('spectral')
    plt.colorbar(format=matplotlib.ticker.LogFormatter())

    # use masked_where() with already generated border_map to overlay the image
    plt.imshow(numpy.ma.masked_where(border_map == 0, numpy.zeros(border_map.shape)))

    # plot centroids
    plt.scatter(centroids[:,1], centroids[:,0])

    # scatter will add a extra range to the axis. Undo it.
    plt.gca().set_xlim(0, 360)
    plt.gca().set_ylim(200, 0)
    plt.title('map with centroids. k = ' + str(k) + ', lambd = ' + str(lambd))

    plt.show()
    plt.close()
```

In [10]:
```
# test with different parameters
xnid,xpop,nx,ny = getData()
for k in range(25, 100, 25):
    for lambd in numpy.logspace (6, 8, 3):
        print "k: ", k, ", lambd: ", lambd
        centroids = initialize(xpop, k, lambd)
        draw_centroids_map(centroids)
```

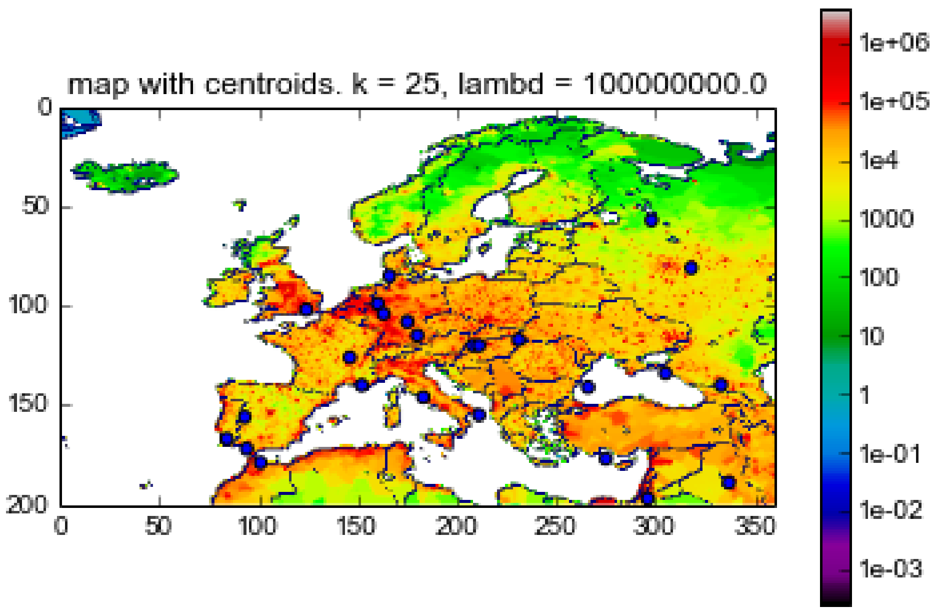
```
k:  25 , lambd:  1000000.0
```



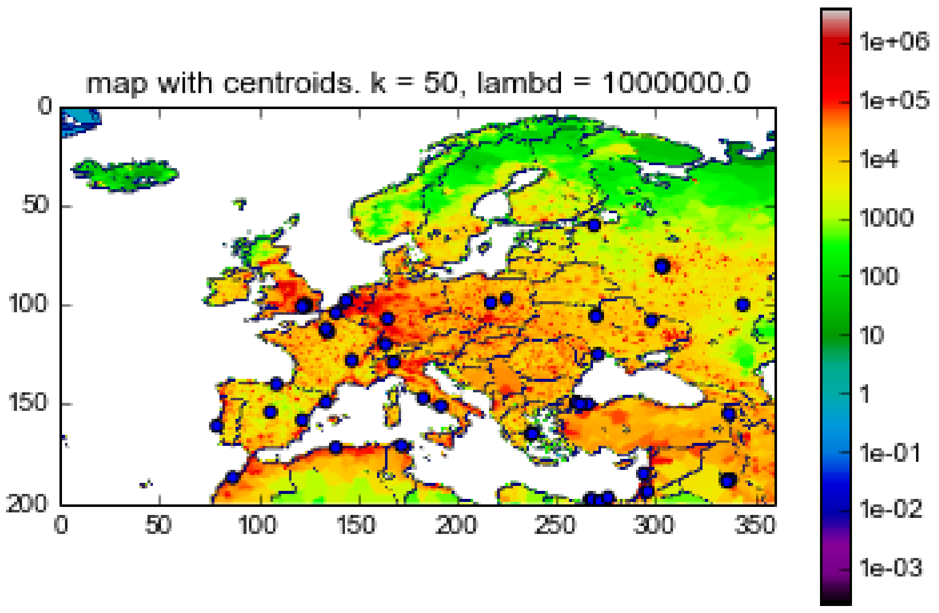
```
k:  25 , lambd:  10000000.0
```



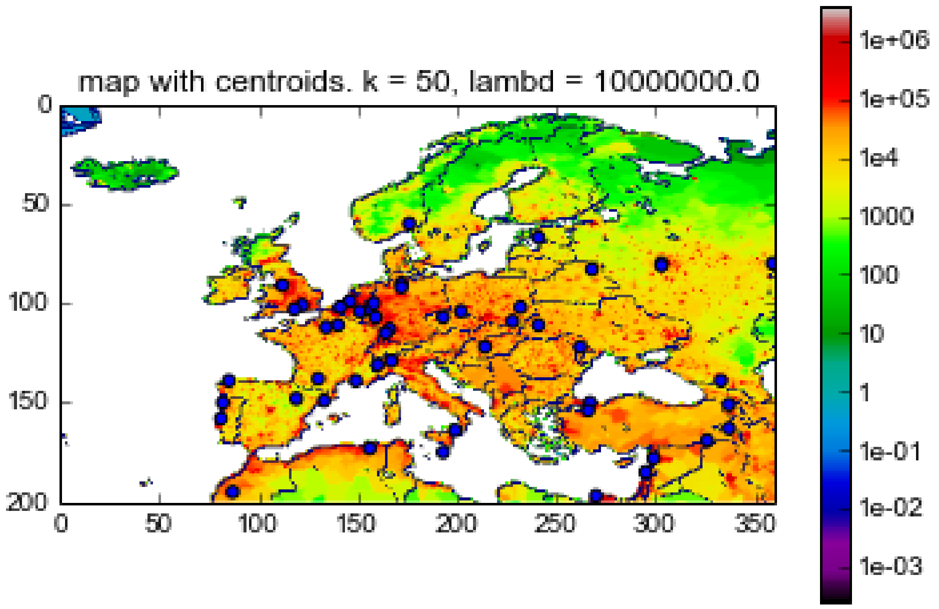
```
k:  25 , lambd:  100000000.0
```



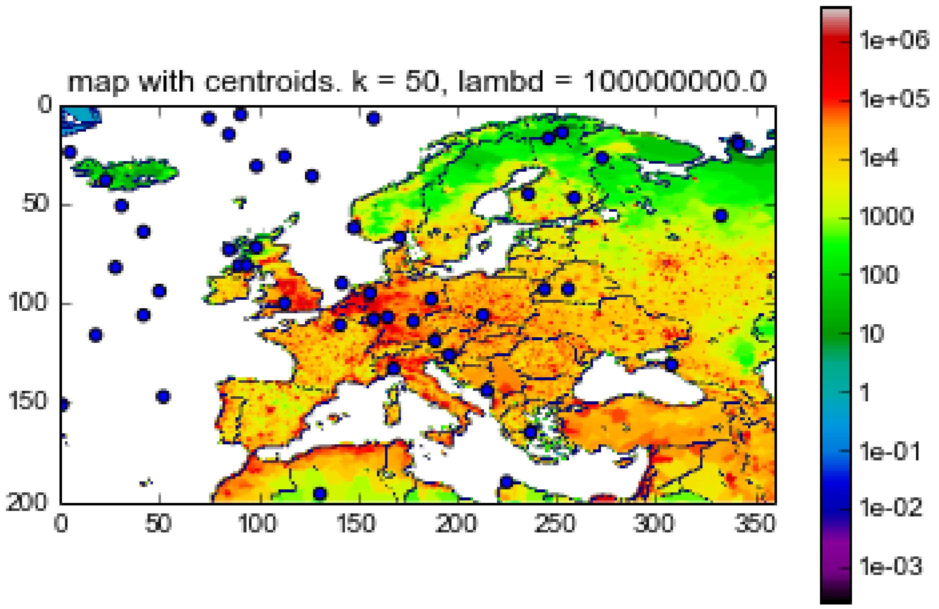
```
k:  50 , lambd:  1000000.0
```



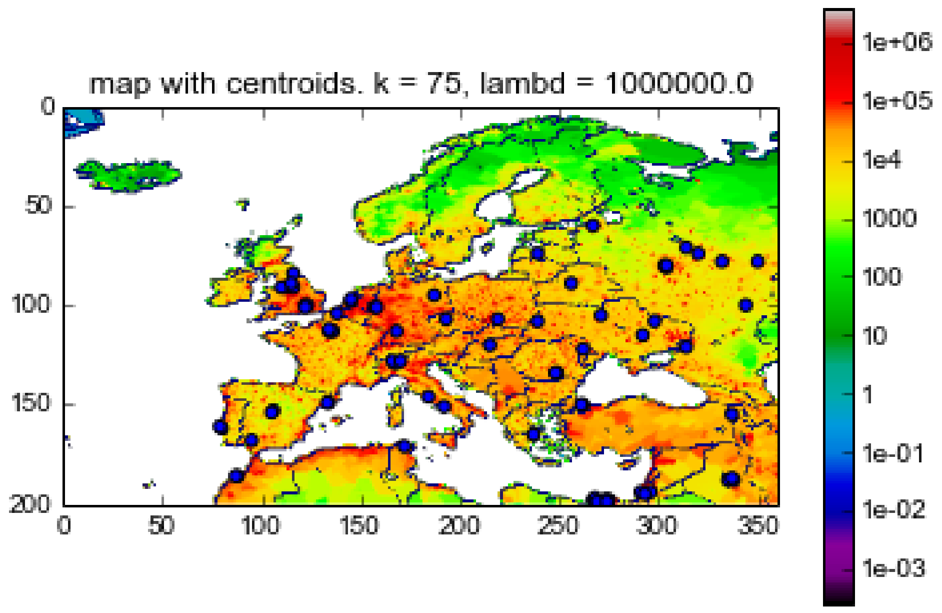
```
k:  50 , lambd:  10000000.0
```



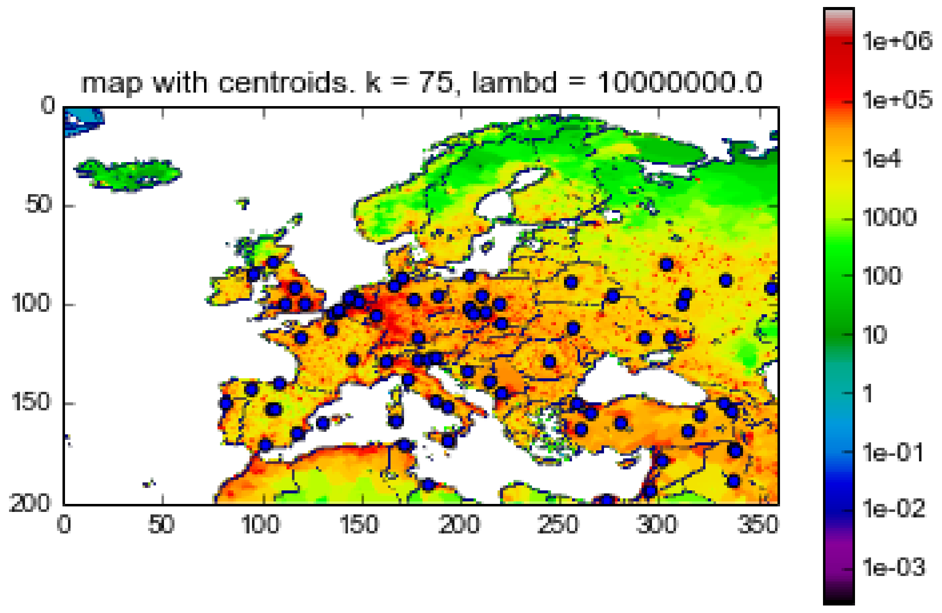
```
k:  50 , lambd:  100000000.0
```



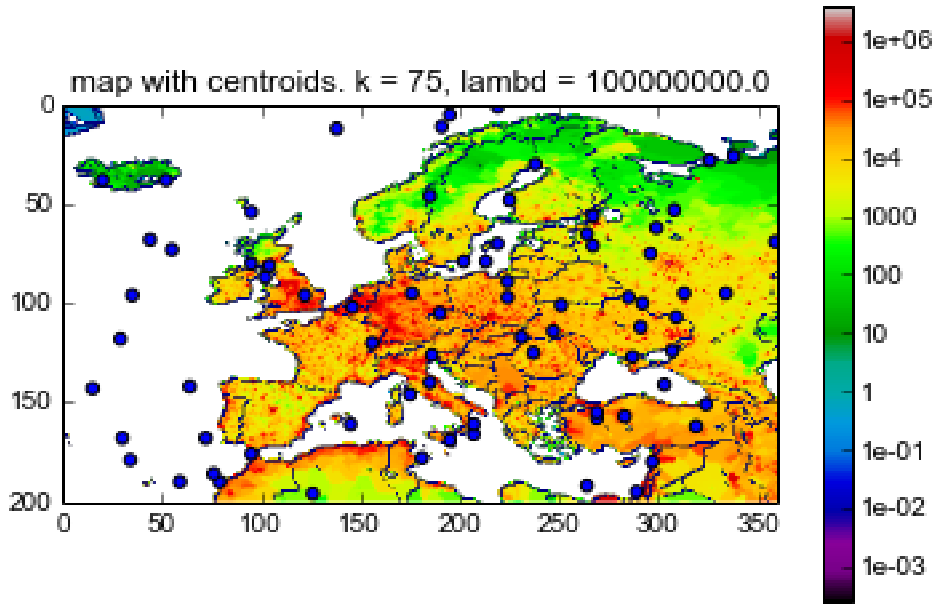
```
k:  75 , lambd:  1000000.0
```



```
k:  75 , lambd:  10000000.0
```



```
k:  75 , lambd:  100000000.0
```



Part 4: Running weighted k-means (20 P)

The company wishes to find good locations of warehouses. We assume that the same percentage of inhabitants of every country uses their services, so the number of users can be approximated with the number of inhabitants. We assume that the cost of transporting their products to users is proportional to the squared euclidian distance to the closest warehouse.

(a) Run the k-means algorithm with the number of clusters $k = 100$. Save your results, as they will be used later. Run at least 20 iterations of k-means.

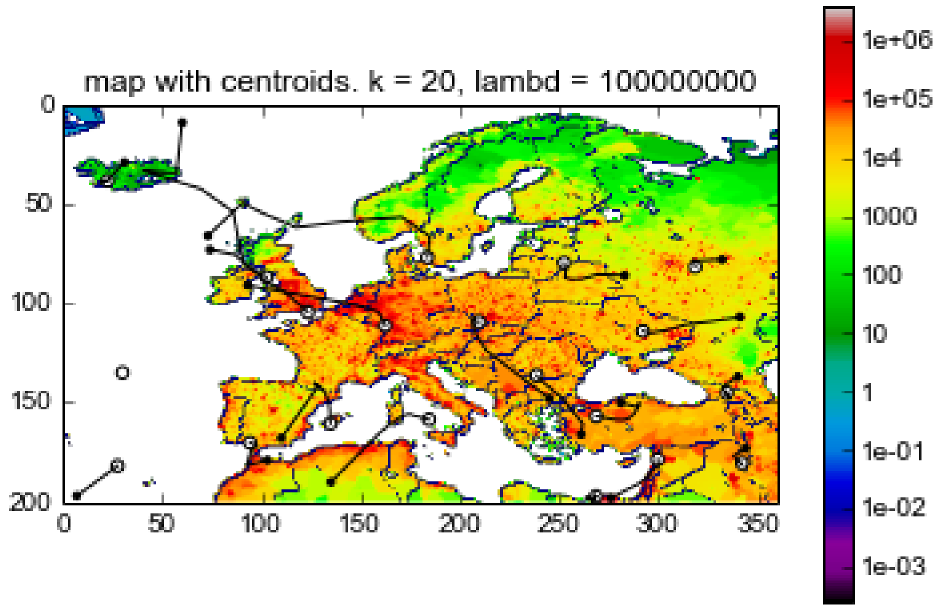
(b) Plot the path of the centroids throughout the training procedure (for example, the initial centroids as a black dot, the final centroids as a white dot and the path that they followed as a black line). The path should be plotted in superposition to the hybrid map built in Part 1.

In [11]:
```
def draw_centroids_list_map(c_list):

    # new figure
    plt.figure()

    # plot population
    imgplot = plt.imshow(xpop, norm=matplotlib.colors.LogNorm())
    imgplot.set_cmap('spectral')
    plt.colorbar(format=matplotlib.ticker.LogFormatter())

    # use masked_where() with already generated border_map to overlay the image
    plt.imshow(numpy.ma.masked_where(border_map == 0, numpy.zeros(border_map.shape)))

    # plot initial centroids
    plt.scatter(c_list[0][:,1], c_list[0][:,0], c='black', s=6)
    #plt.scatter(c_list[-1][:,1], c_list[-1][:,0], c='white')
    # plot line
    c_array = numpy.array(c_list)

    #plt.plot(c_array[:][:][1], c_array[:][:][0], color='k', linestyle='-', linewidth=
    for i in range(c_array.shape[1]):
        plt.plot(c_array[:,i,1],c_array[:,i,0], color='k', linestyle='-', linewidth=1)

    plt.scatter(c_list[-1][:,1], c_list[-1][:,0], c='white')
    # scatter will add a extra range to the axis. Undo it.
    plt.gca().set_xlim(0, 360)
    plt.gca().set_ylim(200, 0)

    plt.title('map with centroids. k = ' + str(k) + ', lambd = ' + str(lambd))

    plt.show()
    plt.close()
```

In [22]:
```
# RUN K-MEAN

iterations = 10
lambd = 100000000
k = 20

xnid, xpop, nx,ny = getData()

points = numpy.transpose(numpy.nonzero(xpop))
weights = xpop[points[:,0], points[:,1]]

#Initialize centroids
centroids = initialize(xpop, k, lambd)
c_list = [numpy.array(centroids)]
# start k-means
for i in range(iterations):
    print "iteration "+str(i)
    centroids, J = wkmeans(centroids, points, weights)
    c_list.append(numpy.array(centroids))
```

```
iteration 0
iteration 1
iteration 2
```

In [23]:
```
# Visualize results
draw_centroids_list_map(c_list)
```



Part 5: Focusing on the German market (10 P)

We have determined that our initial assumption that the number of customers in a country is proportional to its number of inhabitants is incorrect. Analysis has shown that customers in Germany (xnid=111), Austria(xnid=104) and Switzerland (xnid=109) are three times more likely to buy the product than customers in other countries.

(a) Explain how the model should be modified to take into account this new constraint

(b) Run k-means on this new problem with $k = 50$ and show the same type of map as in Part 5 for this new setting.

(c) Describe the qualitative change in the allocation of factories accross Europe.

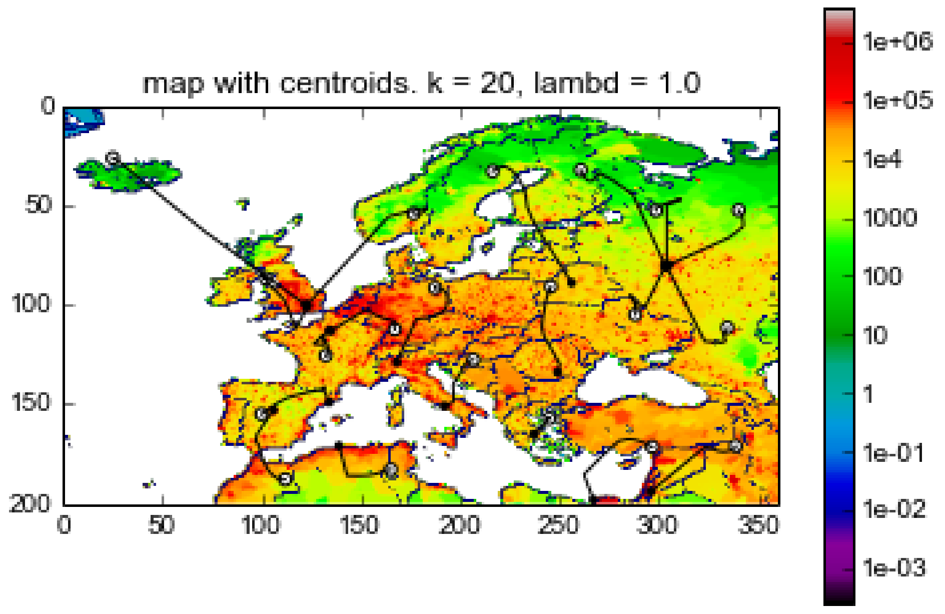
a) in order to take this new contraint into account we have to weight the points located in the defined countries more than we do normal points. to accomplish this we generate a array the same shape as points filled with ones if no weighting is providet and the points weight otherwise

In [24]:
```
lambd = 1.
iterations = 50

xnid, xpop, nx,ny = getData()
points = numpy.transpose(numpy.nonzero(xpop))
weights = numpy.ones((points.shape[0]))

#Initialize centroids
centroids1 = initialize(xpop, k, lambd)
c_list1 = [numpy.array(centroids1)]

# start k-means with no weight at all (every entry is 1)
for i in range(iterations):
    print '.',
    centroids1, J1 = wkmeans(centroids1, points, weights)
    c_list1.append(numpy.array(centroids1))

# a := final centroids rounded pos
a = numpy.array(numpy.round(c_list1[-1]),dtype=numpy.int)
# sum of centroids located in germany
loc_a = xnid[a.T[0],a.T[1]]
sum_a = sum(loc_a == 111) + sum(loc_a == 104) + sum(loc_a == 109)

print "\nnumber of factories without weights: "+str(sum_a)

#Initialize centroids
centroids = initialize(xpop, k, lambd)
c_list = [numpy.array(centroids)]

#apply new weights:
countries = xnid[numpy.nonzero(xpop)]
weight = 3
weights[countries[:] == 111] = weight
weights[countries[:] == 104] = weight
weights[countries[:] == 109] = weight

# start k-means
for i in range(iterations):
    print '.',
    centroids, J = wkmeans(centroids, points, weights)
    c_list.append(numpy.array(centroids))

# a := final centroids rounded pos
a = numpy.array(numpy.round(c_list[-1]),dtype=numpy.int)
# sum of centroids located in germany
loc_a = xnid[a.T[0],a.T[1]]
sum_a = sum(loc_a == 111) + sum(loc_a == 104) + sum(loc_a == 109)

print "\nnumber of factories with weights: "+str(sum_a)
```

c) as printed out there are more factories located in german speaking area if this area is weighted more. if the weighting is increased even more factories will be located there. and as centroids do not multiply in time, there will be less factories in the rest of this map.

b)

In [26]:
```
# Visualize results
draw_centroids_list_map(c_list)
```



Part 6: Shipping restrictions (10pt)

Due to new shipping regulations in the EU, transporting the product across borders is now taxed heavily.

(a) Explain how the new problem can be understood as a k-means problem in a higher-dimensional space (i.e. extending the original longitude/latitude two-dimensional space to more dimensions).

(b) Implement the higher-dimensional k-means problem and produce a map similar to the one produced in Part 5.

(c) Explain what are the qualitative difference between the distribution of factories as a function of the level of cross-border taxation.

a) imagining the map in three dimension were the country-borders are like mountains, points located on these mountains would have a greater distance to each other. instead of really implement more dimensions and increase computation times, we use the weights array as this new dimensions. also because heavy weights are actually good, we have to 'dig out' cliffs instead of mountains

c) because of the taxiation the centroids are more dense in the center of the countries. there is a plot were we actually implemented the difference, its like one gets money at the borders. in this plot one can clearly see how the centroids are all lining up for the taxes ;)

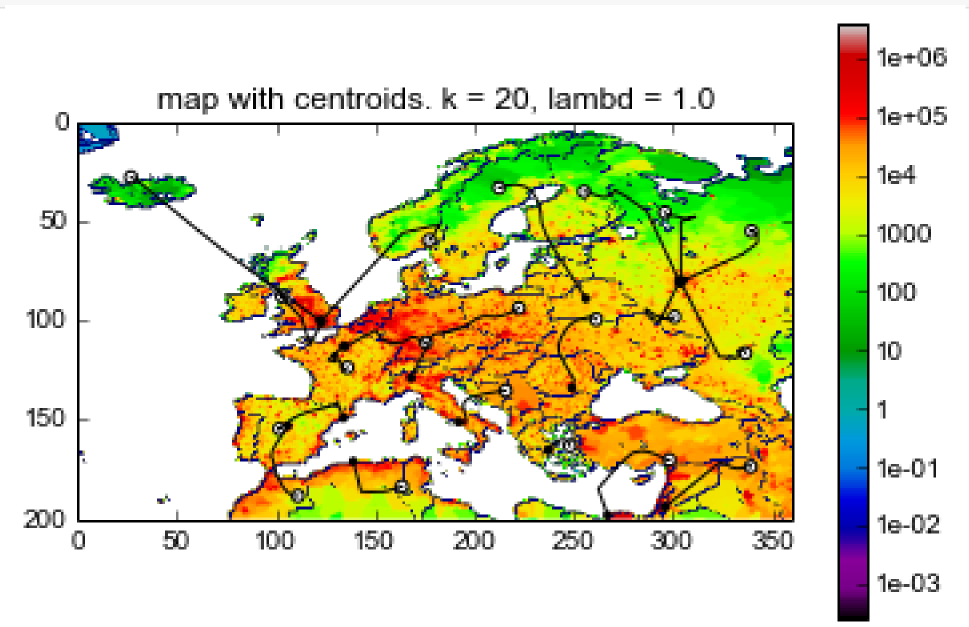
In [27]:
```
"""
Implementation as stated in the Exercise
"""

lambd = 1.
iterations = 50

xnid, xpop, nx,ny = getData()
points = numpy.transpose(numpy.nonzero(xpop))
weights = numpy.ones((points.shape[0]))

#Initialize centroids
centroids = initialize(xpop, k, lambd)
c_list = [numpy.array(centroids)]


#apply new weights:
```

```
image = numpy.zeros(border_map.shape) + 10
image[border_map] = 1
image = flt.gaussian_filter(image, 2, mode='nearest')

weights = image[numpy.nonzero(xpop)]
#weight = 3
#weights[border_weight[:] != 0] = weight

# start k-means
for i in range(iterations):
    print '.',
    centroids, J = wkmeans(centroids, points, weights)
    c_list.append(numpy.array(centroids))
```

```
. .
```

b)

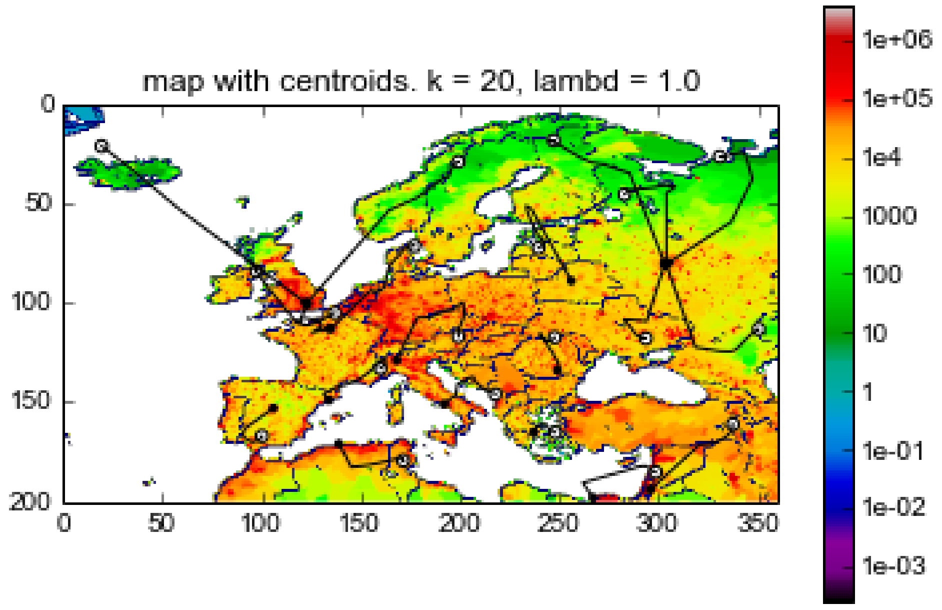
In [28]:
```
# Visualize results
draw_centroids_list_map(c_list)
```



In [20]:
```
"""
Implemetation of the exact opposit for better visual support of the qualitative effect
"""

lambd = 1.
iterations = 50

xnid, xpop, nx,ny = getData()
points = numpy.transpose(numpy.nonzero(xpop))
weights = numpy.ones((points.shape[0]))

#Initialize centroids
centroids = initialize(xpop, k, lambd)
c_list = [numpy.array(centroids)]


#apply new weights:
image = numpy.zeros(border_map.shape)
image[border_map] = 10
image = flt.gaussian_filter(image, 2, mode='nearest')
```

```
weights = image[numpy.nonzero(xpop)]
#weight = 3
#weights[border_weight[:] != 0] = weight

# start k-means
for i in range(iterations):
    print '.',
    centroids, J = wkmeans(centroids, points, weights)
    c_list.append(numpy.array(centroids))
. .
```

In [21]:
```
# Visualize results
draw_centroids_list_map(c_list)
```



Part 7: Discussion (10pt)

Discuss in approximately three paragraphs what are the advantages and limitation of the k-means model to the problem of optimal resource allocation. The discussion could cover points such as validity of the Euclidean distance as a measure of shipping cost, temporal evolution of consumer demand, supply side constraints. Then, discuss how the basic model could be extended to these more realistic scenarios.**ANSWER**

First of all the k-means algorithm may not give us the optimal solution - even if we can ignore side constraints. To solve this problem we could run k-means several times (with different initial centroids) and then chose the solution with the lowest error J.

Another problem is that companies usually don't decide where to build let's say 50 warehouses at once. Usually they have 49 warehouses already and want to decide where to build number 50. (This problem could be solved by minimizing the cost for just one point - very easy to implement)

The euclidean distance is bad when considering local constraints. For example Poland and Sweden seem to be near in terms of distance, but shipping the goods with a farry is for sure more expensive. A realistic extention for the company in question could be to pre-define possible locations of warehouses or cities as knots and set some distances between the knots manually or by a suitable map software.

To take (temporal evolving) consumer demand into account, we first have to meassure the demand by some kind of market research. Then we could easily multiply this demand with the population to get a better weight.

Another simplification we made is that we assume that all warehouses are equaly big. It may be sufficient to build a smaller warehouses in not so populated regions.Submission guidelines

To facilitate grading, please export the notebook to PDF format. This can be done easily by installing the required packages and running

Do not upload the data

ipython nbconvert –to latex sheet07.ipynb && pdflatex sheet07.tex