

# CS1112 Spring 2011 Project 5    due Thursday 4/14 at 11pm

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, “you” below refers to “your group.” You may discuss background issues and general strategies with others, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not ok for you to see or hear another student’s code and it is certainly not ok to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on your own, please seek help from the course staff.

## Objectives

Completing this project will solidify your understanding of numeric array, character array, cell array, structure, and structure array. You will also work with text data files.

## More Debate on the Census Data

Census data are important for many reasons; one among them is the impact on resource allocation by cities and states for serving the local population. Housing and zoning policies, as well as economic incentives, at the city and county levels can be considered “tools” for shifting local population over time. You will use census data to investigate population density-based thresholds for categorizing zipcode areas and applying policies to effect population shift.

We consider a model where each zipcode area in a region is to be classified as rural, urban, and super urban. This model considers rural and most urban land to be desirable, but urban areas that are extremely densely populated will be classified as “super urban” and policies will be put in place that cause a fraction of the population to shift to a rural area. Our model specifications include the following:

- The threshold for classifying a zipcode to be rural is 1000 persons per square mile, i.e., a density less than or equals 1000 indicates rural.
- A non-rural zipcode is urban.
- A set of candidate thresholds for categorizing an urban zipcode to be “super urban” is being studied; they range from 8000 to 30000 persons per square mile.
- Policies can be applied to cause a percentage of the population in a “super urban” zipcode to shift to the nearest rural zipcode. One such percentage can be used in the model; it should be less than 20%.

## 1 Getting the Data

We use the census data from 2000 since the newest census data are not yet in an easily usable form. Download from the course website the files `nyzipcodes.dat`, `NEzipcodes.dat`, and `ReadMe.txt`. `ReadMe.txt` explains the format of the data in the two `.dat` data files; be sure to read it. `nyzipcodes.dat` is the short file containing data on all zipcodes in New York state while `NEzipcodes.dat` contains that zipcode data for the Northeast region. Use the short data file for initial program development and later run your program using the long data file as well.

Define a zipcode structure (struct) by implementing this function:

```
function Z = MakeZipcode(zip, pop, area, dens, lat, long)
% Z is a Zipcode structure such that
%   Z.zip = zip (string, i.e., char array) is the zipcode, e.g., '14853'
%   Z.pop = pop (double) is the population of the zipcode
%   Z.area = area (double) is the land area of the zipcode in sq. miles
%   Z.density = dens (double) is the density of people in the zipcode
%   Z.latitude = lat (double) is the latitude of the zipcode
%   Z.longitude = long (double) is the longitude of the zipcode
```

This function is short; don't be alarmed! Now that we have a definition for a zipcode struct, implement the following function to read a data file and return two arrays of zipcode structs:

```
function [ruralZip, nonRuralZip] = readZipcodes(zipFile, rThresh)
% Read data from the file named by zipFile and return two struct arrays:
% ruralZip is a 1-d struct array containing all the zipcode structs of
% rural zipcodes.
% nonRuralZip is a 1-d struct array containing all the zipcode structs of
% non-rural zipcodes.
% A zipcode is rural if its population density is less than or equal to rThresh.
```

Review Lecture 20 notes and use the built-in functions discussed to write this function. Make effective use of function MakeZipcode.

## 2 Finding the Balance

Now you can start implementing the following function:

```
function optimalThresh = zipcodeBalancer(zipFile, rThresh, suThresh, p)
% Determine the optimal threshold for classifying zipcodes as super urban
% and produce two output data files.
% Parameters:
%   zipFile is the filename of the zipcode data to be analyzed.
%   rThresh is the upper limit for population density of a rural zipcode.
%   suThresh is a vector of lower limits for population density of a super
%   urban zipcode. Each value in suThresh is a candidate threshold; this
%   function determines which threshold is optimal and returns that value
%   as optimalThresh.
%   p is the percentage of a super urban zipcode's population that will
%   move to the nearest rural zipcode.
```

This is the main function that solves the central problem of this project. The “optimal” threshold for the super urban category is determined by trying out all the candidate thresholds in the vector **suThresh**, one at a time, on the data. For each candidate super urban threshold determine the population shift and the resulting percent decrease of super urban zipcodes and percent decrease of rural zipcodes. The optimality of each candidate super urban threshold is then scored as the difference

$$\text{Percent Decrease in Super Urban Zipcodes} - \text{Percent Decrease in Rural Zipcodes}$$

The rationale for the optimality score is that a decrease in super urban zipcodes is the desired outcome but a loss of rural area is undesirable. The super urban threshold with the highest score is the optimal threshold.

Let's look at some example output data files before discussing the calculation details. One of the two files to be produced is **thresholdData.dat**, which records how well each candidate threshold performs. Here is an example where 6 candidate thresholds are evaluated:

Threshold: 10000	Super urban decrease: 5.7%	Rural decrease: 0.6%	Score: 5.1%
Threshold: 14000	Super urban decrease: 7.2%	Rural decrease: 0.5%	Score: 6.8%
Threshold: 18000	Super urban decrease: 7.2%	Rural decrease: 0.3%	Score: 6.9%
Threshold: 22000	Super urban decrease: 9.5%	Rural decrease: 0.3%	Score: 9.2%
Threshold: 26000	Super urban decrease: 7.0%	Rural decrease: 0.2%	Score: 6.8%
Threshold: 30000	Super urban decrease: 6.1%	Rural decrease: 0.2%	Score: 5.9%

The optimal super urban threshold for the above example is 22000, since it results in the highest score. The second output data file to be produced is **optThresholdData.dat**, which shows the details on how many people move from where to where over what distance given the optimal super urban threshold. There is one line of output for each super urban zipcode. Several lines in the file are shown below:

Zipcode 08625 will shift 183 people to zipcode 08505 which is a distance of 0.0834 away  
 Zipcode 10001 will shift 1731 people to zipcode 10020 which is a distance of 0.0181 away  
 Zipcode 10002 will shift 8487 people to zipcode 10170 which is a distance of 0.0374 away  
 Zipcode 10003 will shift 5367 people to zipcode 10170 which is a distance of 0.0243 away  
 Zipcode 10006 will shift 144 people to zipcode 10170 which is a distance of 0.0584 away

Below we discuss the steps for completing function `zipcodeBalancer`.

## 2.1 Distance between zipcodes

Function `zipcodeBalancer` starts by calling function `readZipcodes` to obtain the two struct arrays (rural and non-rural zipcodes). Before trying each candidate threshold value in `suThresh` as described above, you need to first locate for each non-rural zipcode the nearest rural zipcode (since any population shift goes to the nearest rural zipcode). For each non-rural zipcode, determine the nearest rural zipcode and store its index and the distance in one row of numeric 2-d array `nearestRural`:

- In column 1 store the *index* of the nearest zipcode. The index is the position in the `ruralZip` struct array.
- In column 2 store the distance to nearest zipcode. You can simplify distance calculation by treating the longitude and latitude values as x and y coordinates on a Cartesian plane.

The number of rows in `nearestRural` is the number of non-rural zipcodes.

Hints for speeding up computation: (1) Accessing a field in a struct in a struct array takes more time than accessing a value in a simple array. Notice that you need to use each rural zipcode's longitude and latitude multiple times. So instead of making repeated access to those fields, loop through the rural struct array once and store the longitude and latitude values in numeric arrays. (2) You may find the built-in function `min` helpful in this way: `[minVal, idx] = min(v)` returns in `minVal` the smallest value in numeric vector `v` and in `idx` the index where the `minVal` occurs. If `minVal` occurs multiple times in `v` then `idx` is a vector of indices.

## 2.2 Working one candidate threshold value at a time

Implement function `balancePopulation` to deal with the tasks of identifying super urban zipcodes, shifting population, and tallying the result *for one* super urban threshold value. Then later call `balancePopulation` for every candidate threshold value that needs to be evaluated.

```
function [fileData, numSuperUrban, numRural] = balancePopulation(...
    nonRuralZip, ruralZip, nearestRural, suThreshValue, rThresh, p)
% Given a super urban threshold, identify the super urban zipcodes and for
% each shift p% of the population to its nearest rural zipcode.
% Parameters:
%   nonRuralZip is a struct array of non-rural zipcodes
%   ruralZip is a struct array of rural zipcodes
%   nearestRural is a numeric matrix; each row corresponds to a non-rural
%       zipcode: col 1 is the index of the nearest rural zipcode and col 2 is
%       the distance to the nearest rural zipcode. The index is the nearest
%       zipcode's position in struct array rural
%   suThreshValue is the super urban threshold, a scalar
%   rThresh is the rural threshold
%   p is the percentage of a super urban zipcode's population that will move
% Return Parameters:
%   fileData is a cell array of the population shift data. Each row
%       corresponds to one super urban zipcode. There are four cells in each
%       row: (1) super urban zipcode (string), (2) nearest rural zipcode (string),
%       (3) distance between the two zipcodes, (4) number of people moved to
%       nearest rural zipcode
%   numSuperUrban is how many super urban zipcodes there are at the end
%   numRural is how many rural zipcodes there are at the end
```

## 2.3 Optimal threshold and output data files

Back in function `zipcodeBalancer`, you can call `balancePopulation` which you've just completed to evaluate every candidate threshold in `suThresh`. Identify the optimal threshold based on the score and write the output files as discussed in §2.

Here's an example fragment for running the function:

```
zipFile= 'NEzipcodes.dat';  
rThresh= 1000;  
suThresh= 10000:4000:30000; % Try other reasonable values!  
p= 10; % Try other reasonable values!  
optSuperUrbanThresh= zipcodeBalancer(zipFile, rThresh, suThresh, p)
```

Submit your files `MakeZipcode.m`, `readZipcodes.m`, `balancePopulation.m`, and `zipcodeBalancer.m` on CMS.