NENS 230
Assignment 4: Data Visualization

Due date: Tuesday, October 20, 2015

Goals

- Get comfortable manipulating figures
- Familiarize yourself with common 2D and 3D plots
- Understand how color and colormaps work in Matlab

Problem 1: Simulated EEG

In this problem, you will visualize traces of simulated data of a model that mimics normal and abnormal EEG activity. The data, contained in `sim_eeg.mat`, consists of a single time vector `t` and a matrix of four voltage vectors `v`, where each column is from a different simulation.

1. First, plot the four different voltage traces found in `v` as four different subplots in a figure. Use a 2x2 subplot (two rows and two columns).
2. Add labels for the axes (time and voltage). The units are arbitrary in this simulation.
3. Add a title to each subplot based on what that trace is. The titles for the columns are, in order: "Background Activity", "Sporadic Spikes", "Rhythmic Activity", and "Slow quasi-sinusoidal activity".
4. Edit the figure to make the following cosmetic changes (either through code or with the Plot Tools feature):
   (a) Use thicker lines
   (b) Change the color of the axes to be a dark gray. Change the color of the voltage traces to a different RGB color (pick one that you like).
   (c) Turn the grid on for all four subplots. Set the tick direction to point outward by running the following command: `set(gca,'TickDir','out');` This sets the `TickDir` property of the current axes (`gca` stands for get current axes, and returns a *handle* to the current axes).

When you are done, export your plot as an `png` or `pdf` file. Turn in any code you wrote for this part along with the figure. If you edited the plot with Plot Tools, make a note that you did so in a comment in your code. Your output should look something like this:

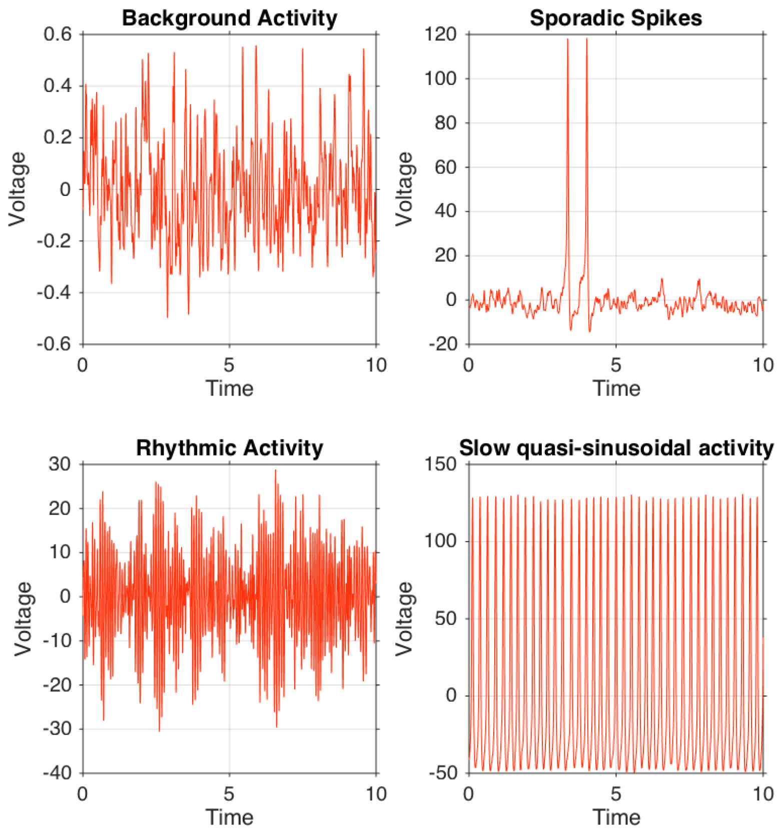

Figure 1: default

Problem 2: Countries by population

For this problem, we will play with data consisting of population estimates of 239 countries. The data is located in `populationData.mat` which has been uploaded to coursework, and was obtained from the CIA World Factbook (https://www.cia.gov/library/publications/the-world-factbook/rankorder/2119rank.html). The `.mat` file contains three vectors: a list of ranks in `rank`, the corresponding name of the country in `country`, and the corresponding population in `population`. Note that the names are stored as a cell array, another type of data structure.

(a) Most populous countries

First, make a bar graph showing the populations of the five most populous countries (these should just be the first five entries in the `population` vector). Divide the numbers by the appropriate scaling factor so that the populations presented in the graph are in billions. See if you can change the default color of the bars to something lighter. Also, label the x-axis tick marks with the actual names of the five countries from the `country` vector (the easiest way may be through Plot Tools, but you can also do it from the command line). Finally, add an appropriate `ylabel` and `title` to your plot.

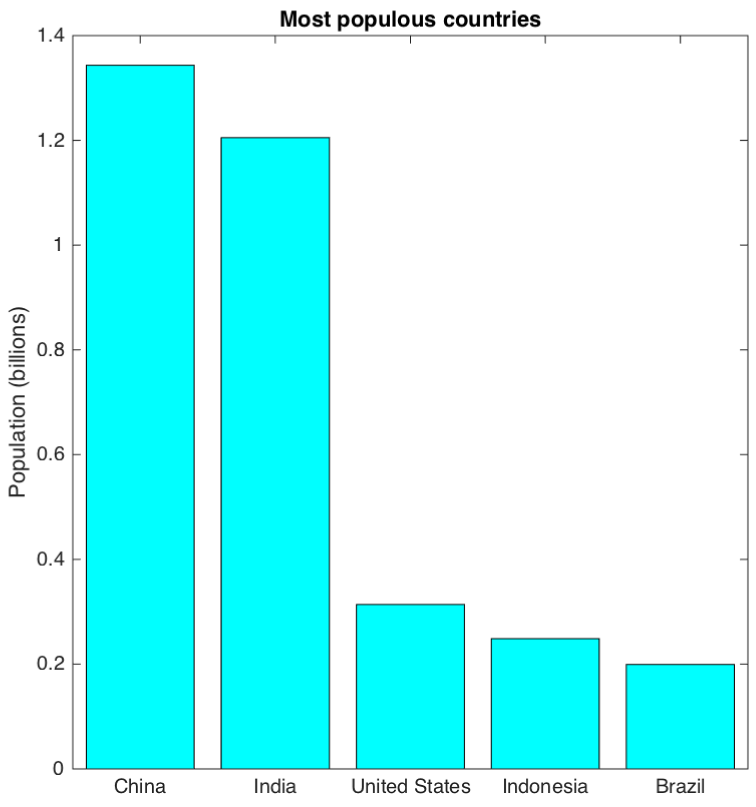

Figure 2: default

(b) Benford's Law

We are now going to use the population data to explore an interesting phenomenon known as Benford's law. Benford's law states that in numbers obtained from many real-life sources of data, the frequency of the leading digit (1-9) follows a specific non-uniform distribution. Intuitively, I think most people would expect that if we got a bunch of numbers from a data set, we would expect roughly the same number of numbers to start with an 8, or a 2, or a 1, for example. However, this is not the case. We will explore this by computing the frequency of leading digits in the population of different countries and comparing this to the distribution predicted by Benford's law. If you haven't heard of Benford's law before, feel free to read up on it on Wikipedia (https://en.wikipedia.org/wiki/Benford%27s_law), because it is pretty interesting (at least I find it interesting).

1. First, make a vector called `digits` which consists of the numbers from 1 to 9 (easiest to do this with colon notation). You should also initialize a 1x9 vector called `digit_frequency` which will store the frequencies of the different digits, you can make this using the `zeros` command.

2. We need to figure out a way to compute the frequency of leading digits in the `population` vector. We will do this with a `while` loop inside a `for` loop. Write a `for` loop to loop over each element of the `population` vector. This should loop over the 239 elements of the vector. In the for loop, store the current element of the `population` vector as a value called `num`. We will perform a manipulation on this number to extract its first digit.

3. If you are given a number, how can you determine what the first digit is? Well, if you know that the number if a decimal between 1 and 10, you can just chop off the decimal (using the `floor` command) to extract just the ones place. If the number is between 10 and 100, you can first divide by 10, and then chop off the decimal. Similarly, if the number is between 100 and 1000, you can divide by 100, and then chop off at the decimal. Hopefully this is making sense. If not, try running the following in the command window: `floor(469/100);` This divides the number 469 by 100, which yields 4.69. The `floor` command chops off the decimal and we are left with the first digit, 4. Now the question becomes: how do we know what number to divide by (how big our initial number is)? To deal with this problem, we will divide in increments of 10, and stop when our number is less than 10. Write a `while` loop that checks if the value in `num` is less than 10. Inside the while loop, divide the number `num` by 10 and store the result back into `num`.

4. OK we should step back a second and make sure we know what is going on. The for loop indexes over every number in our `population` vector. For each number, we need to extract the first digit. We first store the full number as a variable named `num`. Then, using the `while` loop, we divide `num` by 10 until it is less than 10. The output of this should be a number between 1 and 10 where the ones place is the leading digit of the original number. Finally, after the while loop (but still in the for loop!), take the `floor` of num. This should yield a single number corresponding to the leading digit of the population. We want to keep track of how many times we have seen each digit. Increment the corresponding index of the vector we created to store digits counts, `digit_frequency`, by the following line of code: `digit_frequency(num) = digit_frequency(num) + 1;` After the for loop has finished, this vector should store the number of times each digit was found to be a leading digit.

5. One last thing that we need to do is convert the counts in the `digit_frequency` vector to actual frequencies (probabilities). To do this, divide the `digit_frequency` vector by the length of the `population`

vector (and remember to store it back into `digit_frequency`!).

Plot the distribution of digits using the `bar` command. You will want to plot the frequencies you stored in `digit_frequency` against the digits 1-9 stored in `digits`. Feel free to change the color of the bars to something other than deep blue. You should see that the most common leading digit is a 1, and the distribution drops off very quickly. Finally, we would like to compare this empirical distribution to the prediction given by Benford's law. Benford's law predicts that the distribution of leading digits is given by:

$$p(d) = \log_{10}\left(1 + \frac{1}{d}\right)$$

Where d is a digit, 1-9. Compute this equation over the digits given in the `digits` vector (1-9) and store the result in a new vector, `prediction`. You should be able to do this in one line of code, make sure to use dot-divide (`./`) for the fraction and use `log10` instead of just `log` to get the right logarithm (base 10 instead of base e). Use `hold on` to keep the bar plot on the figure, and then plot the prediction on top of the bars as a colored line with stars as line markers. Add labels, a title, and a legend to finish off the figure. Submit both your code and final figure for this problem.

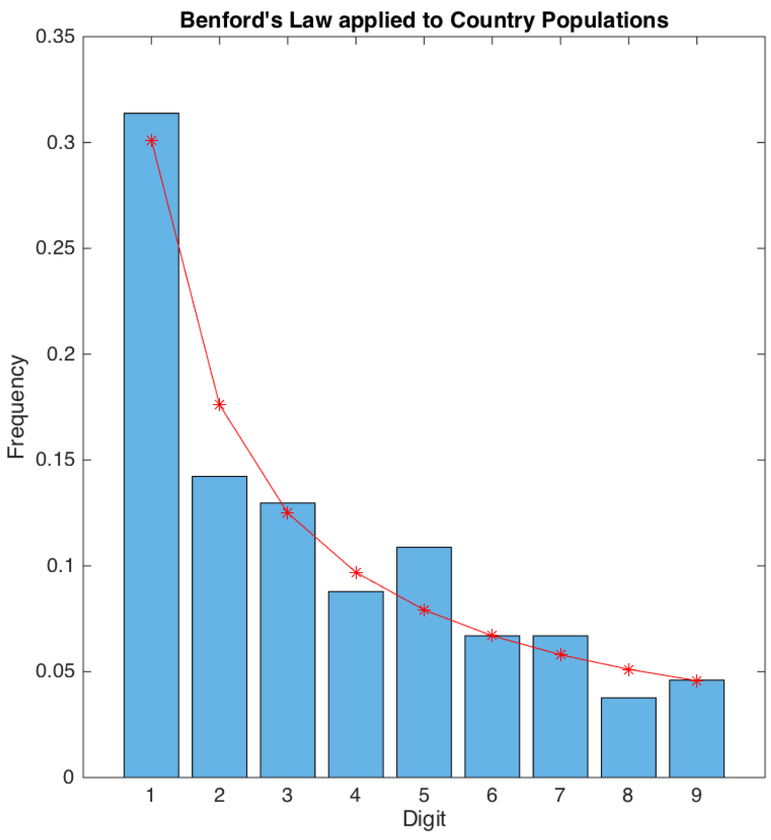

Figure 3: default

Problem 3: 3D visualization

For this problem, you will compare a few different ways to visualize three-dimensional data. First, load the built-in `peaks` dataset by running the following command: `[x,y,z] = peaks(25);` This loads data which consists of matrices of `x` and `y` points, with corresponding heights given in the `z` matrix. You are going to visualize the data in the following ways (make sure to turn in your final plots):

1. Plot the surface using the `pcolor` command (you will probably have to run `help pcolor` to get the syntax (make sure that you use all three variables: `x`, `y` and `z`). How is this different from the `image` command? (answer this question as a comment in your code).

2. Now try plotting the data with the `surf` command. Try out the different `shading` and `lighting` options and pick one you like. You may need to check out `help shading` and `help lighting` for more information. Change the colormap to something that isn't `jet` or `gray`.

3. OK we are now going to explore a command that is used to visualize vector fields: `quiver`. First, we are going to generate a matrix containing the estimated slopes of the peaks data. To do this, run the following command: `[u,v] = gradient(z);` This stores the x- and y-components of the slope vectors in the matrices `u` and `v`, respectively. To visualize the vector field, use the `quiver` command. When done correctly, you should see a vector field with little arrows pointing towards the minima of the peaks data. To get really fancy, let's plot both the vectors and contours. Use `hold on;` to keep your arrows on the plot, and then use the `contour` command to add contours to the plot (note that you only need `x`, `y` and `z` as arguments to the `contour` command).

4. Bonus: plot the data using one of the following: `waterfall`, `ribbon`, or `contour3`.

Export the 3-4 plots you made above as JPGs and send them in along with your code.

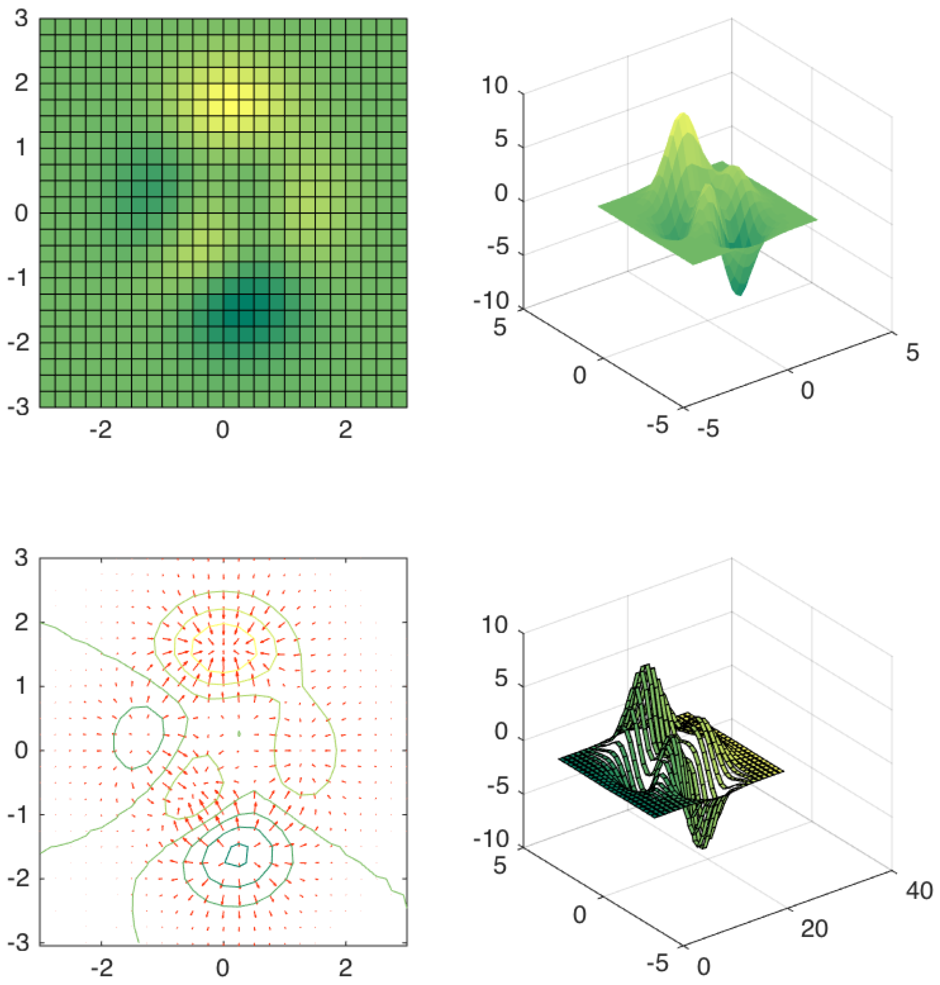

Figure 4: default

Submission

When you are finished, **don't** publish your work with the `publish` command, but instead, zip up the folder containing your code (m-files) and your generated figures. Email this to nens230@gmail.com to submit your assignment with the tag (written exactly) `[Assignment 4]` in the subject line. Make sure it is clear which figures/code correspond to which part of the assignment.