# STATS 607A (Fall '14): Assignment 3
# Due: Nov 16, 2014 23:59:59

Ambuj Tewari

Oct 26, 2014

This version: 3.0 (Problem 3 added)

## Ways to earn extra credit

- +1 for all real bugs in the already supplied code that you find and report to the instructor.
- +1 for each problem where your code is in the top 10 percentile (top 5 students) in terms of running time.
- +1 for each python script where Python style guide checker `pep8` doesn't report any issues. You can test for any style guide issues yourself by typing `pep8 <python-script>` on the shell prompt.

## How to turn in?

Make sure a directory called `stats607-fall2014` (case-sensitive) exists in the `Public` directory already present in your home directory on any of the Bayes machines. If it is not so already, please make it readable for the instructor and GSI by typing:
`fs sa ~/Public/stats607-fall2014/ -acl tewaria read kamwong read`
on the Bayes command prompt. Then create a sub-directory called `assignment_three` within the 607 directory. Make that readable too for us by typing:
`fs sa ~/Public/stats607-fall2014/assignment_three/ -acl tewaria read kamwong read`
We will try to find the following files in your home directory after the submission deadline:
`assignment_three_rmt.py`
`assignment_three_animate.py`
≪ 1 more filename to be added in a later version of this document ≫
`assignment_three_answers.pdf`

The `.py` files should be python scripts that run without any error message. The last file should be a PDF file with answers to all questions below. Make sure the directories containing the files above are all readable to us. You can check whether permissions are ok by typing:

```
fs listacl ~/Public/stats607-fall2014
fs listacl ~/Public/stats607-fall2014/assignment_three
```

You should see 2 lines in the output (in each case) that say: (under "Normal rights:")
`tewaria rl`
`kamwong rl`

# 1 Verifying the Semicircle and Tracy-Widom Laws from Random Matrix Theory [RMT] (10 points)

In this problem, we will numerically verify two important RMT laws following the algorithmic recipes mentioned in Section 1 of the following article:
http://www-math.mit.edu/~edelman/publications/random_matrix.pdf
We will basically write the equivalent Python code for Algorithm 1 in the article linked to above.

The only difference is that we will NOT implement Algorithm 2 that computes the Tracy-Widom distribution. Instead, we will simply read a table containing pre-computed numerical values from the file:
tracy-widom.csv
We will award +5 extra credits to anyone who figures out a way to compute the Tracy-Widom distribution in Python/numpy/scipy. It shouldn't be too difficult given the functions in `scipy.special` (for the Airy function) and `scipy.integrate` (for ODE solvers).

Download an (incomplete!) python script by following the following link:
assignment_three_rmt.py
Click on the "Raw" button on the top right and save the file as `assignment_three_rmt.py`.

Open the python script in your favorite text editor. You will see a bunch of places where a comment says `TASK x.y(.z)`. These are the places where you have to supply your own code.
**Important:** Please only change/add code where the tasks require you to do so (sometimes you'll have to replace a `pass` statement with real code, sometimes you'll be changing existing statements). Please *don't* modify the existing code and comments anywhere else! We might use automated scripts to grade your code and not following this suggestion will break those scripts.

We will now briefly describe your tasks. If something is unclear, please don't hesitate to email the instructor and/or GSI.

## 1.1 Sampling from GOE and computing eigenvalues (2 points)

Create a 2D $n \times n$ ndarray $A$ (`a` in the code) whose entries are iid standard normal. Then set $S = (A + A^\top)/2$ (`s` in the code). The random matrix $S$ is said to be drawn from the GOE (Gaussian Orthogonal Ensemble). GOE, GUE (the one we'll see in the next TASK) and GSE (Gaussian Symplectic Ensemble) are three classical random matrix ensembles. See, for example,
http://en.wikipedia.org/wiki/Random_matrix#Gaussian_ensembles
Once the symmetric matrix $S$ has been generated, store its $n$ real eigenvalues in a row of the 2D ndarray `v`.

TASK 1.1 has 2 subtasks: 1.1.1 and 1.1.2

## 1.2 Sampling from GUE and computing maximum eigenvalue (2 points)

Create a complex valued 2D $n \times n$ ndarray $A$ (`a` in the code) whose entries are iid complex random variables $X + \iota Y$ where $X, Y$ are independent standard normals and $\iota$ is the imaginary unit. Then set $S = (A + A^\star)/2$ (`s` in the code). Note that $A^\star$ denotes the Hermitian (or conjugate) transpose of the matrix $A$. The random matrix $S$ is said to be drawn from the GUE (Gaussian Unitary Ensemble). Once the Hermitian matrix $S$ has been generated, store its maximum (in value, not absolute value) among its $n$ eigenvalues in an entry of the 1D ndarray `v1`.

TASK 1.2 has 2 subtasks: 1.2.1 and 1.2.2.

## 1.3 Normalize GOE eigenvalues (1 point)

The eigenvalues will be on an $O(\sqrt{n})$ scale. Divide `v` by $\sqrt{n/2}$ to get rid of the $n$ dependence in the scale.

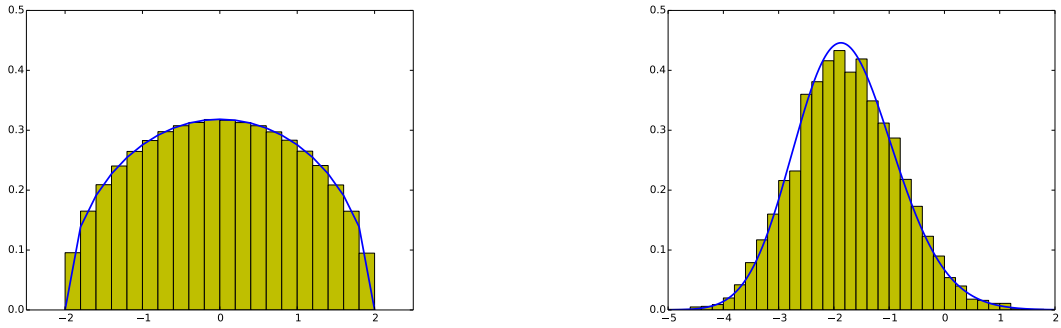

Figure 1: Numerical results versus theoretical predictions made by the Semicircle law for eigenvalue distribution of GOE ensembles (left) and Tracy-Widom law for maximum eigenvalue distribution of GUE ensembles (right)

### 1.4 Plot the histogram of GOE eigenvalues against the theoretical prediction (1 point)

We will bin the eigenvalues in `v` using the function `numpy.histogram()`. We will use the following as bin boundaries:
`-2 -1.8 -1.6 ... 1.8 2`
The call to compute the histogram will result in two variable `hist` and `bin_edges`. You will have `len(bin_edges)` exactly 1 larger than `len(hist)` since the bin counts in `hist` are for eigenvalues that fell in the middle of two successive bin boundary values. Then plot the bin counts using `matplotlib.pyplot.bar()`. The code for plotting the semicircle theoretical prediction and for saving the file to a pdf document is already supplied.

TASK 1.4 has 2 subtasks: 1.4.1 and 1.4.2.

### 1.5 Normalize GUE max eigenvalues (1 point)

The largest eigenvalue will be distributed in a $O(n^{-1/6})$ sized band around its expected value of $O(\sqrt{n})$. So first subtract $2\sqrt{n}$ from `v1` and then multiply each entry by $n^{1/6}$.

### 1.6 Plot the histogram of GUE max eigenvalues against the theoretical prediction (1 point)

This TASK is similar to TASK 1.4 except that we bin `v1` instead of `v` and our bin boundaries will be:
`-5 -4.8 -4.6 ... 1.8 2`
You have to create the bar graph from numerical values in the (normalized) 1D array `v1`. The code to read in the theoretical Tracy-Widom prediction from the file `tracy-widom.csv` and to plot it has already been supplied.

TASK 1.6 has 2 subtasks: 1.6.1 and 1.6.2.

### 1.7 Running the script and discussing the output (2 points)

Once you have supplied all missing pieces of the code, run the script using:
`python assignment_three_rmt.py`

Two pdf files `Semicircle.pdf` and `Tracy-Widom.pdf` will be created by the above script. The figures in them should like the ones shown in Figure 1.

Answer the following questions:

**Q. 1.1** There are several eigenvalue related functions available in `scipy.linalg` including `eig, eigh, eigvals, eigvalsh`. How did you decide which one to choose? Did you look at speed? applicability to this problem? Anything else?

**Q. 1.2** Do you still get the same plots as in Figure 1 if you replace all standard normals you used in the sampling stage with symmetric Bernoulli random variables (i.e. discrete random variables that are either $+1$ to $-1$ with equal probability).

# 2 Creating animation videos for online learning algorithms (10 points)

In this problem, we will create short video animations for two online learning algorithms for binary classification: perceptron and online logistic regression. Both algorithms start with a weight vector $w \in \mathbb{R}^d$ ($d$ will be equal to 2 so that we can plot everything in 2D). They both process the data one example, label pairs at a time.

Given an example $x \in \mathbb{R}^d$ and a label $y \in \{\pm 1\}$, perceptron first tests whether $yw^\top x > 0$. If it is, then no update is made. If not, then it sets $w$ to $w + yx$.

Given an example $x \in \mathbb{R}^d$ and a label $y \in \{\pm 1\}$, online logistic regression performs the update

$$w = w - \eta \ell'(w^\top x, y)x$$

where $\ell'(t, y)$ is the derivative of $\ell(t, y) = \log(1 + \exp(-yt))$ w.r.t. $t$. The step size parameter $\eta$ has been set to its default value of 0.1 in our implementation. The python code for computing the derivative of the logistic loss is already provided in the file:
https://github.com/ambujtewari/stats607a-fall2014/blob/master/homeworks/losses.py
which has already been imported for you in the file for this problem.

Download an (incomplete!) python script by following the following link:
assignment_three_animate.py
Click on the "Raw" button on the top right and save the file as `assignment_three_animate.py`.

## 2.1 Perceptron update (1 point)

Implement the function `perceptron_update`.

## 2.2 Online logistic regression update (1 point)

Implement the function `online_lr_update`.

## 2.3 Create the basic plots (3 points)

Provide code to plot the positives as blue dots, negative as red dots, the separator line (whose equation is $w^\top x = 0$) as a black line.

This TASK consists of 3 subtasks: 2.3.1 through 2.3.3

## 2.4 Fill in the frame update (3 points)

The animation basically works by repeated calling `frame_update` to change the plot (each time with a different argument). We have set up the animation so that `frame_update` will get called with argument drawn from the list `range(2*n+1)`. We won't do anything on the 0th call. On all subsequent calls, we will execute one update of whichever algorithm we're animating. The example picked at iteration `i` will be `(i-1) % n` (along with the corresponding label). Then we will update the separator (black) line and also the current point marked with a yellow star.

This TASK consists of 3 subtasks: 2.4.1 through 2.4.3

### 2.5 Running the script and discussing the output (2 points)

Once you have supplied all missing pieces of the code, run the script using:
`python assignment_three_animate.py`

This will create two `mp4` files in the current directory:
`Online_logistic_regression_anim.mp4` `Perceptron_anim.mp4`
Open them in your favorite video player and answer the following questions:

**Q. 1.1** Perceptron runs on data that is perfectly separable using a (not necessarily unique) linear separator. Does it find one such separator? If not, how many points does the final classifier misclassify? If we keep running the algorithm by cycling through the data, will it eventually classify everything correctly?

**Q. 1.2** Online logistic regression runs on non linearly-separable data. How many points does the final classifier misclassify? If we keep running the algorithm by cycling through the data, will it eventually classify everything correctly?

## 3 Fetching USA jobs data via the web API (10 points)

In this problem, we will fetch data from the USA Jobs API described at:
https://data.usajobs.gov/Rest
Familiarize yourself with how the API works by browsing the above website.

We will store the jobs data for the state of Michigan in a Pandas frame and compute some very simple statistics about jobs from each agency that has jobs available.

Download an (incomplete!) python script by following the following link:
assignment_three_jobs.py
Click on the "Raw" button on the top right and save the file as `assignment_three_jobs.py`.

### 3.1 Implement the function to get data into a Python dict (3 points)

First get the response from the server using the provided URL. Then read the response to get raw JSON text. Convert JSON into a Python dict using the appropriate functions from the `json` module.

This TASK consists of 3 subtasks: 3.1.1 through 3.1.3

### 3.2 Extract job list and number of pages of output (2 points)

You will see that the Python dict obtained upon convert the JSON data into Python dict will have just a few keys. One of them has the actual job data stored as a list in its corresponding value. Another key has the number of "pages" of result (This API has this concept of "pages" probably to ensure that each single call to the web API doesn't take a lot of time). Note that the initial call simply returns the 1st page. There may be additional pages but they have to be fetched explicitly (that's the next task).

This TASK consists of 2 subtasks: 3.2.1 and 3.2.2

### 3.3 Fetch pages of output and keep growing job list (2 points)

We'll run a loop to fetch all remaining pages. Each time we'll simply keep growing out list of jobs.

This TASK consists of 2 subtasks: 3.3.1 and 3.3.2

### 3.4 Create DataFrame and extract unique agencies (2 points)

Convert the list of jobs to a Pandas DataFrame and find out how many unique agencies have their jobs advertised.

This TASK consists of 2 subtasks: 3.4.1 and 3.4.2

### 3.5 Implement the function to convert salary from string to float (1 point)

The salary columns in your DataFrame with have salary stored as a strong, e.g., `$60,000.00`. You'll have to write a function `numeric_value()` that will get rid of the dollar sign and commas and return salary as a float.

### 3.6 Running the script and discussing the output (no points but make sure script runs)

Once you have supplied all missing pieces of the code, run the script using:
`python assignment_three_jobs.py`

The script will produce some text output first about its progress and then about the jobs fetched. The job info is very basic: number of jobs per agency and the average value of the Minimum Salary for those jobs (there's also a Maximum Salary fields for each job but we're ignoring it).