Vladimir Pchelin

# Project: Computation of the density of eigenvalues (DOE) for random matrices

`git@bitbucket.org:ph7vov/sta242.git`

**Overview.** In this project I had two major goals. First of all, I planned to learn new computational techniques. It was interesting to investigate the DOE from the numerical point of view as well. I focused on the ensemble of symmetric random matrices, which entires are iid uniformly distributed variables.The matrices are assumed to be properly normalized so that their largest eigenvalues are of order one. It's known that when the described matrix is large (many rows, many columns) the corresponding DOE $\rho(x)$ should be a semicircle function

$$\rho(x) = \alpha\sqrt{\beta^2 - x^2}, \quad -\beta \le x \le \beta$$

with some constants $\alpha, \beta > 0$. One of my goals was to check this result numerically. To achieve this goal one should, particularly, be able to compute the largest eigenvalue.

**What I did.**

- I used `OpenCL` to estimate maximum eigenvalues with my Intel GPU
- for maximum eigenvalues and densities I applied the `C++` library `Eigen`
- I repeated the same things with `R`
- the performance was compared

**Theoretical insights.** To get the DOE I employed the relation

$$tr\frac{\varepsilon}{(M_n - \lambda)^2 + \varepsilon^2} = \sum_{i=1}^{n} \frac{\varepsilon}{(\lambda_i - \lambda)^2 + \varepsilon^2} \sim \rho(\lambda), \quad \varepsilon \to 0 \tag{1}$$

between a matrix $M_n$ and it's eigenvalues $\lambda_i$. If the DOE is smooth and $\varepsilon > 0$ is small enough only eigenvalues that are close to $\lambda$ contribute to the right hand side of (1). Thus, (1) gives approximately a values of the DOE near $\lambda$ (actually, it's only promotional to the DOE). A similar formula is given by

$$tr\frac{\varepsilon^3}{(M_n - \lambda)^4 + \varepsilon^4} = \sum_{i=1}^{n} \frac{\varepsilon^3}{(\lambda_i - \lambda)^4 + \varepsilon^4} \sim \rho(\lambda), \quad \varepsilon \to 0. \tag{2}$$

In fact, it's not necessary to make $\varepsilon$ very small. It's still theoretically possible to restore the DOE if you know the left side of (1) for some fixed $\varepsilon$ and all $\lambda$.

To find the largest eigenvalue I simply noticed that

$$(trM_n^{2k})^{1/2k} = \left(\sum_{i=1}^{n} \lambda_i^{2k}\right)^{1/2k} \sim \lambda_n, \quad k \to \infty, \tag{3}$$

where $\lambda_n$ is the largest eigenvalue (largest absolute value).

It's expected that computation of polynomials of matrices or inverse matrices is not as time consuming for large matrices as finding eigenvalues explicitly.

**Discussion of the `OpenCL` part, difficulties.** I computed with `OpenCL` the largest eigenvalues on 64 realizations of my ensemble of random symmetric matrices to check how these largest eigenvalues vary.

The major difficulty with `OpenCl` was that I couldn't use other `C++` libraries in my kernel. Even using `double` numbers or multi-dimensional arrays directly is not allowed in `OpenCl`. Moreover, I had to come up with a simple and elementary generator of random numbers for the kernel. Another problem was that my embedded GPU performed poorly. Particularly, the GPU sometimes, probably, didn't have enough memory and forced to reboot the PC. Arithmetic overflow was another big issue.

I decided not to implement embarrassingly parallel matrix inversion on `OpenCL` because this would impose strong restrictions on the size of my matrices and on $\varepsilon$. After all, with my GPU the `OpenCL` code performed not better than `R`.

**`C++` and the `Eigen` library.** I wanted to try some `C++` library I had never used before. That's why I picked `Eigen`. Also, because the developers claim that `Eigen` is faster or on the same level as other popular libraries for linear algebra. Apparently, this library is quite versatile and flexible. I used `Eigen` to invert matrices and to multiply them.

**`R`.** Finally, utilizing `R` was very straightforward. Moreover, `R` appeared to be the fastest.

**Data analysis.** The next graph gives the left hand side of (1), namely

$$sq(\lambda) = tr\frac{\varepsilon}{(M_n - \lambda)^2 + \varepsilon^2} \tag{4}$$

with $\varepsilon = 0.3$.

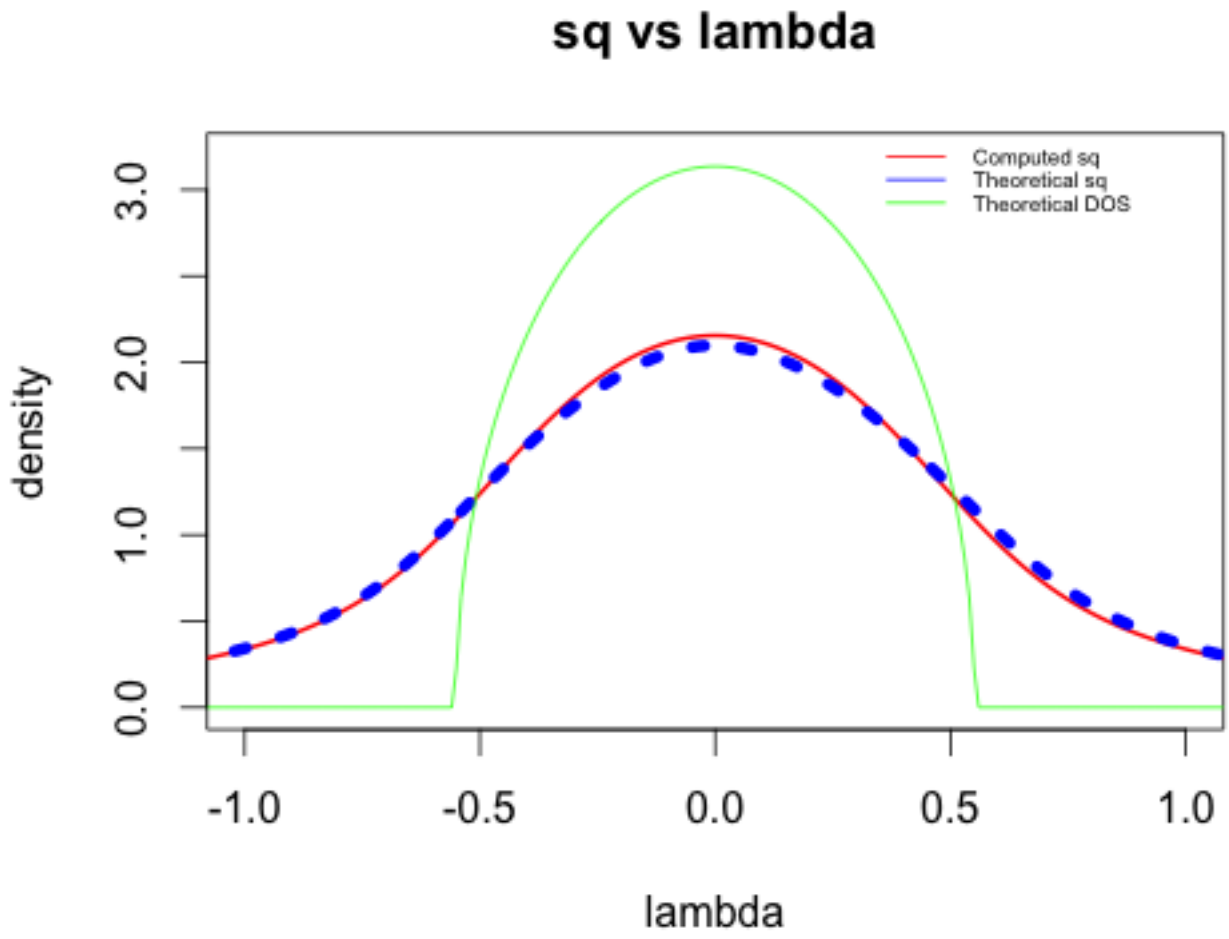


**Figure 1**

The scales on this plot are only relative. The line, that is called "Computed sq", is just a numerically computed $sq(\lambda)$ from (4). The "Theoretical sq" and "Theoretical DOE" are predicted lines based on the data I obtained from matrices, including the computed largest eigenvalue. One can see that two "sq" lines are very close to each other. Thus, we can deduce that the real DOE is very close to the to the "Theoretical DOE". As stated before, based on the knowledge of $sq(\lambda)$ one can restore the DOE.

Now I present the plot of $sq(\lambda)$ and $quad(\lambda)$ with small $\varepsilon = 0.05$. The function $quad(\lambda)$ is defined by

$$quad(\lambda) = tr\frac{\varepsilon^3}{(M_n - \lambda)^4 + \varepsilon^4} \tag{5}$$

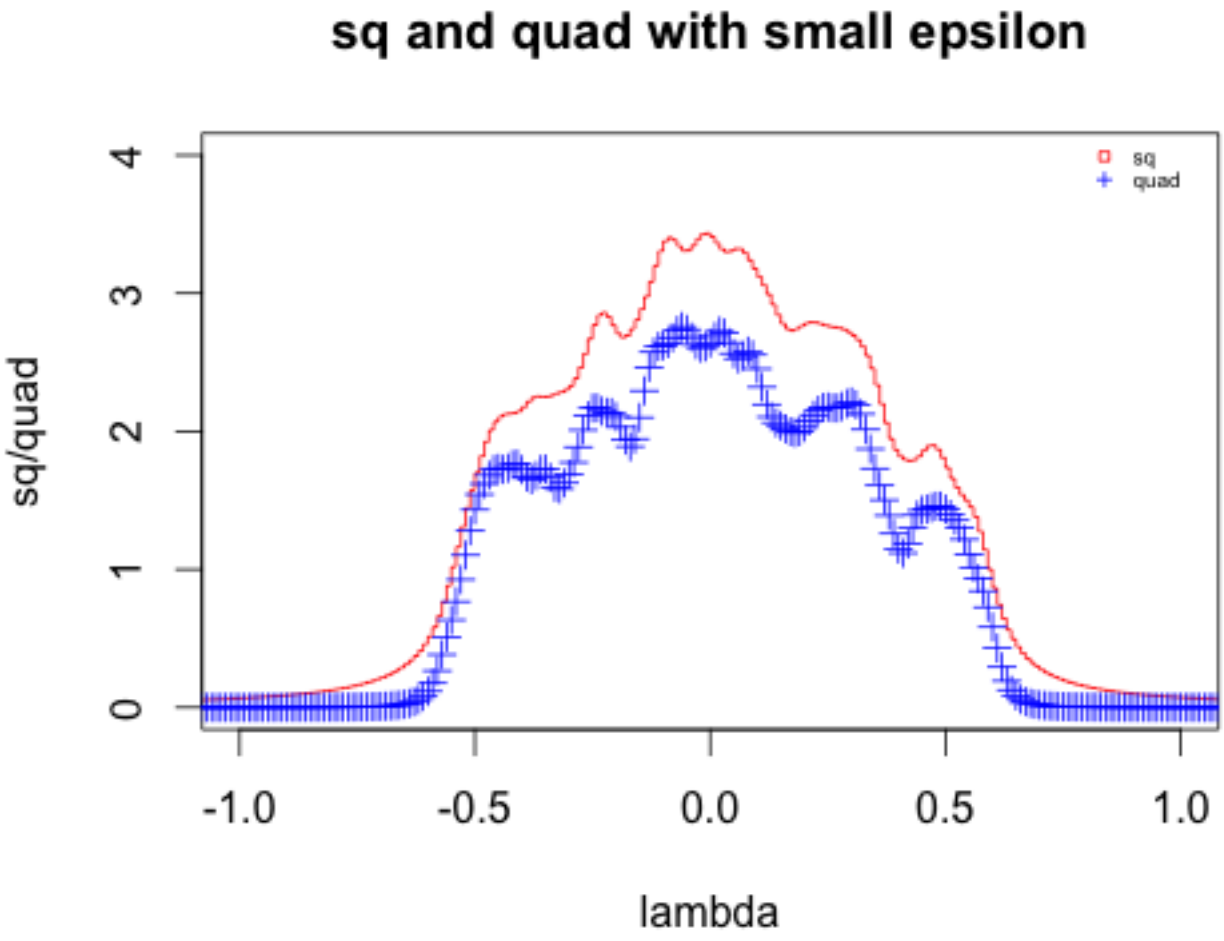


**Figure 2**

Both functions are only approximately propotional to the DOE. Because $\varepsilon$ is small and I used a matrix of order 49 (there are only 49 eigenvalues in total) these functions on the graph are quite irregular. Still, one can get an idea of how the DOE looks like in a prelimit setting.

Using `R` and `Eigen` I got that the largest eigenvalue on the two graphs above should be approximately 0.57. On a contrary, `OpenCL` gave me the upper bound 1.57. The reason is, probably, that the largest eigenvalue is very sensitive to the way we perform randomization. In `OpenCL` I used my own random number generator and this changed the result significantly. It's a very hard task to find the largest eigenvalue precisely since it can fluctuate far away from the support of the DOE. The largest eigenvalue can be found, in principle, even in a region of $\lambda$ where the DOE is zero.

**Performance and comparison of `R` and `Eigen`.** Both `R` and `C++` with `Eigen` gave almost the same results in the sense of data.

First, let's check the performance of routines that compute the estimated DOE. The function, that involves matrix inversion, took 0.22 seconds in `R` and 2.337 in `C++` with `Eigen`. Surprisingly, `R` was faster on my computer.

The times for the function that multiplied matrices to estimate the largest eigenvalues for 64 random samples are given in the table below (in seconds)

| | |
|---|---|
| `OpenCL` | 1.996 |
| `Eigen` | 17.844 |
| `R` | 1.94 |

`OpenCL` and `R` have approximately the same times. But `Eigen` was much slower.

As a conclusion, it's much easier and convenient to use `R` for matrix manipulations. Probably, some `R` operations are optimized so well that you simply can't beat them. Also, I tried to use `Lapack` first. But `Eigen`, in my opinion, is a more user friendly library, so I preferred `Eigen`.

**To prepare the project and learn new techniques I used the following sources.**


- http://www.thebigblob.com/getting-started-with-opencl-and-gpu-computing/
- https://vasanthexperiments.wordpress.com/
- https://developer.apple.com/opencl/
- http://eigen.tuxfamily.org/
- http://www.netlib.org/lapack/
- Matloff N. - Art of R Programming - A Tour of Statistical Software Design


Plus countless number of topics on `stackoverflow`.