Determinant maximisation algorithm for the configuration model for correlation matrices

Sadamori Kojaku and Naoki Masuda

Department of Engineering Mathematics,

Merchant Venturers Building, University of Bristol,

Woodland Road, Clifton, Bristol BS8 1UB, United Kingdom

I. DETERMINANT MAXIMISATION ALGORITHM

Consider an $N \times N$ covariance matrix, denoted by Σ, of the following form:

$$\Sigma = \frac{1}{L} X X^{\top}, \tag{1}$$

where $X = (x_{ij})$ is an $N \times L$ real matrix and $\top$ represents the transposition. One can interpret the ith row of X, i.e., $[x_{i1}, x_{i2}, \ldots, x_{iL}]$, as the ith time series of length L in discrete time. Alternatively, x_{ij} may be the ith feature observed for the jth item. Given an input covariance matrix $\Sigma^{\rm org}$, we aim to find the probability distribution of X, denoted by $P(X)$, which defines a probability distribution of covariance matrix $\Sigma^{\rm con}$ through Eq. (1), such that the entropy

$$-\int P(X) \ln P(X) \mathrm{d}X \tag{2}$$

is maximised under constraints. We impose that the generated covariance matrix $\Sigma^{\rm con}$ preserves the expectation of the diagonal elements of $\Sigma^{\rm org}$, i.e.,

$$\int \Sigma^{\rm con}_{ii} P(X) \mathrm{d}X = \Sigma^{\rm org}_{ii} \quad (1 \leq i \leq N), \tag{3}$$

and the expectation of the row sum (equivalently, the column sum) of the off-diagonal elements of $\Sigma^{\rm org}$, i.e.,

$$\sum_{j=1;j\neq i}^{N} \int \Sigma^{\rm con}_{ij} P(X) \mathrm{d}X = \sum_{j=1;j\neq i}^{N} \Sigma^{\rm org}_{ij} \quad (1 \leq i \leq N). \tag{4}$$

In Ref. [1], we showed that the maximum-entropy distribution of X takes form

$$P(X) = \prod_{l=1}^{L} \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} \exp\left[-\frac{1}{2} \boldsymbol{x}_\ell^{\top} \Sigma^{-1} \boldsymbol{x}_\ell\right], \tag{5}$$

where $\boldsymbol{x}_\ell$ is the ℓth column of matrix X, i.e., $\boldsymbol{x}_\ell = [x_{1\ell}, x_{2\ell}, \ldots, x_{N\ell}]^\top$. Substituting Eq. (6) into Eq. (2) yields

$$\begin{aligned}
-\int P(X)\ln P(X)\mathrm{d}X &= \sum_{\ell=1}^{L}\left[\int\left(\frac{1}{2}\boldsymbol{x}_\ell^T\Sigma^{-1}\boldsymbol{x}_\ell\right)P(X)\mathrm{d}X + \int \ln\left(\sqrt{(2\pi)^N|\Sigma|}\right)P(X)\mathrm{d}X\right] \\
&= \sum_{\ell=1}^{L}\left[\int \mathrm{Tr}\left(\frac{1}{2}\Sigma^{-1}\boldsymbol{x}_\ell^T\boldsymbol{x}_\ell\right)P(X)\mathrm{d}X + \ln\left(\sqrt{(2\pi)^N|\Sigma|}\right)\int P(X)\mathrm{d}X\right] \\
&= \sum_{\ell=1}^{L}\left[\mathrm{Tr}\left(\frac{1}{2}\Sigma^{-1}\Sigma\right) + \frac{N}{2}\ln(2\pi) + \frac{1}{2}\ln|\Sigma|\right] \\
&= \frac{NL}{2} + \frac{NL}{2}\ln(2\pi) + \frac{L}{2}\ln|\Sigma|. \qquad (6)
\end{aligned}$$

The first and the second terms on the right-hand side of Eq. (6) are constant. Therefore, one can transform the maximisation of Eq. (2) subject to Eqs. (3) and (4) into the following determinant maximisation problem:

$$\max_{\Sigma} \ln|\Sigma|, \qquad (7)$$

subject to constraints

$$\Sigma_{ii} = \Sigma_{ii}^{\mathrm{org}} \quad (1 \le i \le N), \qquad (8)$$

$$\sum_{j=1;j\neq i}^{N}\Sigma_{ij} = \sum_{j=1;j\neq i}^{N}\Sigma_{ij}^{\mathrm{org}} \quad (1 \le i \le N), \qquad (9)$$

where we have used the relationship $\int \Sigma_{ij}P(X)\mathrm{d}X = \Sigma_{ij}$ to rewrite the constraints. In Ref. [1], we found that Σ has a specific form as parameterised in Eq. (7) in Ref. [1] and numerically solved for the parameters α_i and β_i $(1 \le i \le N)$ by a steepest descent algorithm (Eqs. (11) and (12) in Ref. [1]). In contrast, here we do not use the parameterised Σ.

The objective function, $\ln|\Sigma|$, is a concave function with respect to Σ [2]. The feasible region (i.e., set of Σ satisfying the constraints) is a convex set because it is the intersection of the set of positive semidefinite matrices and hyperplanes. Therefore, this determinant maximisation problem is a convex optimisation problem, in which one can efficiently find the global optimum using a numerical solver. We refer to the algorithm that solves the maximisation problem as the DMCC algorithm (DMCC stands for the Determinant Maximisation algorithm for the Configuration model for Correlation marices).

II. COMPUTATION TIME

We compare the CPU time of the original algorithm [1] and the DMCC algorithm. We implemented the original algorithm in MATLAB and Python. We implemented the

TABLE I: Average CPU time of the different algorithms on the five empirical correlation matrices.

Data	N	Average CPU Time (s)		
		Original (MATLAB)	Original (Python)	DMCC (Python)
Motivation	30	11	33	13
fMRI1	264	29,391	14,889	858
fMRI2	264	36,990	18,482	800
Japan	264	26,183	13,368	1,644
US	325	56,359	27,332	2,919

DMCC algorithm in Python, in which we solved the determinant maximisation problem using a splitting conic solver [3]. All the codes are on GitHub [4]. We applied the three different algorithms to the empirical correlation matrices used in Ref. [1]. We used Intel Xeon E5-2680 v4 processor with 28 logical cores and 4GB memory. The implemented algorithms use some cores in parallel. We compute the CPU time of each algorithm as the sum of the CPU time over all cores. The different algorithms and runs found the same optimal solution up to the numerical error. For each data, we run each algorithm 30 times. Then, we averaged the CPU time over 30 runs.

Table I shows the average CPU time of the different algorithms. For all data except Motivation, the Python implementation of the original algorithm took less CPU time than its MATLAB counterpart. The DMCC algorithm required by far the least CPU time for all data except Motivation. It consumed approximately between 4% and 40% of the CPU time of the Python implementation of the original algorithm. In addition, unlike the original algorithm, the DMCC algorithm does not require a learning rate parameter (ϵ in Eqs. (11) and (12) in Ref. [1]) that needs some tuning.

[1] N. Masuda, S. Kojaku, and Y. Sano, Phys. Rev. E **98**, 012312 (2018).

[2] L. Vandenberghe, S. Boyd, and S. Wu, SIAM J. Mat. Anal. Appl. **19**, 499 (1998).

[3] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, J. Opt. Theo. Appl. **169**, 1042 (2016).

[4] MATLAB and Python codes. Available at https://github.com/naokimas/config_corr/.