Applied Mathematics and Computation 363 (2019) 124628

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Smallest eigenvalue of large Hankel matrices at critical point: Comparing conjecture with parallelised computation

Yang Chen[a], Jakub Sikorowski[a], Mengkun Zhu[b,*]

[a] Department of Mathematics, University of Macau, Avenida da Universidade, Taipa, Macau

[b] School of Mathematics and Statistics, Qilu University of Technology (Shandong Academy of Sciences), Jinan 250353, China

ARTICLE INFO

MSC:
15B57
34E05
42C05
65F15
65Y05

Keywords:
Random matrix
Parallel eigensolver
Extremely ill-conditioned Hankel matrices
Smallest eigenvalue

ABSTRACT

We propose a novel parallel numerical algorithm for calculating the smallest eigenvalues of highly ill-conditioned Hankel matrices. It is based on the *LDLT* decomposition and involves finding a $k \times k$ sub-matrix of the inverse of the original $N \times N$ Hankel matrix H_N^{-1}. The computation involves extremely high precision arithmetic, message passing interface, and shared memory parallelisation. We demonstrate that this approach achieves good scalability on a high performance computing cluster (HPCC) which constitutes a major improvement of the earlier approaches. We use this method to study a family of Hankel matrices generated by the weight $w(x) = \mathrm{e}^{-x^{\beta}}$, supported on $[0, \infty)$ and $\beta > 0$. Such weight generates a Hankel determinant, a fundamental object in random matrix theory. In the situation where $\beta > 1/2$, the smallest eigenvalue tends to 0 exponentially fast. If $\beta < 1/2$, which is the situation where the classical moment problem is indeterminate, then the smallest eigenvalue is bounded from below by a positive number. If $\beta = 1/2$, it is conjectured that the smallest eigenvalue tends to 0 algebraically, with a precise exponent. The algorithm run on the HPCC producing a fantastic match between the theoretical value of $2/\pi$ and the numerical result.

1. Background and motivation

Hankel matrices have entries that are moments [1,12] of probabilities measures or weight functions. They play an important role in the theory of Hermitian random matrices [14]. The study of the largest and smallest eigenvalues are important since they provide useful information about the nature of the Hankel matrix generated by a given density, e.g. they are related with the inversion of Hankel matrices, where the condition numbers are enormously large.

Given $\{\mu_j\}$, the moment sequence of a weight function $w(x) (> 0)$ with infinite support s,

$$\mu_j := \int_s x^j w(x) dx, \quad k = 0, 1, 2, \ldots, \tag{1.1}$$

it is known that the Hankel matrices

$$H_N := (\mu_{i+j})_{i,j=0}^{N-1}, \quad N = 1, 2, \ldots$$

are positive definite [10].

[*] Corresponding author.

E-mail addresses: yangbrookchen@yahoo.co.uk, yayangchen@umac.mo (Y. Chen), sikorowski@gmail.com (J. Sikorowski), Zhu_mengkun@163.com (M. Zhu).

https://doi.org/10.1016/j.amc.2019.124628
0096-3003/© 2019 Elsevier Inc. All rights reserved.

Let λ_1 be the smallest eigenvalue of H_N. The asymptotic behavior of λ_1 for large N has a broad interest, see e.g. [3,4,6,7,9,15–19]. The authors in [2,13] have studied the behavior of the condition number $\mathrm{cond}(H_N) := \frac{\lambda_N}{\lambda_1}$, where λ_N denotes the largest eigenvalue of H_N.

Szegö [15] investigated the asymptotic behavior of λ_1 for the Hermite weight $w(x) = \mathrm{e}^{-x^2}, x \in \mathbb{R}$ and the classical Laguerre weight $w(x) = \mathrm{e}^{-x}, x \geq 0$. He found[1]

$$\lambda_1 \simeq A N^{\frac{1}{4}} B^{\sqrt{N}},$$

where A, B are certain constants, satisfying $0 < A, 0 < B < 1$. Moreover, in the same paper, it was shown that the largest eigenvalue λ_N of the Hankel matrices $[\frac{1}{i+j+1}]_{i,j=0}^{N-1}$, $[\Gamma(\frac{i+j+1}{2})]_{i,j=0}^{N-1}$ and $[\Gamma(i+j+1)]_{i,j=0}^{N-1}$ were approximated by $\frac{\pi}{2}$, $\Gamma(N+\frac{1}{2})$ and $(2N)!$ respectively. Widom and Wilf [17] studied the situation in which the density $w(x)$ is supported on a compact interval $[a, b]$, such that the Szegö condition

$$\int_a^b \frac{-\ln w(x)}{\sqrt{(b-x)(x-a)}} dx < \infty,$$

holds. They found that

$$\lambda_1 \simeq A\sqrt{N} B^N.$$

Chen and Lawrence [6] found the asymptotic behavior of λ_1 with the weight function $w(x) = \mathrm{e}^{-x^\beta},\ x \in [0, \infty),\ \beta > \frac{1}{2}$. This work has been generalized to the weight function $w(x) = x^\alpha \mathrm{e}^{-x^\beta},\ x \in [0, \infty),\ \alpha > -1,\ \beta > \frac{1}{2}$, see Zhu *et al* [20]. In both cases, the theoretical challenge to finding the asymptotic behavior is at the critical point $\beta = \frac{1}{2}$. It marks the transition point at which the moment problem becomes indeterminate. Berg et al. [3] proved that the moment sequence (1.1) is indeterminate if and only if $\lambda_1(N)$ is bounded away from 0 for all N including ∞.

This is a new criterion for the determination of the Hamburger moment problem. Chen and Lubinsky [7] found the behavior of $\lambda_1(N)$ when $w(x) = \mathrm{e}^{-|x|^\alpha},\ x \in \mathbb{R},\ \alpha > 1$. Berg and Szwarc [4] proved that $\lambda_1(N)$ has exponential decay to zero for any measure which has compact support. Recently, Zhu et al. [19] studied the Jacobi weight, $w(x) = x^\alpha(1-x)^\beta,\ x \in [0, 1],\ \alpha > -1,\ \beta > -1$ and derived an approximation formula of $\lambda_1(N)$,

$$\lambda_1(N) \simeq 2^{\frac{15}{4}} \pi^{\frac{3}{2}} \left(1 + 2^{\frac{1}{2}}\right)^{-2\alpha} \left(1 + 2^{-\frac{1}{2}}\right)^{-2\beta} N^{\frac{1}{2}} \left(1 + 2^{\frac{1}{2}}\right)^{-4(N+1)},$$

which reduces to Szegö's result [15] if $\alpha = \beta = 0$.

This paper is concerned with a numerical computation that is motivated by random matrix theory. We are interested in finding the numerical value of the smallest eigenvalue of a class of $\mathbb{R}^{N\times N}$ Hankel matrices.

2. Mathematical statement of the problem

We consider the weight function

$$w(x) := \mathrm{e}^{-x^\beta}, \qquad x \in [0, \infty), \quad \beta > 0, \tag{2.1}$$

and the moments, based on (1.1), are given by

$$\mu_k = \frac{1}{\beta}\Gamma\left(\frac{k+1}{\beta}\right), \quad k = 0, 1, 2, \ldots. \tag{2.2}$$

In this case, the entries of the Hankel matrix

$$H_N = \begin{bmatrix} \mu_0 & \mu_1 & \mu_2 & \cdots & \cdots & \mu_{N-1} \\ \mu_1 & \mu_2 & & & & \vdots \\ \mu_2 & & \ddots & & & \vdots \\ \vdots & & & \ddots & & \mu_{2N-4} \\ \vdots & & & & \mu_{2N-4} & \mu_{2N-3} \\ \mu_{N-1} & \cdots & \cdots & \mu_{2N-4} & \mu_{2N-3} & \mu_{2N-2} \end{bmatrix} \tag{2.3}$$

increase factorially along the leading diagonal.

Remark 2.1. It is clear that μ_k given above is a rapidly increasing function of k, as long as $\beta > 0$. We shall see later that $\beta = 1/2$ is of great interest since, at this value of β, the smallest eigenvalue $\lambda_1(N)$ will tend to zero slowly.

In this paper, we focus on the study the smallest eigenvalue at the critical point $\beta = \frac{1}{2}$.

[1] In all of this paper, $a_N \simeq b_N$ means $\lim_{N\to\infty} a_N/b_N = 1$.

2.1. Condition number

Let $\{\lambda_i\}_{i=1}^N$ be the N positive eigenvalues of H_N. In order to highlight the numerical challenge, we would like to estimate the condition number of the problem

$$\mathrm{cond}(H_N) = \lambda_N/\lambda_1,$$

where λ_N is the largest eigenvalue and λ_1 is the smallest. We observe that the largest eigenvalue λ_N is bounded by the trace of the matrix, so

$$\lambda_N \leq \sum_{i=1}^{N} \lambda_i = \mathrm{tr}(H_N) = \sum_{k=0}^{N-1} \frac{1}{\beta}\Gamma\left(\frac{2k+1}{\beta}\right).$$

On the other hand it must be larger or equal than the largest entry along the diagonal, i.e.

$$\lambda_N \geq \frac{1}{\beta}\Gamma\left(\frac{2N-1}{\beta}\right).$$

For sufficiently large N, we note

$$\sum_{k=0}^{N-1} \frac{1}{\beta}\Gamma\left(\frac{2k+1}{\beta}\right) \simeq \frac{1}{\beta}\Gamma\left(\frac{2N-1}{\beta}\right),$$

therefore,

$$\lambda_N \simeq \frac{1}{\beta}\Gamma\left(\frac{2N-1}{\beta}\right).$$

For the case of $\beta = \frac{1}{2}$, the smallest eigenvalues are presented in Table 2. They are of order $\frac{1}{10}$ and slowly decreasing with N. Therefore, we estimate the condition number (assuming $\lambda_1 = \frac{1}{10}$) as

$$\mathrm{cond}(H_N) \simeq 20\ \Gamma(4N-2).$$

Notice that the condition number grows factorially with increasing N; as a result, we say that these matrices are extremely ill-conditioned. This property makes the numerical computation particularly challenging.

The condition number of a problem quantifies the sensitivity of the output value to small changes in the input. For example, in our case, a finite precision of numerical representation of the Hankel matrix introduces a (hopefully small) change of the computed eigenvalue. We want this change to be less than 10^{-15}. However, an astronomically large condition number makes it hard, and the initial Hankel matrix must be initialized with extreme amount of precision, for example $N = 4500$ needs $\sim$70,000 digits of precision. Subsequently the intermediate arithmetic operations must be performed with even higher precision in order for the rounding error not to introduce large errors in the output. These extreme precision requirements demand significant computing resources, and present novel challenges and trade offs.

We would like to stress that the algorithm we use in this paper was chosen to be as numerically stable as possible (see comparison between Secant, Householder, Jacobi, Lanczos in [9] and Section 5.2 on page 20), and the numerical challenges are intrinsic to the problem and are not due to instabilities of the algorithms.

3. Properties of Hankel matrices

The Hankel matrices we defined in Section 2 have many interesting properties; here we would like to describe those properties that are used in this paper. We notice that the matrix in Eq. (2.3) is explicitly symmetric. If $\pi_N(x)$ is a polynomial of degree N with real coefficients c_j, $j = 0, 1, \ldots N$, namely,

$$\pi_N(x) := \sum_{j=0}^{N} c_j x^j,$$

then the quadratic form

$$\sum_{i,\,j=0}^{N} c_i \mu_{i+j} c_j = \int_0^\infty [\pi_N(x)]^2 w(x) dx,$$

is positive definite.

The authors in [6] have studied the small eigenvalues with respect to the weight function (2.1) and derived the asymptotic expression for smallest eigenvalue for $\beta > \frac{1}{2}$. They have found that the smallest eigenvalue decreases exponentially with increasing N. On the other hand, the situation for $0 < \beta < \frac{1}{2}$, the weight (2.1) is Stieltjes indeterminate: that is to say, there are infinitely many measures supported on $[0, \infty)$ with the moments (2.2). The authors in [6] found that the smallest eigenvalue is bounded from below by a positive number for any N.

At the critical point $\beta = \frac{1}{2}$, the asymptotic behavior changes abruptly, and we have a phase transition. Furthermore, the authors of [6] argued that under certain assumptions the asymptotic expression for the smallest eigenvalue is

$$\lambda_1(N) \simeq 8\pi \frac{\sqrt{\log(4\pi N\mathrm{e})}}{(4\pi N\mathrm{e})^{\frac{2}{\pi}}}, \qquad \text{for } \beta = \frac{1}{2}, \tag{3.1}$$

for large N. Some of those assumptions could not be proven, therefore, we treat Eq. (3.1) as a conjecture. In the following Sections 3.1 and 3.2 we summarise the main steps of [6] performed to find Eq. (3.1).

Remark 3.1. We display here, for $\beta = \frac{7}{4}$, the inverse of the smallest eigenvalue for large N, [6]

$$\frac{1}{\lambda_1} \simeq \frac{1}{8\pi^{\frac{5}{4}}} \frac{c^{\frac{1}{4}}}{\sqrt{A_0}} \mathrm{e}^{\sec\left(\frac{7\pi}{4}\right)} N^{-\frac{5}{7}} \exp\left[\frac{2N^{\frac{5}{7}}}{\sqrt{\pi c}}\left(A_0 - \frac{A_1}{c}\frac{1}{N^{\frac{4}{7}}}\right)\right],$$

where

$$c = 4\left[\frac{\left(\Gamma\left(\frac{7}{4}\right)\right)^2}{\Gamma\left(\frac{7}{2}\right)}\right]^{\frac{4}{7}}, \quad A_0 = \frac{14\sqrt{\pi}}{5}, \quad A_1 = \frac{7\sqrt{\pi}}{3}.$$

3.1. *Polynomials* $\mathrm{P_n}$

The main mathematical objects in the derivation of Eq. (3.1) are $P_j(x)$, the orthonormal polynomials associated with the weight $\exp\left(-x^{\beta}\right),\ x \geq 0,\ \ \beta > 0,$ such that

$$\int_0^{\infty} P_i(x)P_j(x)\mathrm{e}^{-x^{\beta}}dx = \delta_{ij},$$

where δ_{ij} denotes the Kronecker's delta, i.e., $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ for $i \neq j$.

It is shown [6] that the smallest eigenvalue $\lambda_1(N)$ is bounded from below

$$\lambda_1(N) \geq \frac{2\pi}{\sum_{j=0}^{N} K_{jj}}, \tag{3.2}$$

where

$$K_{jk} := \int_{-\pi}^{\pi} P_j(-\mathrm{e}^{\mathrm{i}\phi})P_k(-\mathrm{e}^{-\mathrm{i}\phi})d\phi.$$

Using the Christoffel–Darboux formula [10] and the result presented in [5] for large N off-diagonal recurrence coefficients, it was found that

$$\sum_{j=0}^{N} K_{jj} \simeq \pi^2 N^2 \int_{-\pi}^{\pi} \frac{P_N(-\mathrm{e}^{\mathrm{i}\phi})P_{N+1}(-\mathrm{e}^{-\mathrm{i}\phi}) - P_N(-\mathrm{e}^{-\mathrm{i}\phi})P_{N+1}(-\mathrm{e}^{\mathrm{i}\phi})}{\mathrm{e}^{\mathrm{i}\phi} - \mathrm{e}^{-\mathrm{i}\phi}} d\phi \tag{3.3}$$

Finally, the authors of [6] used the asymptotic expression for $P_N(x)$ for large N

$$P_N(x) \simeq \frac{(-1)^N}{2\pi}(-x)^{-\frac{1}{4}} N^{-\frac{1}{2}} \exp\left[\frac{\sqrt{-x}}{\pi}\log\left(\frac{4\pi N\mathrm{e}}{\sqrt{-x}}\right)\right], \tag{3.4}$$

to evaluate Eq. (3.3) for large N. We perform the integration in the following section.

3.2. *Saddle point approximation*

In this section we evaluate the integral in Eq. (3.3) using the Laplace method, see [8]. We start by substituting the asymptotic expressions Eq. (3.4) into Eq. (3.3). This leads to an integral with an integrand that has a maximum at $\phi = 0$ and falls off to 0 towards $\phi = \pm\pi$

$$\begin{aligned}\sum_{j=0}^{N} K_{jj} \simeq -\int_{-\pi}^{\pi} d\phi \frac{N}{4\left(\mathrm{e}^{\mathrm{i}\phi} - \mathrm{e}^{-\mathrm{i}\phi}\right)}\Bigg\{ &\exp\left[\frac{\mathrm{e}^{\frac{\mathrm{i}\phi}{2}}}{\pi}\log\left(4\pi N\mathrm{e}^{1-\frac{\mathrm{i}\phi}{2}}\right) + \frac{\mathrm{e}^{-\frac{\mathrm{i}\phi}{2}}}{\pi}\log\left(4\pi(N+1)\mathrm{e}^{1+\frac{\mathrm{i}\phi}{2}}\right)\right] \\ &- \exp\left[\frac{\mathrm{e}^{-\frac{\mathrm{i}\phi}{2}}}{\pi}\log\left(4\pi N\mathrm{e}^{1+\frac{\mathrm{i}\phi}{2}}\right) + \frac{\mathrm{e}^{\frac{\mathrm{i}\phi}{2}}}{\pi}\log\left(4\pi(N+1)\mathrm{e}^{1-\frac{\mathrm{i}\phi}{2}}\right)\right]\Bigg\}\end{aligned} \tag{3.5}$$

Furthermore, the width of the peak decreases as N increases. As a consequence, we can use the Laplace method to evaluate the integral for large N. We recast the integrand as $\mathrm{e}^{-f(\phi,N)}$ and expand $f(\phi, N)$ in a Taylor series

$$\sum_{j=0}^{N} K_{jj} \simeq \int_{-\pi}^{\pi} \mathrm{e}^{-f(\phi,N)}d\phi \simeq \int_{-\infty}^{\infty} \mathrm{e}^{-f_0(N)-f_1(N)\,\phi-\frac{1}{2}\,f_2(N)\,\phi^2-\frac{1}{6}\,f_3(N)\,\phi^3-\frac{1}{24}\,f_4(N)\,\phi^4+\cdots}d\phi \tag{3.6}$$

Expanding around $\phi = 0$, we find that $f_1(N)$ and $f_3(N)$ vanish. Moreover,

$$\frac{f_4(N)}{f_2^2(N)} = \mathcal{O}\left(\frac{1}{\log(N)}\right) \ll 1$$

for large N, therefore, we can treat $f_4(N)$ and higher order terms as small perturbation[2] and expand the exponential

$$\sum_{j=0}^{N} K_{jj} \simeq \mathrm{e}^{-f_0(N)} \sqrt{\frac{2\pi}{f_2(N)}} \left(1 - \frac{f_4(N)}{8\, f_2^2(N)} + \ldots\right), \tag{3.7}$$

where

$$\begin{aligned}
\mathrm{e}^{-f_0(N)} &= 2^{-3+\frac{4}{\pi}}\, \mathrm{e}^{\frac{2}{\pi}}\, \pi^{\frac{2}{\pi}-1}\, N^{\frac{3}{2}+\frac{1}{\pi}}\, (N+1)^{-\frac{1}{2}+\frac{1}{\pi}} \log\left(1+1/N\right), \\
12\pi^2 f_2(N) &= -3\pi\,(2+\pi - 2\log 4\pi) + \log^2 N + 3\pi \log(N+1) \\
&\quad + [3\pi - 2\log(N+1)]\log N + \log^2(N+1), \\
240\,\pi^4 f_4(N) &= -30\pi^3(-3+\pi + \log 4\pi) + 2\log^4 N - 15\pi^3 \log(N+1) \\
&\quad - 8\log^3 N \log(N+1) - 20\pi^2 \log^2(N+1) + 2\log^2(N+1) \\
&\quad - 4\log^2 N\big[5\pi^2 - 3\log^2(N+1)\big] \\
&\quad + \big[-15\pi^3 + 40\pi^2 \log(N+1) - 8\log^3(N+1)\big]\log N,
\end{aligned}$$

Expanding $f_1(N), \ldots, f_4(N)$ around $N \to \infty$, we obtain

$$\begin{aligned}
\mathrm{e}^{f_0(N)} &= 2^{\frac{4}{\pi}-3}\mathrm{e}^{\frac{2}{\pi}}\pi^{\frac{2}{\pi}-1}N^{\frac{2}{\pi}}\left[1 + \frac{\pi - 1}{\pi}\frac{1}{N} + \mathcal{O}\left(\frac{1}{N^2}\right)\right], \\
f_2(N) &= \frac{1}{2\pi}\left[\log\left(4\pi\,\mathrm{e}^{-1-\frac{\pi}{2}}N\right) + \frac{1}{4\pi N} + \mathcal{O}\left(\frac{1}{N^2}\right)\right], \\
f_4(N) &= -\frac{1}{8\pi}\left[\log(4\pi\,\mathrm{e}^{\pi-3}N) + \mathcal{O}\left(\frac{1}{N}\right)\right].
\end{aligned} \tag{3.8}$$

Taking only the first term in the above equations and neglecting all corrections to the saddle point approximation in Eq. (3.7), we obtain

$$\sum_{j=0}^{N} K_{jj} \simeq \frac{(4\pi N\mathrm{e})^{\frac{2}{\pi}}}{4\sqrt{\log\left(4\pi\,\mathrm{e}^{-1-\frac{\pi}{2}}N\right)}},$$

which together with Eq. (3.2) gives[3]

$$\lambda_1(N) \simeq \frac{8\pi\sqrt{\log\left(4\pi\,\mathrm{e}^{-1-\frac{\pi}{2}}N\right)}}{(4\pi\,\mathrm{e})^{\frac{2}{\pi}}N^{\frac{2}{\pi}}}. \tag{3.9}$$

Subsequently including the next-to-leading order correction in the saddle point, i.e. including the $\frac{f_4(N)}{8\, f_2^2(N)}$ term, see Eq. (3.7), we obtain

$$\sum_{j=0}^{N} K_{jj} \simeq \frac{(4\pi N\mathrm{e})^{\frac{2}{\pi}}}{4\sqrt{\log\left(4\pi\,\mathrm{e}^{-1-\frac{\pi}{2}}N\right)}}\left(1 + \frac{\pi}{16\log N}\right), \tag{3.10}$$

$$\lambda_1(N) \simeq \frac{8\pi\sqrt{\log\left(4\pi\,\mathrm{e}^{-1-\frac{\pi}{2}}N\right)}}{(4\pi\,\mathrm{e})^{\frac{2}{\pi}}N^{\frac{2}{\pi}}}\left(1 - \frac{\pi}{16\log N}\right) \tag{3.11}$$

$$\simeq \frac{8\pi\sqrt{\log\left(4\pi\,\mathrm{e}^{-1-\frac{\pi}{2}-\frac{\pi}{8}}N\right)}}{(4\pi\,\mathrm{e})^{\frac{2}{\pi}}N^{\frac{2}{\pi}}} \tag{3.12}$$

[2] Similarly $\frac{f_{2n}(N)}{f_2^n(N)} = \mathcal{O}(\frac{1}{\log^n(N)})$ and $f_{2n+1}(N) = 0$ for $n = 1, 2, 3\ldots$

[3] Authors of this publication have noticed that Eq. (3.9) does not exactly match the Eq. (3.1) which is quoted from [6], we suspect there was a typo in the earlier publication.

$$\simeq \frac{8\pi\sqrt{\log N}}{(4\pi Ne)^{\frac{2}{\pi}}}\left[1+\frac{8\log\left(4\pi\,e^{-1-\frac{\pi}{2}}\right)-\pi}{16\log N}\right]. \tag{3.13}$$

The transformations from Eqs. (3.10) to (3.13) neglect all next-to-next-to-leading order terms. Moreover, we need to stress that this expression might not include all of the next-to-leading order contributions. For example, it does not include next-to-leading order contributions (potentially) coming from

- The approximation in Eq. (3.3),
- The asymptotic nature of Eq. (3.4).

However, we notice that the subleading term that we calculated corrects only the log term in Eq. (3.12) and does not affect the overall $N^{-\frac{2}{\pi}}$ factor, with exponent $-\frac{2}{\pi}$, which we refer to as the leading exponent. In Section 6 we show that directly computing the determinant of Hankel matrices we were able to find $-\frac{2}{\pi}$ to a good precision.

Keeping in mind that Eq. (3.10) might not capture the full next-to-leading order contribution, it would be prudent to use only the leading order approximation

$$\sum_{j=0}^{N} K_{jj} \simeq \frac{(4\pi Ne)^{\frac{2}{\pi}}}{4\sqrt{\log N}}, \tag{3.14}$$

$$\lambda_1(N) \simeq \frac{8\pi\sqrt{\log N}}{(4\pi Ne)^{\frac{2}{\pi}}}. \tag{3.15}$$

Observe that the asymptotic expression in Eq. (3.1) does not decrease exponentially with N, as opposed to the $\beta > \frac{1}{2}$ case [6]. Moreover, the sub-leading terms in Eq. (3.13) are suppressed only by $\log N$ terms and decrease very slowly with increasing N. This makes the difference between Eq. (3.14) and the numerically computed value also decrease very slowly with N; this was indeed observed in [9]. The differences were decreasing with N, however, much slower than for $\beta \neq 1/2$, and, as a result, the numerics did not convincingly confirm Eq. (3.1). In this paper, we endeavour to improve upon that.

4. Numerical algorithm

In this section, we describe the novel numerical algorithm for computing the smallest eigenvalues of a highly-ill conditioned matrix. We tweak and optimise the algorithm for Hankel matrix in Eq. (2.3).

Large parts of the new algorithm are based on the Secant algorithm described in [9]. However, for completeness, we will explain the new algorithm without assuming any reader's knowledge of [9].

4.1. Precision

The posed problem is a highly ill-condition, therefore, in order to obtain accurate results we have to perform arithmetic operations and store numbers with extremely high precision. In order to obtain appropriate precision, we have used the arbitrary precision integer arithmetic implemented in GNU Multiple Precision Arithmetic Library. Each element of the initial matrix H is represented by

$$\frac{\mathbb{Z}_{GMP}}{2^K} \tag{4.1}$$

where K represents the number of bits of precision of the calculation, and $\mathbb{Z}_{GMP}$ is the GMP arbitrary precision integer. This way we obtain a representation of a number with a fixed number of bits to the right of the decimal point and an arbitrary number of bits to the left of the decimal point. Furthermore, all intermediate computations during the *LDLT* decomposition are done with twice as much precision to the right of the decimal point. After, the *LDLT* decomposition we truncate the numbers to K bits of precision.

In order to test if the number of bits of precision K was enough to obtain the correct output, we perform the same computation with an increased number of bits of precision. Only when the eigenvalues calculated match the eigenvalues calculated with precision up to 10^{-15} do we record them as the correct output.

4.2. Sketch of algorithm

Frequently it is much easier to compute the largest eigenvalue of a matrix rather than the smallest eigenvalue, see for example [11]. At the same time, the smallest eigenvalue of an invertible matrix is the largest eigenvalue of its inverse.

Therefore, it might prove advantageous to treat the problem of calculating the smallest eigenvalue of a Hankel matrix as a problem of finding the inverse of the Hankel matrix and subsequently finding its largest eigenvalue. The challenge with this approach is to compute the inverse of the Hankel matrix H_N^{-1}.

Conveniently for us, it was not necessary to compute all entries of H_N^{-1}. It turns out that if one is interested only in the largest eigenvalues of H_N^{-1}, one can discharge all entries of H_N^{-1} except a small top-left $k \times k$ section, and compute the largest eigenvalues of that $k \times k$ section. We argue this claim in the following section. Moreover, k does not need to be large. We found that $k = 8$ was enough to achieve 10^{-12} precision for $N = 4500$, see Table 2.

4.3. *Top-left* k *by* k *section of* H_N^{-1}

The authors used Wolfram Mathematica software to experiment with the inverses of Hankel matrices for $N \leq 100$ and $\beta = \frac{1}{2}$. Using rational number representation where both the numerator and denominator were integers with arbitrary-precision, it was possible to find the exact inverse for up to $N = 100$ with the computational power of a modern laptop.

Using these exact results the authors observed that, apart from a few entries at the top-left corner of the matrix, other entries are decreasing faster then factorially with increasing row and column numbers. Moreover, these entries introduce only a tiny factor when computing the largest eigenvalue of H_N^{-1}. For all $N < 100$, it was sufficient to compute $k = 6$ section to calculate the largest eigenvalue to within 10^{-15} precision. Furthermore, in order to estimate the error due to the truncation i.e. using k smaller than N, it was enough to calculate the eigenvalue for a smaller top-left section of H_N^{-1} with $k - 1$ by $k - 1$ entries and find the difference

$$\Delta\lambda_N(k) \equiv \left|\lambda_N(k) - \lambda_N\right| \geq \left|\lambda_N(k) - \lambda_N(k-1)\right| \to 0 \text{ for suitable } k,$$

where

- λ_N is the largest eigenvalue of H_N^{-1},
- $\lambda_N(k)$ is the largest eigenvalue of the top-left k by k section of H_N^{-1},
- $\lambda_N(k-1)$ is the largest eigenvalue of the top-left $k-1$ by $k-1$ section of H_N^{-1},
- $\Delta\lambda_N(k)$ is the error.

This relation, although not formally proved for any N, is a direct consequence of the fact that the entries of H_N^{-1} decrease very rapidly, which continued to be the case for any N. For example for $N = 4500, k = 8,$ the top-left k by k section of H_{4500}^{-1} reads

$$\begin{bmatrix}
5.01 & -8.03 & 3.84 & -0.851 & 0.107 & -8.52\times10^{-3} & 4.66\times10^{-4} & -1.85\times10^{-5} \\
-8.03 & 19.1 & -10.5 & 2.51 & -0.328 & 2.70\times10^{-2} & -1.51\times10^{-3} & 6.07\times10^{-5} \\
3.84 & -10.5 & 6.29 & -1.58 & 0.214 & -1.80\times10^{-2} & 1.02\times10^{-3} & -4.18\times10^{-5} \\
-0.851 & 2.51 & -1.58 & 0.410 & -5.70\times10^{-2} & 4.90\times10^{-3} & -2.83\times10^{-4} & 1.17\times10^{-5} \\
0.107 & -0.328 & 0.214 & -5.70\times10^{-2} & 8.10\times10^{-3} & -7.06\times10^{-4} & 4.13\times10^{-5} & -1.73\times10^{-6} \\
-8.52\times10^{-3} & 2.70\times10^{-2} & -1.80\times10^{-2} & 4.90\times10^{-3} & -7.06\times10^{-4} & 6.24\times10^{-5} & -3.69\times10^{-6} & 1.55\times10^{-7} \\
4.66\times10^{-4} & -1.51\times10^{-3} & 1.02\times10^{-3} & -2.83\times10^{-4} & 4.13\times10^{-5} & -3.69\times10^{-6} & 2.20\times10^{-7} & -9.33\times10^{-9} \\
-1.85\times10^{-5} & 6.07\times10^{-5} & -4.18\times10^{-5} & 1.17\times10^{-5} & -1.73\times10^{-6} & 1.55\times10^{-7} & -9.33\times10^{-9} & 3.99\times10^{-10}
\end{bmatrix}$$

Therefore, we expect that we can continue to use this relation to estimate the truncation error for any N, as long as k is sufficiently large.

Furthermore, the same argument should hold for $0 < \beta < \frac{1}{2}$ as well as for $\beta > \frac{1}{2}$, as long as one chooses the appropriate k.

4.3.1. *Wider applicability of the algorithm*

The algorithm discussed in this paper for finding the largest eigenvalues of a matrix M should be applicable to other matrices, provided that the entries of M decrease sufficiently quickly[4] with the column and row numbers. Such a matrix might arise in other systems, for example, when the coupling between the elements of the system gets stronger in a certain direction.

4.4. LDLT *algorithm*

In order to find the first $k \times k$ entries of H_N^{-1} the authors have employed *LDLT* matrix decomposition. In this subsection, we start by describing the sequential algorithm for computing the *LDLT* matrix decomposition. In the next subsection, we follow by describing the parallelised version of the *LDLT* algorithm.

It is often considered the paradigm of numerical linear algebra that an algorithm must correspond to a matrix factorisation. For the matrices we considered, i.e. Hankel matrices that are symmetric positive-semi-definite, it is natural to use

[4] If the entries of M do not decrease sufficiently quickly one might need to $k \approx N$, which would result in a suboptimal algorithm.

Cholesky factorisation. In order to avoid the square root operations, we have to use the *LDLT* variant of Cholesky factorisation

$$H = LDL^T$$

where[5] L is a lower triangular matrix with ones along the diagonal and D is a diagonal matrix.

In order to obtain the matrix factorisation in terms of L and D, we employed the algorithm presented in the box named Algorithm 1. Its outer loop is over the columns of A from left to right. The i^{th} column of A we will call A_i. For each of those columns A_i, we first divide it by the diagonal entry i.e. $A_i[i] = A_{i\,i}$. The result we call B_i. Subsequently we loop over all entries of the matrix to the right of A_i and subtract

Algorithm 1 Serial LDLT code

```
for (i=1 to N) {
    // this loop precomputes $B_i$
    for (j=i+1 to N)
      $B_i[j] = A_i[j]/A_i[i]$;
    // these loops apply $A_i$ and $B_i$ to all entries to the right of $A_i$
    for (j=i+1 to N)
      for (k=j to N)
      $A_j[k] = A_j[k] - A_i[j] * B_i[k]$;
}
```

```
A[j][k] = A[j][k] - A[i][j]*B[i][k]
```

Notice that the term we subtract i.e. $A_i[j]*B_i[k]$ involves only the A_i picked by the outer loop and the B_i that we calculated at this step of the loop.

It is important to stress that the divisions involved in calculating the vector B_i

```
B[i][j] = A[i][j] / A[i][i]
```

are very expensive computationally. Therefore, significant amount of time is saved by precomputing it as indicated in the Algorithm 1 box rather than subtracting

```
A[j][k] = A[j][k] - A[i][j]*A[i][k]/A[i][i].
```

4.4.1. Parallelising LDLT

In order to parallelise the *LDLT* decomposition on a computing cluster we followed the steps of [9]. We assigned each column A_i to one and only one of the nodes on the cluster. In order to assign similar amount of work to each node, we assigned columns to nodes using a (balanced) round robin approach. For n nodes this approach assigns first n columns to nodes 1 up to n and then next n columns to nodes n down to 1, see Table 1. This process gets repeated until each column is assigned to a node. This approach was found to produce satisfactory results for balancing the computational load and memory requirements on a homogeneous system in [9] and no further improvements were introduced in that paper.

Splitting the columns between nodes enables the parallelisation of the “for (j=i+1 to N)” loop in the Algorithm 1 over the nodes, see Algorithm 2 box.

4.4.2. Communication between nodes

In order to to distribute data between nodes and communicate between them we use OpenMPI library to implement MPI. As indicated in Algorithm 2 box, initial distribution of the data between the nodes is done using MPI broadcasts. Subsequently at each loop iteration we broadcast A_{i+1} (and part of B_{i+1}) from the node that is assigned A_{i+1} to all the other nodes.

We broadcast A_{i+1} (and part of B_{i+1}) as soon as they are calculated i.e. at ith step of the loop, see Algorithm 2. Moreover, the communication is run in a separate thread that allows the transmission to overlap with the computation.

When sending B_{i+1} we are faced with a choice, either:

- we can broadcast the whole A_{i+1} and make each node compute B_{i+1} locally,
- or compute B_{i+1} on the node that is assigned column A_{i+1} and broadcast both A_{i+1} and B_{i+1} to the other nodes.

[5] The matrix H is positive semi-definite, as a result, the *LDLT* factorization will be numerically stable without pivoting. Therefore, pivoting is not necessary. Further, the authors suspect that for the case of Hankel matrices in Eq. (2.3) any permutations would lead to slower execution and higher numerical errors.

Table 1
The balanced round robin assignment of columns to nodes.

column 1	→	node 1
column 2	→	node 2
...		...
column $n-1$	→	node $n-1$
column n	→	node n
column $n+1$	→	node n
column $n+2$	→	node $n-1$
...		...
...	→	node 2
...	→	node 1
...	→	node 2
...		...
column $N-1$	→	...
column N	→	...

Algorithm 2 The parallel version of the LDLT algorithm to be run by each node independently.

```
Assign the columns to nodes in a balanced round robin fashion
Broadcast the values needed to construct the initial matrix
Compute $B_1$
for (i=1 to N) {
   if(column i+1 is assigned to this node) {
    Apply the $B_i$ to column $A_{i+1}$
    Compute $B_{i+1}$
    Initiate background transmit of $A_{i+1}$ (and a part of $B_{i+1}$)
    Apply $A_i$ and $B_i$ to all $A_{i+2}$... $A_N$ assigned to this node.
    Wait for transmit to complete
   }
else {
    Initiate background receive of $A_{i+1}$ (and a part of $B_{i+1}$)
    Apply $A_i$ and $B_i$ to all $A_{i+1}$... $A_N$ assigned to this node.
    Wait for receive to complete
    Compute any missing $B_{i+1}$ elements - discussed in detail below
   }
}
```

On one hand, the time needed to broadcast B_{i+1} is significantly longer than the time needed to compute it. On the other hand, for small values of i there is a significant amount of idle communication bandwidth. We use this bandwidth to broadcast both A_{i+1} and B_{i+1} in parallel to the computation. However, for smaller values of i we limit the portion of B_{i+1} that we broadcast and let each node calculate the remaining part of B_{i+1}. The final transmission algorithm is summarized in Algorithm 3:

Algorithm 3 The A_{i+1} and B_{i+1} transmission algorithm.

```
serialize the $A_{i+1}$ column
    send the $A_{i+1}$ column
    break the $B_{i+1}$ column into chunks of 100 values
    while (... there are more chunks... and
       ... at least 8000 more multiplications to perform...) {
     send the next chunk of $B_{i+1}$
    }
```

The thresholds of 100 values and 8000 multiplications as presented in the Algorithm 3 box were found empirically by the authors of [9] “to work well on a variety of problem sizes and cluster geometries”. This paper did not try to improve upon those.

4.4.3. Hybrid MPI and OpenMP parallelisation

We have implemented the Algorithm 2 using both MPI and OpenMP to manage the parallelism. In the preceding subsection, we have just described how MPI is used to distribute columns between nodes. In contrast, OpenMP is used to spread the computation assigned to a particular node across multiple cores within a node. In particular, we have used OpenMP to parallelise the inner most loops associated with

- compute B_i,
- apply the A_i and B_i to all A_{i+1}... A_N assigned to this node;

see Algorithm 2 box. The second of those loops is a loop over both rows and the columns assigned to a node. A primitive parallelisation would involve OpenMP parallelising one of those loops

```
for (j=i+1 to N) {
   #pragma omp parallel for
   for (k=j to N)
     $A_j[k] = A_j[k] - A_i[j] * B_i[k]$;
}
```

However, it was noticed in [9] that the implicit barrier at the end of the innermost loop resulted in significant waste of computing time. Following [9] we have implemented A as a single index array which could be iterated over with a single loop

```
# pragma omp parallel for threadprivate(j,k) schedule(dynamic,chunk_size)
for (index=lastIndex to firstIndex) {
   j=... decode j from index...
   k=... decode k from index...
   $V_{index} = V_{index} - A_i[j] * B_i[k]$;
}
```

This loop is equivalent to the double loop above. Further improvements come from the following:

- dynamic scheduling - The computation time of each loop iteration varies considerably depending on the location in the matrix. Dynamic scheduling uses a queue to store block of work and assigns them the processor thread when they become available.
- *chunk_size* - Dynamic scheduling introduced a trade off. The larger the block size, the more time is wasted at the final barrier at the end of the loop; however, the smaller block size, the larger the overhead becomes due to queue management overhead. We have used

$$chunk\_size = \max\left(5, \frac{number\_of\_loop\_iterations}{200 * number\_of\_OpenMP\_threads}\right). \tag{4.2}$$

- We make the OpenMP loop run down through the indices. The underlying reason for going backwards is that the matrix entries become smaller toward the upper left corner of the matrix. This leads to the smaller execution blocks that lead to less time wasted at the final barrier at the end of the loop.

4.5. *Finding the inverse of* L

In this subsection we will show that the *LDLT* matrix decomposition described in the previous section is the most computationally heavy step to find the first k by k entries of H_N^{-1}. Given L and D such that

$$A = LDL^T,$$

in order to find the inverse

$$H_N^{-1} = (L^{-1})^T D^{-1} L^{-1},$$

or in index notation

$$\left(H_N^{-1}\right)_{jk} = \sum_{l=1}^{N} L_{lj}^{-1} L_{lk}^{-1} / D_{ll}, \tag{4.3}$$

we need to find L^{-1}. We find the inverse of L by performing in-place Gauss-Jordan raw elimination. Let us first present the sequential version of the algorithm‘

Moreover, we are only interested in the first m by m entries of H_N^{-1}, therefore, we are interested only the first N by m entries of L^{-1}. These can be found by a significantly faster algorithm, see Algorithm 4 box below.

Algorithm 4 The sequential version of in-place Gauss-Jordan raw elimination.

```
for (i=0 to N-1)
{
   for (j=i+1 to N-1)
   {
     f=A[j][i];
     A[j][i]=-f;
     for(k=0 to i-1)
       A[j][k]=A[j][k] - f*A[i][k];
   }
}
```

4.5.1. Parallelisation of Gauss-Jordan elimination

We parallelised the Gauss-Jordan elimination in a very similar way to the *LDLT* decomposition. We have distributing the rows of A over the nodes. We loop over rows of A from top to bottom. At iteration number j we broadcast the row number i of i to the rest of the nodes. After the transmission is finished, each node updates all rows assigned to it

```
A[k][j]=A[k][j] + A[i][j]*A[k][i];
```

We summarise the parallelised algorithm in Algorithm 5 box.

Algorithm 5 Sequential Gauss-Jordan raw elimination for the first N by m entries of L^{-1}.

```
for (i=0 to N-1)
{
   for (j=i+1 to N-1)
   {
    f=A[j][i];
    if(i < m)
       A[j][i]=-f;
   for(k=0 to min(m,i)-1)
       A[j][k]=A[j][k] - f*A[i][k];
   }
}
```

There are some important differences between how *LDLT* decomposition and Gauss-Jordan elimination parallelised.

- Unlike for *LDLT* decomposition, we use no OpenMP parallelisation. It would be natural to use OpenMP to parallelise the innermost for loop. However, we noticed that due to the upper limit $(= n)$ the overhead was much larger than possible gains for small $m \sim 10$).
- Due to small size of the broadcasted message which contains at maximum m entries, we did not need a parallel communication and computation.[6] Table 2 shows that, during out tests, the time spend communicating in the parallelised Gauss-Jordan was very short.

4.6. Finding the inverse of Hankel matrix

Having finished the Algorithm 4, we obtain the first N by k entries of L^{-1} in place of the first N by k entries of L. Subsequently, we can perform the sum

$$\left(H_N^{-1}\right)_{jk} = \sum_{l=1}^{N} L_{lj}^{-1} L_{lk}^{-1} / D_{ll}$$

to obtain the first $k \times k$ entries of H_N^{-1}. We perform the sum on the first node. The other nodes send the relevant entries of L^{-1} as they become needed. We cast the final values of H_N^{-1} into double precision floating point numbers. Finally, we use GNU Scientific Library to find the largest 2 eigenvalues of the $k \times k$ truncated H_N^{-1}.

When truncating H_N^{-1} to the first $k \times k$ entries, we introduce a truncation error. In Section 4.2 we have argued that the truncation error is small, however, we would like to estimate it. For that purpose we calculate the largest 5 eigenvalues of $(k-1) \times (k-1)$ truncated H_N^{-1}. We then use the difference between the eigenvalues for k truncation and $k-1$ truncation to estimate the truncation error, see Table 2.

[6] One might suspect that an overhead from starting a parallel communication might actually result in a slower over all performance for small $m \sim 10$.

Algorithm 6 The parallelised in-place Gauss-Jordan raw elimination for the first N by k entries of L^{-1}.

```
Assign the columns to nodes in a balanced round robin fashion
for (i=0 to N-1)
{
   n = the smallest of i and m
    if(row i is assigned to this node)
     Broadcast first n entries of i row of A to the other nodes
    else
     Receive first n entries of i row of A
    for (j=i+1 to N-1)
    {
     if(row j is assigned to this node)
    {
       f=A[j][i];
       if(i < k)
         A[j][i]=-f;
       for (k=0 to n-1)
         A[j][k]=A[j][k] - f*A[i][k];
     }
   }
}
```

5. Numerical results

In Section 4, we have presented a new numerical algorithm for computing the smallest eigenvalues of Hankel matrices, which we have implemented the algorithm in C programming language. In this section, we present the numerical results obtained using the new implementation.

5.1. Computed eigenvalues

In this subsection, we present the computations we performed on High-Performance Computing Cluster of the University of Macau. Each computation was performed on 3 computing nodes, each node with 2 Intel Xeon E5-2690 v3 E5-2697 v2 CPUs and 64 GB of RAM. Each CPU had 12 cores/24 threads and 30 MB of cache. Therefore, in total, we had 36 cores and 192 GB of RAM available.

The numerical results obtained together with the timing profiles for the corresponding calculations we have presented in Table 2 on page 21. The table columns correspond to

N is the rank of the matrix.
Required Precision is the minimum precision of arithmetic operations K to make the numerical error less than 10^{-15}. We were increasing the value of K in the steps of 1024. The quoted minimum required precision K was the smallest multiple of 1024 such that the outputs of both K and $K+1024$ matched to 10^{-15}. For more details see Section 4.1 on page 9.
Number of nodes - the number of nodes used for the computation. For each computation, we have checked that, during the computationally heavy part i.e. *LDLT* decomposition, we have used 24 cores at (close to) 100% at each node.
Total Wall time is the total time taken by the computation as measured by “a clock on the wall”.
LDLT time is the time taken by the *LDLT* decomposition part of the algorithm.
Inversion of L time is divided into time spent performing arithmetic operations involved in finding the inverse of L, and time spent communicating between the nodes. We also quote the sum the above.
Transpose time is the time taken to swap between entire columns being assigned to a particular node for the LDLT decomposition and entire rows being assigned to a particular node for the inversion of L.
Inverse of H_N is the time taken by multiplication of two L^{-1} and D^{-1} matrices to find the truncated H_N^{-1}.
Truncation error is the estimate of the error resulting from truncation of H_N^{-1} to k by k size, see Section 4.6 for details.

All of the above times were measured on the host node i.e. the node that was recruiting other nodes to store assigned columns and to perform operations on the assigned columns see Section 4.

There are a few important things to notice in Table 2. First, the inversion of L time constitutes only a few percents of the total wall times taken by the computation, which demonstrates that the *LDLT* decomposition is the computationally heavy part of the computation. Second, the time taken by matrix multiplication to find H_N^{-1} constitutes an even smaller fraction of the total wall times taken by the computation, even though, it was performed on a single node. Third, the “Send/Receive”

Table 2
Numerically calculated smallest eigenvalues of Hankel matrix with $\beta = \frac{1}{2}$ together with the required precision and timing data of the corresponding computation. For an explanation of the column names see Section 5.1.

N	Precision Required	No. of nodes	Total wall time	LDLT time	Transpose time	**Inversion of *L*time**			Inverse of H_N	Calculated eigenvalues					
	[bits]		[s]	[s]	[s]	Total	Arithmetics	Send/ Receive	[s]	Smallest	Truncation error	2^{nd} smallest	Truncation error	3^{nd} smallest	Truncation error
500	1024	3	20.4	11.5	0.587	0.958	0.813	0.143	6.84	0.1204653471966412	-6.52×10^{-16}	1.116696239796391	-2.75×10^{-14}	33.53605844924584	-3.45×10^{-11}
1000	2048	3	136	109	3.83	11.7	11.5	1.23	9.17	0.08208748342129053	-5.57×10^{-15}	0.8694471685364237	-2.51×10^{-13}	16.74741576006559	-1.41×10^{-10}
1500	3072	3	741	645	12.5	50.7	45.8	5.00	28.9	0.06529477501882298	-1.75×10^{-14}	0.7587290286009394	-8.72×10^{-13}	11.56571839375061	-2.98×10^{-10}
2000	4096	3	2586	2374	28.3	142	129	12.9	33.4	0.05543072589537470	-3.68×10^{-14}	0.6903595403385252	-2.02×10^{-12}	9.032809963814945	-4.95×10^{-10}
2500	4096	3	6717	6298	52.9	297	271	26.6	53.2	0.04878757749929328	-6.39×10^{-14}	0.6418871023190091	-3.80×10^{-12}	7.522048961034497	-7.23×10^{-10}
3000	5120	3	15343	14578	94.9	581	530	51.0	61.6	0.04394036934849594	-9.83×10^{-14}	0.6047656504596126	-6.28×10^{-12}	6.513621908349736	-9.77×10^{-10}
3500	6144	3	29001	27818	140	940	933	6.99	71.4	0.04021149503682476	-1.40×10^{-13}	0.5749057442617896	-9.52×10^{-12}	5.789838905207254	-1.25×10^{-09}
4000	7168	3	56445	54426	227	1645	1502	142	86.3	0.03723304780176154	-1.88×10^{-13}	0.5500577166937035	-1.36×10^{-11}	5.243332063875005	-1.55×10^{-09}
4500	7168	3	94242	91303	325	2436	2217	218	94.1	0.03478615399760864	-2.43×10^{-13}	0.5288610646385768	-1.84×10^{-11}	4.814940754432488	-1.87×10^{-09}

Table 3
Comparison between the new algorithm and Secant algorithm used in [9]. For an explanation of the column names see Section 5.1.

	Secant				New algorithm			
N	Precision Required [bits]	Wall time [s]	No. of iterations	Average iteration [s]	Precision Required [bits]	Wall time [s]	LDLT time [s]	Inv. L time [s]
200	512	230.6	8	25.6	388	22.4	22.2	0.037
400	1024	505.4	8	56.15	758	55.1	54.2	0.527
600	1024	1652.9	11	137.7	1024	134.9	131.8	2.24
800	2048	2395.7	8	266.2	1536	376.4	367.9	6.79
1000	3072	8542.4	7	1067.8	2048	954.6	935.5	15.7

time is a small fraction of the “Inversion of L time”, therefore, parallelising the communication and arithmetic would not lead to significant speed-up.

5.2. Timing: the new algorithm against Secant algorithm

The authors of [9] compared the Secant algorithm with a number of classical eigenvalue algorithms including Householder, Jacobi, Lanczos on the task of finding the smallest eigenvalue of Hankel matrices. It was found that, for the case of large and extremely ill-conditioned matrices, Secant algorithm proved to be much more efficient. In this paper, we try to improve on the Secant algorithm with the new algorithm presented in Section 4. Here we would like to compare the timing of the new algorithm implementation against the implementation of Secant algorithm provided by [9].

In Table 3 we have compared the time needed to compute the smallest eigenvalue of N by N Hankel matrix from Eq. (2.3) to 10^{-15} numerical accuracy with the two algorithms for different values of N. For each N we have scanned over precision K, defined in Eq. (4.1), and chosen the minimum value that produced the desired accuracy and called it the “required precision”. The values of precision K we scanned over were 256, 388, 512, 768, 1024, 1536, 2048, 3072, 4096 bits. The computation was performed on Thinkpad T480 with Intel Core i5-8250U Processor using close to 100% of all 8 threads for both implementations.

The table contains the number of iterations the Secant algorithm needs to converge on the zero of the characteristic polynomial of the matrix for each N, let us call it $n_{iterations}$. The total wall time taken by the Secant implementation was composed of $n_{iterations}$ *LDLT* decompositions plus the initial decomposition. Therefore

$$t_{\text{Wall Secant}} = (n_{iterations} + 1) \times t_{\text{Average iteration Secant}}$$

Looking at Table 3, we can notice two improvements over the Secant algorithm:

- The computation time required by the new implementation is very close to the time required for the Secant implementation to complete an average iteration. Therefore, the new implementation is $\sim (n_{iterations} + 1)$ times faster. This is to be expected as the new algorithm performs the computationally heavy *LDLT* decomposition only once, whilst the Secant algorithm does that $(n_{iterations} + 1)$ number of times.
- We notice that the precision K required to compute the smallest eigenvalue with the desired precision is typically significantly smaller for the new implementation. This can be attributed to the fact that the Secant algorithm finds a zero of the characteristic polynomial whose values are typically astronomically large, and therefore, finding the zero to satisfactory precision is harder than finding a “satisfactory” *LDLT* decomposition. This leads to small but significant speed gains for larger matrices.

5.3. Scaling with number of nodes

Authors of [9] extensively discuss how and present evidence that their implementation of Secant algorithm scales well with the number of nodes. In this section, we would like to check that this statement extends to the new algorithm.

We have timed our implementation of the new algorithm for a different number of nodes. We have used Google Cloud Platform to create a cluster of 6 “f1-micro” virtual machines[7] and run our implementation on a various number of nodes to check the scaling of the computation time with the number of nodes. The results are presented in Table 4. The row names follow the same convention as Table 2 with one difference; we also quote the total CPU time which is the sum of time consumed by all of the CPUs utilized by the program. For this setup we estimate the CPU time using

$$t_{CPU} \simeq n_{nodes} \times t_{Wall}.$$

We can see that the total CPU time only slowly increases with the number of nodes, which leads to the total wall time steadily decreasing with the number of nodes. We can see that we achieve parallelisation over the nodes of the cluster, and it scales well with the number of recruited nodes.

[7] These machines have 2.5 GHz CPU and 0.6 GB of memory.

Table 4
Timing test of the scalability of the algorithm with the number of nodes participating in the calculation. For an explanation of the column names see Section 5.3.

Number of nodes		2	3	4	5	6
Total wall time		29.3	20.1	16.5	13.8	12.6
Total CPU time		58.6	60.4	66.1	69.2	75.5
LDLT time	**Total wall**	27.4	18.4	14.7	12.3	11
	Total CPU	54.9	55.1	58.9	61.7	66.3
Inverse of L time	**Total wall**	0.672	0.465	0.366	0.301	0.263
	Total CPU	1.34	1.40	1.46	1.51	1.58
	Arithmetics	0.608	0.424	0.311	0.257	0.205
	Send/Receive	0.0641	0.041	0.055	0.044	0.058
Inverse H_N time		0.22	0.158	0.175	0.169	0.176
Eigenvalue time		0.004	0.034	0.034	0.002	0.002

6. Leading exponent determination

In this section, we use the numerically calculated eigenvalues for $\beta = \frac{1}{2}$ from Section 5.1 to compare it to Eq. (3.14) and extract the leading exponent introduced in Section 3.2.

The authors of [9] have also compared eigenvalues they have numerically calculated for $\beta = \frac{1}{2}$ to Eq. (3.1). Their approach was to directly compare the numerically calculated eigenvalue and the corresponding value predicted by Eq. (3.1). They have observed a significant difference[8] even for relatively large N e.g. $> 25\%$ for $N = 1500$. These differences are due to the subleading in N order term not captured by Eq. (3.14).

In contrast, we would like to concentrate on the scaling of the smallest eigenvalue with N

$$\lambda_1(N) \propto N^{-\frac{2}{\pi}}\sqrt{\log N}$$

and emphasise the algebraic factor $N^{\frac{-2}{\pi}}$ in our analysis. For that purpose, we recast Eq. (3.14) which reads

$$\lambda_1(N) \simeq \frac{8\pi\sqrt{\log N}}{(4\pi Ne)^{\frac{2}{\pi}}}, \tag{6.1}$$

as

$$\log\left[\frac{8\pi}{\lambda_1(N)}\sqrt{\log N}\right] \simeq \frac{2}{\pi}\log(4\pi Ne). \tag{6.2}$$

We notice that, if we plot $\log[\frac{8\pi}{\lambda_1(N)}\sqrt{\log N}]$ against $\log(4\pi Ne)$ for large N, then we should find a straight line with the gradient equal to the leading exponent i.e. $\frac{2}{\pi}$. For smaller N, we would expect some deviation from a straight line due to the subleading terms not captured by Eq. (6.1).

This picture emphasises the leading exponent. Moreover, as we have noted in Section 3.2, the subleading terms do not affect the overall $N^{-\frac{2}{\pi}}$ factor, and they are suppressed by a double log-function. Therefore, we would expect the leading exponent determination to be less affected by the subleading terms than the direct comparison performed by [9].

6.1. Linear fit

We plotted the eigenvalues presented in Table 2 on a x-y plot where

$$x \equiv \log(4\pi Ne), \tag{6.3}$$

$$y \equiv \log\left(\frac{8\pi}{\lambda_1(N)}\sqrt{\log N}\right), \tag{6.4}$$

in Fig. 1. Subsequently, we have fit the data points with a linear model. We have included a constant term in the linear model to allow for the differences for smaller values of N. We weighted each point proportionally to N^2 to put higher weight on the larger values of N and found the best fit is

$$y = 0.04630 + 0.63646 \times x$$

with adjusted R-squared $= 0.999997$, and residuals presented in Fig. 2.

[8] The difference is significantly smaller when we compare the numerics to Eq. (3.14) instead of Eq. (3.1).

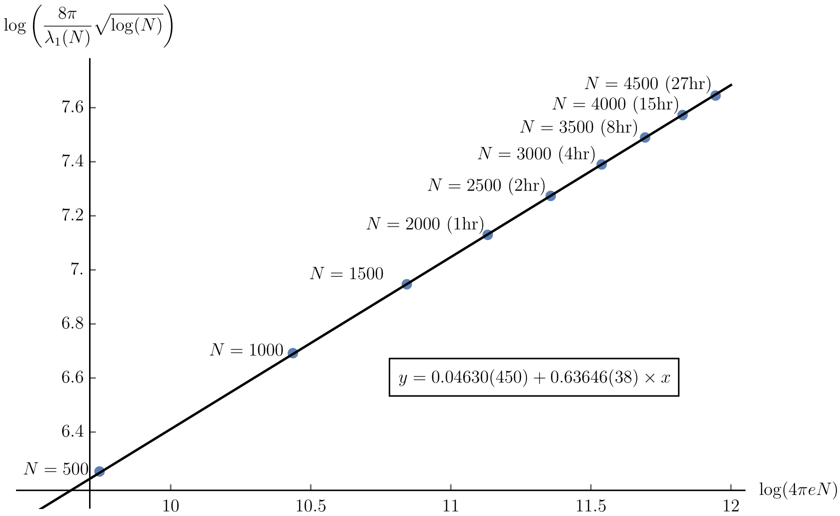

Fig. 1. Plot of the numerically calculated eigenvalues from Table 2 with a fitted linear model.

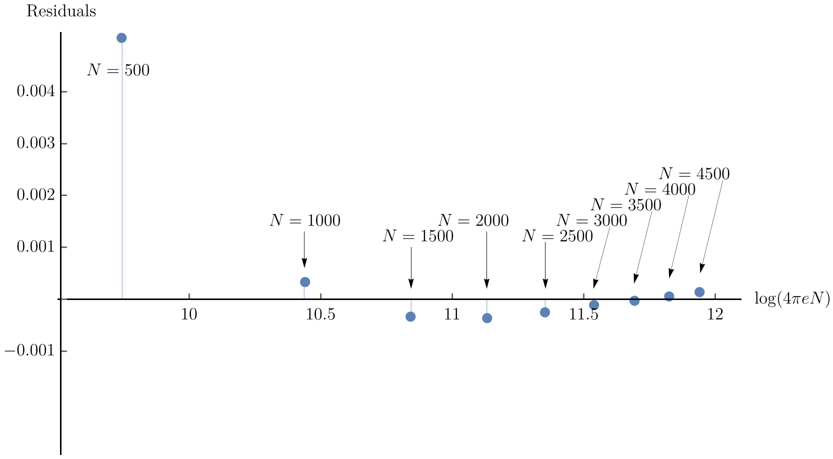

Fig. 2. The residuals of the linear fit in Fig. 1.

We can see that the residuals tend to decrease as N increases. The gradient value is only 0.025% away from the predicted value $\frac{2}{\pi} \simeq 0.63662$, i.e.

$$\frac{0.63662 - 0.0.63646}{0.63662} \simeq 0.00025$$

Moreover, the 95% confidence interval[9] for the gradient [0.63510, 0.63781] contains the $\frac{2}{\pi}$. Therefore, we conclude that the smallest eigenvalue of the Hankel matrices in Eq. (2.3) at the critical point $\beta = \frac{1}{2}$ decays algebraically $\lambda_1(N) \sim N^{-\frac{2}{\pi}}$ for large N, and the numerical results are a strong evidence that Eqs. (3.4) and (3.14) are true asymptotically for large N.

7. Discussion

In this paper, we have introduced a novel algorithm for finding the smallest eigenvalues of a matrix that is particularly suited for Hankel matrices generated by the probability density function $w(x) = e^{-x^{\beta}}$. We have developed an implementation of the algorithm that runs on a high performance computing cluster making use of both shared memory parallelisation within a node and a distributed memory parallelisation between nodes of the cluster. We have shown that the new algorithm improves on Secant algorithm previously considered to be the state of the art algorithm to study of Hankel matrices [9]. Moreover, we show that it scales well with the number of participating nodes.

[9] We should mention that the points in Fig. 1 have negligible (numerical and truncation) error bars associated with them, but since we are fitting the model parameters to the data points those model parameters acquire statistical errors.

We have employed the novel algorithm to find the smallest eigenvalues of Hankel matrices at the critical point $\beta = \frac{1}{2}$ for matrices up to $N = 4500$. We have plotted the numerical results on a custom log-log plot and determined that at the leading exponent is $-\frac{2}{\pi}$, therefore

$$\lambda_1(N) \simeq N^{-\frac{2}{\pi}},$$

where $\lambda_1(N)$ is the smallest eigenvalue of the Hankel matrix in Eq. (2.3) at $\beta = \frac{1}{2}$ critical point.

7.1. Significance of the $\beta = \frac{1}{2}$ result

Berg et al. [3] proved that the minimum eigenvalues of Hankel matrices generated by probability density function $w(x)$

$$(H_N)_{ij} = \int_0^\infty x^{i+j} w(x) dx, \quad 0 \le i, j \le N-1,$$

is bounded away from zero for all N including N equals infinity if and only if the associated moment problem has more than one solution. For the weight $w(x) = \mathrm{e}^{-x^\beta}$, $x \ge 0$, $\beta > 0$, it can be shown that the smallest eigenvalue tends to zero exponentially fast in N if $\beta > \frac{1}{2}$. At $\beta = 1/2$ which is a critical point, the moment problem is at the verge of being indeterminate, and the smallest eigenvalue tends to zero algebraically in N. For $0 < \beta < \frac{1}{2}$, the smallest eigenvalue tends to a strictly positive constant whose value depending on β has not yet been found.

7.2. Preconditioner

The authors suspect that there might be space for further improvements in the design of the numerical algorithm to calculate the smallest eigenvalue. The Hankel matrix that we consider has particularly natural and effective preconditioner and post-conditioner that make its condition number much more tame. One can consider

$$\left(\tilde{H}_N\right)_{ij} = \frac{\Gamma\left(\frac{i+j-1}{\beta}\right)}{\sqrt{\Gamma\left(\frac{2i-1}{\beta}\right)\Gamma\left(\frac{2j-1}{\beta}\right)}}, \quad \text{for } i, j = 1, 2, 3\ldots$$

which has a nice property that $(\tilde{H}_N)_{ij} = 1$ for $i = j$ and a significantly smaller condition number. Therefore, it might be significantly easier to find the determinant of $(\tilde{H}_N)_{ij}$ than the original $(H_N)_{ij}$. However, $(\tilde{H}_N)_{ij}$ is just the original Hankel matrix pre-multiplied and post-multiplied by two diagonal matrices. Therefore, the determinant of the original $(H_N)_{ij}$ is just the determinant of $(\tilde{H}_N)_{ij}$ multiplied by the diagonal terms of pre and post-multiplier. This sounds like a powerful idea to improve Secant algorithm, which relays on a repeated evaluation of (modified) determinants of Hankel matrix. However, we were surprised to find out that the pre-conditioner and post-conditioner did not bring any efficiency improvements to the Secant algorithm. This might be partially explained by the fact that $(\tilde{H}_N)_{ij}$ still has a very large condition number, though much smaller than $(H_N)_{ij}$. We feel that this situation deserves further clarification, and we leave it as a future direction.

Acknowledgements

Y. Chen, J. Sikorowski and M. Zhu would like to thank the Science and Technology Development Fund of the Macau SAR for generous support in providing FDCT 130/2014/A3 and FDCT 023/2017/A1. We would also like to thank the University of Macau, Macau, China for generous support via MYRG 2014-00011 FST, MYRG 2014-00004 FST and MYRG 2018-00125 FST.

The authors would like to thank a referee for a careful reading of the manuscript, and for reminding us to add the Section 4.3 for supporting our main idea. We would also like to thank Prof. Gordon Blower for English proofreading.

References

[1] N.I. Akhiezer, The classical moment problem and some related questions in analysis, Edinburgh: Oliver and Boyd (1965).

[2] B. Beckermann, The condition number of real Vandermonde, Krylov and positive definite Hankel matrices, Numer Math. 85 (2000) 553–557.

[3] C. Berg, Y. Chen, M.E.H. Ismail, Small eigenvalues of large Hankel matrices: the indeterminate case, Math. Scand. 91 (2002) 67–81.

[4] C. Berg, R. Szwarc, The smallest eigenvalue of Hankel matrices, Constr Approx 34 (2011) 107–133.

[5] Y. Chen, M.E.H. Ismail, Thermodynamic relations the Hermitian matrix ensembles, J. Phys. A Math. Gen. 30 (1997) 6633–6654.

[6] Y. Chen, N. Lawrence, Small eigenvalues of large Hankel matrices, J. Phys. A Math. Gen. 32 (1999) 7305–7315.

[7] Y. Chen, D. Lubinsky, Smallest eigenvalues of Hankel matrices for exponential weights, J. Math. Anal. Appl. 293 (2004) 476–495.

[8] N.G. De Bruijn, Asymptotic Methods in Analysis, Interscience, New York, 1958.

[9] N. Emmart, Y. Chen, C. Weems, Computing the smallest eigenvalue of large ill-conditioned Hankel matrices, Commun. Comput. Phys. 18 (2015) 104–124.

[10] M.E.H. Ismail, Classical and quantum orthogonal polynomials in one variable, in: Encyclopedia of Mathematics and its Applications, 98, Cambridge University Press, Cambridge, UK, 2005.

[11] J. Kuczyński, H. Woźniakowski, Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start, SIAM J. Matrix Anal. Appl. 13 (1992) 1094–1122.

[12] M.G. Krein, A.A. Nudelman, Markov moment problem and extremal problems, Providence RI American Mathematical Society, 1977.

[13] D.S. Lubinsky, Condition numbers of Hankel matrices for exponential weights, J. Math. Anal. Appl. 314 (2006) 266–285.

[14] M.L. Mehta, Random Matrices, third ed., Elsevier (Singapore) Pte Ltd., Singapore, 2006.

[15] G. Szegö, On some hermitian forms associated with two given curves of the complex plane, Trans. Am. Math. Soc. 40 (1936) 450–461.

[16] J. Todd, Contributions to the solution of systems of linear equations and the determination of eigenvalues, Nat. Bur. Stand. Appl. Math. Ser. 39 (1959) 109–116.

[17] H. Widom, H.S. Wilf, Small eigenvalues of large Hankel matrices, Proc. Am. Math. Soc. 17 (1966) 338–344.

[18] H. Widom, H.S. Wilf, Errata: Small eigenvalues of large Hankel matrices, Proc. Am. Math. Soc. 19 (1968) 1508.

[19] M. Zhu, Y. Chen, N. Emmart, C. Weems, The smallest eigenvalue of large Hankel matrices, Appl. Math. Comput. 334 (2018) 375–387.

[20] M. Zhu, N. Emmart, Y. Chen, C. Weems, The smallest eigenvalue of large Hankel matrices generated by a deformed Laguerre weight, Math. Meth. Appl. Sci. 42 (2019) 3272–3288.