NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. (2016)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nla.2035

On the fast Lanczos method for computation of eigenvalues of Hankel matrices using multiprecision arithmetics

Shaun Bangay and Gleb Beliakov*,†

School of Information Technology, Deakin University, 221 Burwood Highway, Burwood, 3125, Australia

SUMMARY

The use of the fast Fourier transform (FFT) accelerates Lanczos tridiagonalisation method for Hankel and Toeplitz matrices by reducing the complexity of matrix–vector multiplication. In multiprecision arithmetics, the FFT has overheads that make it less competitive compared with alternative methods when the accuracy is over 10 000 decimal places. We studied two alternative Hankel matrix–vector multiplication methods based on multiprecision number decomposition and recursive Karatsuba-like multiplication, respectively. The first method was uncompetitive because of huge precision losses, while the second turned out to be five to 14 times faster than FFT in the ranges of matrix sizes up to $n = 8192$ and working precision of $b = 32\,768$ bits we were interested in. We successfully applied our approach to eigenvalues calculations to studies of spectra of matrices that arise in research on Riemann zeta function. The recursive matrix–vector multiplication significantly outperformed both the FFT and the traditional multiplication in these studies.

Received 19 December 2014; Revised 02 December 2015; Accepted 28 December 2015

KEY WORDS: eigenvalues; Hankel matrix; Toeplitz matrix; Lanczos method; multiprecision arithmetics

1. INTRODUCTION

Numerical computation of eigenvalues is a generic problem that arises in multiple domains. Various numerical methods are collected, for instance, in [1–3]. Here, we are interested in particular in Lanczos algorithm, because it allows fast implementation for matrices with special structure, such as circulant matrices, and the related Toeplitz and Hankel matrices [4–6]. The algorithm is based on an efficient tridiagonalisation of a circulant matrix by using fast Fourier transform (FFT) for matrix–vector multiplication. The Lanczos algorithm takes $O(n^3)$ time for general matrices to compute all the eigenvalues, but thanks to FFT, it is performed in $O(n^2 \log n)$ for Hankel and Toeplitz matrices. However, as reported in [6, 7], the accuracy of this method could be quite low.

In this article, we are interested in implementing Lanczos method using multiprecision arithmetics. One of the reasons is to stabilise the algorithm so that it does not suffer from accuracy loss for larger matrices. The second reason is that in our background application, concerned with the analysis of Riemann's zeta function [8, 9], it is required to perform computations with extremely high accuracy of 10 000 decimal places and more. Thus, the use of multiprecision packages is a must.

There is a number of packages that offer arbitrary precision arithmetics, in particular Arprec, GMP, MPFR, Flint, Arb and some others [10–13]. These packages implement the standard arithmetical operations and often a number of elementary and special functions. The standard Lanczos algorithm (without FFT) can be implemented directly. However, we were able to find only one implementation of FFT in multiprecision arithmetics, which is needed for the fast multiplication algorithm for Hankel matrices, namely, the multiprecision computing toolbox for Matlab [14]. This package is based on `kissfft` library [15].

*Correspondence to: Gleb Beliakov, School of Information Technology, Deakin University, 221 Burwood Highway, Burwood, 3125, Australia.
†E-mail: gleb@deakin.edu.au

In multiprecision arithmetics, one specifies the desired accuracy in terms of the number of bits b used to hold the mantissa of the values. Multiprecision addition is relatively cheap with complexity $O(b)$; however, multiplication is significantly more expensive. Depending on the number of bits used, one or another method is invoked. In particular, for sufficiently large $b \geqslant b_{FFT}$, FFT is used for multiplication in GMP, MPFR and various derivative libraries (the threshold b_{FFT} depends on the hardware: for ×386 architecture, b_{FFT} ranges between 2816 and 7168 in GMP 4.3.2, but these values also depend on the version of the library).

One approach to fast Hankel matrix multiplication is to change the basic type in an FFT library from float or double to multiprecision. This approach was taken in the Matlab multiprecision computing toolbox [14]. However, there are also some alternatives that we wanted to explore. Because the fast multiprecision multiplication is itself based on FFT, it means that we need to perform one FFT as a basic step inside another. It might be possible to unroll these operations into just one standard FFT. On the other hand, there are several parallel FFT libraries (in single or double precision) that make use of (a) multiple cores of one CPU, (b) many cores of a graphics processing unit, and (c) multiple CPUs using message passing interface. Changing the basic scalar type in these highly optimised libraries is not simple, as their performance heavily relies on the lengths of scalar type and machine words, extensive use of cache and other hardware-related optimisations. Adjusting the length of the scalar type may drastically worsen the performance.

As the initial step, we present and compare various alternatives for fast implementation of the Lanczos algorithm in multiprecision arithmetics, based on FFT. Bearing in mind that ideally we would want to make use of the existing optimised FFT libraries, we attempted to decompose each multiprecision number into a sum of float or double precision numbers, then applied a standard FFT library and eventually reconstructed multiprecision vectors from the arrays of floats. We call this approach decomposition.

An alternative approach to fast matrix–vector multiplication, not based on FFT, is to use an economical Karatsuba-like matrix–vector multiplication in multiprecision arithmetics, developed by Beliakov [16]. Here, the multiprecision numbers are not decomposed; instead, a recursive process is applied to matrix–vector product reducing the complexity to $\Theta(n^{\log_2 3})$.

Our contribution is the derivation and a comparative numerical study of the following four alternatives for Hankel matrix–vector multiplication: (1) the decomposition approach, (2) implementation of FFT in multiprecision arithmetics, (3) Karatsuba-like matrix multiplication from [16], and (4) the direct schoolbook matrix–vector multiplication. We provide an experimental confirmation of the superiority of one or another method for specific ranges of matrix size and accuracy, in particular confirming that the recursive method from [16] offers significant numerical advantages in the range of our interest.

From the aforementioned results, we derive a new efficient method for computation of eigenvalues and eigenvectors of Hankel and Toeplitz matrices in multiprecision arithmetics based on Lanczos tridiagonalisation and partial re-orthogonalisation. We compare the numerical performance of this method with a state-of-the-art direct method based on Schur decomposition. We also propose a coarse-grained multicore paralellisation of our method for shared memory architectures.

This paper is structured as follows. In Section 2, we will formally describe the motivating problem and give a brief introduction to fast Lanzcos algorithm. In Section 3, we present the alternatives for using FFT in multiprecision arithmetics. In Section 4, we present recursive economic Karatsuba-like matrix–vector multiplication from [16]. In Section 5, we detail some of the numerical experiments and quantify the performance of our algorithms, including the loss of accuracy. Section 6 concludes.

2. MOTIVATION AND PROBLEM FORMULATION

2.1. Riemann's zeta function and eigenvalues

The need to find eigenvalues of a Hankel matrix in multiprecision arithmetics came from research on the famous Riemann's *zeta function*. The distribution of zeroes of zeta function ζ has puzzled mathematicians for over a century. The famous Riemann hypothesis [17], which was included by D. Hilbert at the very end of the 19th century as part of his eighth problem, and which is also one of

DOI: 10.1002/nla

the Clay Institute seven Millennium Problems [18], is to prove or disprove that all non-real zeroes of $\zeta(z)$ lie on the critical line $\Re(z) = \frac{1}{2}$. Among other things, Riemann's zeta function is closely related to the distribution of prime numbers, whose fundamental role in mathematics is well known.

Zeta function is defined in the complex plane by the analytic continuation of the series

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s}, \tag{1}$$

which converges for $\Re(s) > 1$. It satisfies the functional equation

$$\zeta(s) = 2^s \pi^{s-1} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s)\zeta(1-s). \tag{2}$$

Riemann also defined a symmetric version of the aforementioned, using function ξ,

$$\xi(s) = \frac{1}{2}\pi^{-s/2} s(s-1)\Gamma\left(\frac{s}{2}\right)\zeta(s), \tag{3}$$

which yields a simpler functional equation

$$\xi(s) = \xi(1-s). \tag{4}$$

The *trivial zeroes* of ζ are negative even integers, and they are its only real zeroes. The other, *non-trivial zeroes* can only be found in the strip $0 < \Re(s) < 1$. The zeroes of the function ξ are exactly the non-trivial zeroes of the function ζ.

An important relation of zeta function to prime numbers was given by von Mangoldt. Let Chebyshev function ψ be

$$\psi(x) = \sum_{q \text{ is a power of prime } p}^{q \leqslant x} \ln(p) = \ln(\mathrm{LCM}(1, 2, \ldots, \lfloor x \rfloor)).$$

Then for non-integer x greater than 1, by von Mangoldt's theorem $\psi(x)$, can be expressed as

$$\psi(x) = x - \sum_{n=1}^{\infty} \frac{x^{-2n}}{-2n} - \sum_{\xi(\rho)=0} \frac{x^{\rho}}{\rho} - \ln(2n). \tag{5}$$

The first sum runs over the trivial zeroes of zeta, and the second sum runs over the non-trivial zeroes. In fact, knowing zeroes of zeta function, one could compute primes by merely looking at the graph of the right-hand side of (5) and identifying the powers of primes by jumps in the graph.

One approach to the analysis of zeta function presented in [8] consists of studying a related function obtained from ζ by a change of variables

$$z = \frac{w}{w+1}, \quad w = \frac{z}{1-z}.$$

Under this transformation, the critical line $\Re(z) = \frac{1}{2}$ becomes the critical circle $|w| = 1$, with the half-plane $\Re(z) < \frac{1}{2}$ becoming the interior of the circle. The points

$$w_n = \frac{z_n}{1-z_n} = -\frac{2n}{2n+1}, n = 1, 2, \ldots,$$

become the trivial zeros of the function

$$\bar{\zeta}(w) = \zeta^*\left(\frac{w}{w+1}\right),$$

with $\zeta^*(z) = 2(z-1)\zeta(z)$, an auxiliary entire function used to cancel out the only pole of zeta function at $z = 1$.

DOI: 10.1002/nla

The Riemann hypothesis that all non-trivial zeros of ζ lie on the critical line can now be formulated through an equivalent sequence of subhypotheses RH_l that the trivial zeros of $\bar{\zeta}$ are the only zeros lying on the closed disk $|w| \leqslant \frac{2l+1}{2l+2}$ for every $l = 1, 2, \ldots$ [19].

The relation of the aforementioned with the eigenvalues is the following. The function $\frac{1}{\bar{\zeta}(w)}$ can be written in the Taylor series expansion

$$\frac{1}{\bar{\zeta}(w)} = 1 + \tau_1 w + \ldots + \tau_m w^m + \ldots$$

The first subhypothesis RH_1 was written in [19] as the following statement

$$\lim_{m\to\infty} ((-1)^m \tau_m)^{\frac{1}{m}} = \frac{3}{2}.$$

Next, by using the following Taylor series for $\bar{\zeta}$,

$$\bar{\zeta}(w) = 1 + \theta_1 w + \ldots + \theta_m w^m + \ldots,$$

we can express $\tau_m = (-1)^m \det(L_{1,m})$ where $L_{1,m}$ is the Toeplitz matrix

$$L_{1,m} = \begin{bmatrix} \theta_1 & 1 & 0 & \ldots & 0 \\ \theta_2 & \theta_1 & 1 & \ldots & 0 \\ \theta_3 & \theta_2 & \theta_1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_{m-1} & \theta_{m-2} & \theta_{m-3} & \ldots & 1 \\ \theta_m & \theta_{m-1} & \theta_{m-2} & \ldots & \theta_1 \end{bmatrix}.$$

Therefore, the subhypothesis RH_1 can be written in terms of the determinant of matrix $L_{1,m}$, which is naturally the product of its eigenvalues $\lambda_{1,m,n}, n = 1, \ldots, m$, and hence can be stated as

$$\lim_{m\to\infty} \left(\prod_{n=1}^{m} \lambda_{1,m,n} \right)^{\frac{1}{m}} = \frac{3}{2}.$$

Hence, the spectrum of $L_{1,m}$ is of interest. The initial calculations of the positions of the eigenvalues of $L_{1,m}$ were calculated in [19] and a number of conjectures have been made.

By considering the eigenvalues of the matrices $L_{l,m}$, equivalent subhypotheses $RH_l, l = 1, 2, \ldots$ have been presented in [19], which also relate to the spectra of the matrices $L_{l,m}$.

In the subsequent work [8, 9], the author has shifted focus to an alternative representation of the determinants of $L_{l,m}$, namely, by rearranging the columns of these matrices to acquire Hankel matrices

$$M_{l,m} = (-1)^{l+m} \begin{bmatrix} \theta_{l+m-1} & \theta_{l+m-2} & \ldots & \theta_{l+1} & \theta_l \\ \theta_{l+m-2} & \theta_{l+m-3} & \ldots & \theta_l & \theta_{l-1} \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ \theta_l & \theta_{l-1} & \ldots & \theta_{l-m+2} & \theta_{l-m+1} \end{bmatrix}. \tag{6}$$

In the earlier expression, $\theta_0 = 1$ and $\theta_k = 0$ for $k < 0$.

Clearly, $\det(M_{l,m}) = \det(L_{l,m})$; however, the spectra of the matrices $M_{l,m}$ are quite different, starting from the fact that $M_{l,m}$ are real symmetric and all eigenvalues $\mu_{l,m,n}, n = 1, \ldots, m$ are real. The subhypotheses RH_l can be stated through the limits of the geometric means of the eigenvalues of $M_{l,m}$. The positions of the eigenvalues $\mu_{l,m,n}$ have been analysed in [8, 9], and several conjectures have been formulated.

DOI: 10.1002/nla

Two things prompted the current work. First, it would be interesting to perform larger scale analysis of the spectra of $M_{l,m}$ in order to collect numerical evidence in favour of some conjectures in [8], or to the contrary, disproving the conjectures about asymptotics of $\mu_{l,m,n}, m, n \to \infty$. Second, it would be interesting to study analogous matrices and spectra for Dirichlet L-functions $L(s, \chi)$. Of course, Riemann zeta function is the first member of the family of Dirichlet L-functions. As Dirichlet L-functions share many properties, a valid question is whether properties of the spectra of $M_{l,m}$ will be shared or not.

This motivating exposition brings us to the need to compute the eigenvalues of real and complex Hankel matrices with high accuracy multiple times. The high accuracy is essential to distinguish the true behaviour of the eigenvalues from numerical artefacts. A recent study [20] involving computation of matrix determinants in multiprecision arithmetics revealed that some theoretically meaningful properties of such determinants could be observed only at an extremely high precision level, and therefore, it was essential to perform large-scale computations with high working precision. This in turn required rather complex modifications, adaptations and parallelisation of classical linear algebra algorithms [21] as off-the-shelf solutions were either not suitable or not scalable.

Note that the matrices involved in this study could be quite large, and finite precision cancellation errors could be significant. In addition, some of the conjectures formulated in [8, 9] state that there are no multiple eigenvalues; hence, we may need to distinguish numerically between closely located eigenvalues.

On the other hand, we are interested in the whole sequences of spectra of matrices $M_{l,m}, l, m = 1, 2, \ldots$, and also of matrices based on other Dirichlet L-functions. That will involve calculations of eigenvalues in bulk, and consequently, we are interested in developing a very efficient numerically stable algorithm, in multiprecision arithmetics.

We formally state the requirements.

Problem. Our aim is to develop a numerically efficient algorithm in multiprecision arithmetics to calculate the spectra of real and complex Hankel matrices.

The current sizes of the matrices we are interested in are up to $m = 5000$, and the initial numerical accuracy is of the order of 10 000 decimal places. Larger matrices do not fit the 132-Gb RAM of workstations available to us at the moment and hence require the development of parallel distributed memory algorithms that are beyond the scope of this study.

2.2. *Lanczos method*

From now on, n will denote the size of the matrix. A Hankel matrix has the following form

$$A_H = \begin{bmatrix} a_1 & a_2 & \ldots & a_{n-1} & a_n \\ a_2 & a_3 & \ldots & a_n & a_{n+1} \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ a_{n-1} & a_n & \ldots & a_{2n-3} & a_{2n-2} \\ a_n & a_{n+1} & \ldots & a_{2n-2} & a_{2n-1} \end{bmatrix}. \tag{7}$$

For matrix size n, it can be represented through an array of elements of size $2n - 1$, which we will also denote by A. A Toeplitz matrix has a different form

$$A_T = \begin{bmatrix} a_n & a_{n+1} & \ldots & a_{2n-2} & a_{2n-1} \\ a_{n-1} & a_n & \ldots & a_{2n-3} & a_{2n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_2 & a_3 & \ldots & a_n & a_{n+1} \\ a_1 & a_2 & \ldots & a_{n-1} & a_n \end{bmatrix} \tag{8}$$

and can be converted to a Hankel matrix by reversing the order of rows or columns, that is, by multiplying it by a permutation matrix. Both matrices can be embedded into circulant $2n \times 2n$ matrices.

DOI: 10.1002/nla

The Lanczos algorithm is formulated as Algorithm 1. It computes a symmetric tridiagonal matrix J similar to a Hermitian or a complex-symmetric matrix A. Matrix J is represented by its two co-diagonals $\alpha \in \mathbb{C}^n, \beta \in \mathbb{C}^{n-2}$. Once the matrix is tridiagonalised, an efficient QR iteration algorithm with implicit Wilkinson shift [3, 22] is applied to find the eigenvalues (and, optionally, the eigenvectors). Alternatively, Gu and Eisenstat algorithm [23] (see also [22], chapter 4) can also be used for more efficient computations.

Algorithm 1: Lanczos tridiagonalisation.

Input: n, $A \in \mathbb{C}^{n\times n}$ Hermitian or complex-symmetric.
Output: $J \in \mathbb{C}^{n\times n}$ symmetric tridiagonal.
The co-diagonals of J are $\alpha, \beta \in \mathbb{C}^n$, $\alpha_i = J_{ii}$, $\beta_i = J_{i-1,i} = J_{i,i-1}$.
1 Initialise $v_1 \in \mathbb{C}^n$ such that $||v_1|| = 1$;
$v_0 = 0$;
$\beta_1 = 0$;
2 For $i = 1, \ldots, n-1$ do:

2.1 $w_i = Av_i$;
2.2 $\alpha_i = v_i^T w_i$;
2.3 $w_i = w_i - \alpha_i v_i - \beta_i v_{i-1}$;
2.4 $\beta_{i+1} = ||w_i||$;
2.5 $v_{i+1} = \frac{1}{\beta_{i+1}} w_i$;

3 $w_n = Av_n$;
$\alpha_n = v_n^T w_n$
3 Return J (in the form of α, β).

The most time-consuming step in Lanczos algorithm is the matrix–vector multiplication in step 2.1. For general matrices, the complexity of Lanczos algorithm is $O(n^3)$. However, fast multiplication of a circulant matrix by a vector can be achieved using FFT in $O(n \log n)$ operations [2]. In this paper, the logarithm always refers to base 2, $\log_2$. Consequently, multiplication by a Hankel (or Toeplitz) matrix can also be performed in $O(n \log n)$ time using FFT. In this way, the complexity of Lanczos method becomes $O(n^2 \log n)$.

Note that direct methods for calculating eigenvalues are usually based on first reducing the matrix to the upper Hessenberg form followed by QR iteration [24]. The complexity of such methods is $O(n^3)$, and the special structure of Hankel matrices is not used.

The schoolbook multiplication of a Hankel matrix by a vector $y = A_H x$ is computed as follows:

$$y_i = \sum_{i=1}^{n} a_{i+j-1} x_i.$$

The FFT-based multiplication can be achieved by using direct (FFT) and inverse ($IFFT$) Fourier transforms

$$\hat{y} = IFFT(FFT(\hat{a}) * FFT(\hat{c})),$$

where the auxiliary vectors are

$$\hat{a} = (a_n, a_{n+1}, \ldots, a_{2n-1}, a_1, \ldots a_{n-1})^T,$$

$$\hat{x} = (x_n, x_{n-1}, \ldots, x_1, 0, \ldots, 0)^T$$

and

$$\hat{y} = (y_1, y_2, \ldots, y_n, \ldots)^T,$$

DOI: 10.1002/nla

so that the desired product is given by the first n components of $\hat{y}$. The operation $*$ is the componentwise multiplication of two vectors. The operation count is $30n \log n + O(n)$ additions and multiplications [6, 7]. The authors of [6, 7] claimed that the FFT algorithm becomes superior to the schoolbook multiplication for $n > 16$ when using standard (hardware-based) data types. However, for multiprecision data types, this estimation does not hold. Our computations in multiprecision arithmetics show that for $b = 32\,768$ bits accuracy, FFT becomes superior only for $n > 1000$.

One approach to fast Hankel matrix multiplication is to change the basic type in an FFT library from float or double to multiprecision. This approach was taken in the Matlab multiprecision computing toolbox [14]. However, there are also some alternatives that we wanted to explore. Because the fast multiprecision multiplication is itself based on FFT, it means that we need to perform one FFT as a basic step inside another. It might be possible to unroll these operations into just one standard FFT, where we can use several optimised parallel FFT libraries (in single or double precision). Changing the basic scalar type in these highly optimised libraries is not simple, as the length of the scalar type may drastically worsen the performance.

3. DECOMPOSITION APPROACH

Bearing in mind that ideally we would want to make use of the existing optimised FFT libraries, we attempted to decompose each multiprecision number into a sum of float or double precision numbers, then applied a standard FFT library (including optimised parallel libraries) and eventually reconstructed multiprecision vectors from the arrays of floats. We call this approach decomposition. We now present the details of the decomposition approach to implementation of FFT in multiprecision arithmetics. We restrict our presentation to real-valued data, although the same approach is valid for complex-valued data with relevant modifications.

Every multiprecision number is represented as a sum $a = B_0 \sum_{k=1}^{l} B^{k-1} a_k$, where B is the base used, typically $B = 2^{32}$ or $B = 2^{64}$, and B_0 is a factor. The number l is derived from the number of bits in the mantissa b as $l = \lceil b / \log B \rceil$.

Let us show that we can write the product $y = A_H x$, in which every element of A_H and x is a sum $a_i = \sum_k a_{ik}$, $x_i = \sum_k x_{ik}$, as a product of an auxiliary larger Hankel matrix $\hat{A}$ by a vector $\hat{x}$, whose components are the terms of the sums. Consider the matrix

$$\hat{A} = \begin{bmatrix} a_{11} & \dots & a_{1l} & a_{11} & \dots & a_{1l} & a_{21} & \dots & a_{n,l-1} & a_{nl} \\ a_{12} & \dots & a_{11} & a_{12} & \dots & a_{21} & a_{22} & \dots & a_{n,l} & a_{n+1,1} \\ \vdots & \vdots & \vdots & & \iddots & & \vdots & \vdots & \vdots & \vdots \\ a_{nl} & \dots & \dots & \dots & \dots & \dots & \dots & \dots & a_{2n-1,l-1} & a_{2n-1,l} \end{bmatrix}. \tag{9}$$

This matrix is a Hankel matrix determined by the vector $(a_{11}, \dots, a_{1l}, a_{11}, \dots, a_{1l}, \dots, a_{2n-1,l})$, and it has size $m = 2nl$. Notice that the components of the sum for each element of a_i appear twice. Let also

$$\hat{x} = (x_{11}, x_{12}, \dots, x_{1l}, 0, \dots, 0, x_{21}, \dots, x_{2l}, 0, \dots, 0, x_{31}, \dots 0)^T.$$

We can now compute the product $\hat{y} = \hat{A}\hat{x}$ by using FFT. Let us examine the element $\hat{y}_1$. It will be

$$\hat{y}_1 = a_{11}x_{11} + a_{12}x_{12} + \dots + a_{1l}x_{1l} + \dots = \sum_{k=1}^{l} \sum_{i=1}^{n} a_{ik} x_{ik}.$$

Next,

$$\hat{y}_2 = \sum_{k=1}^{l} \sum_{i=1}^{n} a_{i,(k+1) mod\, l} x_{ik}, \ \hat{y}_3 = \sum_{k=1}^{l} \sum_{i=1}^{n} a_{i,(k+2) mod\, l} x_{ik}, \ \dots$$

$$\hat{y}_l = a_{1l}x_{11} + a_{11}x_{12} + \dots + a_{1,l-1}x_{1l} + \dots = \sum_{k=1}^{l} \sum_{i=1}^{n} a_{i,(k+l-1) mod\, l} x_{ik}.$$

DOI: 10.1002/nla

Next, we add the elements $\hat{y}_1 + \hat{y}_2 + \cdots + \hat{y}_l$ and rearrange the sum to obtain

$$\hat{y}_1 + \cdots + \hat{y}_l = a_{11} \sum_{k=1}^{l} x_{1k} + a_{12} \sum_{k=1}^{l} x_{1k} + \cdots + a_{1l} \sum_{k=1}^{l} x_{1k} + a_{21} \sum_{k=1}^{l} x_{1k} + \ldots$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{l} \left(a_{ik} \sum_{j=1}^{l} x_{ij} \right) = \sum_{i=1}^{n} \left(\sum_{k=1}^{l} a_{ik} \right) \left(\sum_{j=1}^{l} x_{ij} \right) = \sum_{i=1}^{n} a_i x_i = y_1.$$

Thus, we obtained the first component of the desired solution y. When we proceed this way with the rest of the l-length segments of vector $\hat{y}$, we acquire the resulting formula

$$y_i = \sum_{k=l(i-1)+1}^{il} \hat{y}_k. \tag{10}$$

Consequently, we obtained the desired product $y = A_H x$ by decomposing each entry a_i and x_i into a sum, and then by multiplying the auxiliary matrix and vector $\hat{y} = \hat{A}\hat{x}$ constructed from the individual terms of the sums, and recovering y using partial sums of the elements of $\hat{y}$. Now, because every multiprecision number was represented as a sum of terms with standard accuracy, we can compute the product $A_H x$ by decomposing the entries into standard precision numbers, performing the product with standard precision and then recovering the multiprecision result.

To analyse the complexity of this process, recall that the product $\hat{A}\hat{x}$ is performed using FFT, so the complexity will be $O(nl \log(nl)) = O(nb \log(nb))$, depending on the number of bits b of multiprecision numbers. This complexity is smaller than complexity of performing FFT in multiprecision arithmetics $O(nb \log n \log b))$, and the fact that we unrolled the two nested FFTs (we remind that multiplication in multiprecision arithmetics is also performed by FFT for large b) into one FFT that uses standard data type may help to speed up the algorithm, because the existing FFT libraries are highly optimised. The wide availability of a number of such libraries on different platforms is a benefit.

On the other hand, a potential caveat is that losses of accuracy when performing computations with the standard accuracy could happen. The reason is that the terms of the sums representing mantissas of multiprecision numbers are very different in magnitude, and thus, the standard FFT may lose accuracy during summations, even if the sum (10) is performed using higher accuracy. In Section 5, we will describe the results of our numerical experiments, where we show that this loss of accuracy takes place, and that the resulting vector y is obtained with less than 92 bits accuracy.

4. RECURSIVE ECONOMIC MULTIPLICATION

In this section, we present an alternative way of computing the product $y = A_H x$ presented by Beliakov in [16]. This method was inspired by Karatsuba multiplication of scalars [25]. We shall now briefly present the method from [16] for completeness. Consider two rows of a Hankel matrix multiplied by vector x

$$\begin{aligned} &\ldots + a_i x_i + a_{i+1} x_{i+1} + \ldots \\ &\ldots + a_i x_{i-1} + a_{i+1} x_i + \ldots \end{aligned} \tag{11}$$

Apparently, we need four multiplications and two additions. However, consider a more economical scheme. Let

$$C = (a_i + a_{i+1}) x_i, \; D = a_{i+1}(x_i - x_{i+1}), \; E = a_i (x_{i-1} - x_i).$$

DOI: 10.1002/nla

Then, the first row in (11) can be written as $C - D$ and the second row as $C + E$. Hence, we compute the same quantities using three multiplications and five additions. If we now perform these operations in every column of A and every pair of rows, we achieve a saving of one quarter of multiplications; hence, the operations count will be $\frac{3}{4}n^2$ multiplications, at the expense of a few (cheaper) additions. Furthermore, the additions can be amortised, as exactly the same differences between x_i and x_{i+1} appear in the subsequent rows. We will perform a detailed operations count after we present the whole algorithm based on this observation.

We create three auxiliary vectors (we remind that a Hankel matrix is represented by a $(2n-1)$ –vector). Let $m = \lfloor \frac{n+1}{2} \rfloor$ and $m_1 = \lceil \frac{n+1}{2} \rceil$. Let also

$$C : c_i = a_{2i-1} + a_{2i}, i = 1, \ldots, m$$

be a Hankel matrix containing the pairwise sums of neighbouring elements of A,

$$D : d_i = a_{2i}, i = 1, \ldots, m \text{ and } E : e_i = a_{2i-1}, i = 1, \ldots, m_1$$

be Hankel matrices containing the even and odd elements of (the vector representation of) A, respectively, and vectors

$$f : f_i = x_{2i-1} - x_{2i}, \;\; i = 1, \ldots, m, \;\; g : g_i = x_{2i-2} - x_{2i-1}, \; i = 1, \ldots, m_1$$

contain pairwise (even and odd) differences of the elements of x, with the convention $x_0 = x_{n+1} = 0$ and also $a_{2n} = a_{2n+1} = 0$, and vector

$$h : h_i = x_{2i-1}, i = 1, \ldots, m.$$

The Hankel matrices C and D will be of size $m \times m$, and E will be of size $m_1 \times m_1$ (for an odd n, we have $m = m_1$). Let us now compute the quantities $p = Ch$, $q = Df$ and $r = Eg$. Now every odd-numbered element of y can be written as $y_{2i-1} = p_i - q_i$ and every even numbered element $y_{2i} = p_i + r_i, i = 1, \ldots, m$.

To verify this, consider

$$\begin{aligned}
p_i &= \sum_{j=1}^{m} c_{i+j-1} h_j = \sum_{j=1}^{m} (a_{2i+2j-3} + a_{2i+2j-2}) x_{2j-1} \\
&= \sum_{j=1}^{m} a_{2(i-1)+2j-1} x_{2j-1} + \sum_{j=1}^{m} a_{2(i-1)+2j} x_{2j-1}, \\
q_i &= \sum_{j=1}^{m} d_{i+j-1} f_j = \sum_{j=1}^{m} a_{2i-1+2j} x_{2j-1} - \sum_{j=1}^{m} a_{2i-1+2j} x_{2j}, \\
r_i &= \sum_{j=1}^{m_1} e_{i+j-1} g_j = \sum_{j=1}^{m_1} a_{2(i-1)+2j-1} x_{2j-2} - \sum_{j=1}^{m_1} a_{2(i-1)+2j-1} x_{2j-1}.
\end{aligned}$$

We see that $p_i - q_i$ collapses into

$$y_{2i-1} = \sum_{j=1}^{m} a_{2(i-1)+2j-1} x_{2j-1} + \sum_{j=1}^{m} a_{2(i-1)+2j-1} x_{2j} = \sum_{k=1}^{n} a_{2i-1+k-1} x_k,$$

and $p_i + r_i$ collapses into

$$y_{2i} = \sum_{j=1}^{m} a_{2(i-1)+2j} x_{2j-1} + \sum_{j=1}^{m_1} a_{2(i-1)+2j-1} x_{2j-2} = \sum_{k=1}^{n} a_{2i+k-1} x_k,$$

taking into account $x_0 = 0$ and $a_{2n} = 0$.

DOI: 10.1002/nla

Let us now count the number of operations. Vectors p and q require $m \times m$ multiplications and vector r requires $m_1 \times m_1$ multiplications, which is at most m_1^2. These vectors also require at most m_1^2 additions.

Construction of matrix C requires m additions and vectors f and g need m and m_1 additions, respectively. So, overall, we need at most $3m_1^2$ multiplications and $3m_1^2 + 3m_1 + n$ additions, which gives us at most $\frac{3}{4}(n+1)^2 + O(n)$ multiplications and additions, that is, one quarter improvement over the schoolbook matrix–vector multiplication (and only $O(n)$ cost of extra additions, which, as we see, are amortised).

Next, let us apply this algorithm recursively. We see that the products $p = Ch$, $q = Df$ and $r = Eg$ involve Hankel matrices of half the size of the original. The recursive algorithm is presented as Algorithm 2.

Algorithm 2: Calculation of the product of a Hankel matrix by a vector.

Input: n, $a \in \mathbb{C}^{2n-1}$, $x \in \mathbb{C}^n$.
Output: $y \in \mathbb{C}^n$.

1 If $n = 2$ /* explicit formula */
  $y_1 = a_1x_1 + a_2x_2$; $y_2 = a_2x_1 + a_3x_2$;
  return y
2 /* Prepare auxiliary vectors */
  $m = floor((n+1)/2)$, $m_1 = ceil((n+1)/2)$
3 For $i = 1, \ldots, m$ do: $f_i = x_{2i-1} - x_{2i}$, $h_i = x_{2i-1}$. /* take $x_{n+1} = 0$ */
4 For $i = 1, \ldots, 2m-1$ do: $c_i = a_{2i-1} + a_{2i}$, $d_i = a_{2i}$. /* take $a_{2n} = 0$ */
5 For $i = 1, \ldots, m_1$ do: $g_i = x_{2i-2} - x_{2i-1}$ /* take $x_0 = x_{n+1} = 0$ */
6 For $i = 1, \ldots, 2m_1 - 1$ do: $e_i = a_{2i-1}$ /* take $a_{2n+1} = 0$ */
7 /* call Algorithm 2 recursively */
  7.1 call Algorithm2(m, c, h, p)
  7.2 call Algorithm2(m, d, f, q)
  7.3 call Algorithm2($m1, e, g, r$)
8 For $i = 1, \ldots, m$ do:
  8.1 $y_{2i-1} = p_i - q_i$
  8.2 if $2i \leqslant n$ then $y_{2i} = p_i + r_i$
9 return y.

Note that an efficient implementation of Algorithm 2 should avoid explicit assignment operations for h_i, d_i and e_i (which bear non-negligible cost in multiprecision arithmetics) and instead use references to the addresses of the variables (pointers).

The analysis of complexity of the recursive algorithm is performed in a standard way using the recurrence relation $T(n) = 3T(\frac{n}{2}) + f(n)$, where $T(n)$ is the cost of Algorithm 2 for input of size n, 3 is the number of invocations of this algorithm at step 7, and $f(n)$ is the cost of preparation of the auxiliary vectors at steps 3–6, which is $\Theta(n)$. Then, applying the master theorem [26], the cost of the recursive algorithm $T(n) = \Theta(n^{\log 3}) \approx \Theta(n^{1.585})$, which is the same complexity as Karatsuba algorithm and the method reported in [27].

The direct operations count was also performed in [16] when $n = 2^k$, in which case the depth of recursion is at most $k = \log n$. This count revealed that recursive Algorithm 2 uses at most $3 \times 3^k = 3n^{\log 3}$ multiplications and $4n^{\log 3} + O(n)$ additions. It follows that Algorithm 2 from [16] will be computationally more efficient than the schoolbook matrix–vector multiplication for every $n > 3$ regardless of the cost of multiprecision addition and multiplication.

Furthermore, the nature of Algorithm 2 offers opportunities for parallelisation. Indeed, the invocations of Algorithm 2 can be carried out by separate threads and on separate host computers. For example, a thread executing Algorithm 2 can spawn two other threads, so that the three recursive

DOI: 10.1002/nla

calls at step 7 are executed in parallel. All threads will have the same amount of work and no load balancing issues arise. This parallelisation strategy is particularly useful on shared memory architectures, where no data movement between hosts is required. Spawning different threads at step 7 is performed until all the available cores are used, although in practice twice as many threads as physical cores can be used to take advantage of hyper-threading.

5. NUMERICAL EXPERIMENTS

In the first set of experiments, we compare the performance of four methods for multiprecision matrix–vector multiplication using Hankel matrices with randomly and systematically generated entries of different accuracies. We record the average CPU time as a function of matrix size and precision of the result. The four methods compared are as follows: (1) the decomposition approach; (2) implementation of FFT in multiprecision arithmetics; (3) recursive economic matrix multiplication from [16]; and (4) the direct schoolbook matrix–vector multiplication as the baseline. We also verify the accuracy of the end result by comparing it with the values calculated by the baseline direct multiplication with a four times larger accuracy.

Computations were performed on an Intel ×386 64-bit workstation with 8 Gb of RAM running Linux. All the methods were implemented in C++ language compiled with g++. GMP, MPFR and MPC libraries [10, 11] were used for multiprecision arithmetics and `kissfft` [15] with multiprecision arithmetics was used for FFT.

Our experiments are summarised in Tables I and II. These tables reveal that unfortunately the decomposition approach delivers the result with less than the expected number of correct mantissa bits. We used a 128-bit type `float128` with 112 mantissa bits and 15 bits exponent as the base data type in the FFT algorithm (in `kissfft` library), but the resulting product had at most 92 correct mantissa bits regardless the precision of the inputs and that output accuracy decreased with the size of the matrix n. In spite of its better asymptotic computational complexity, the significant overheads related to decomposing and copying data resulted in excessive CPU time. The remaining methods maintained full or almost full accuracy as expected. Hence, the decomposition method is not suitable for our purposes.

The CPU timings, which are almost identical in Tables I and II, point out to the superiority of the FFT-based multiprecision multiplication over the direct multiplication only for large n. In particular, for $b = 32768$ mantissa bits, n must be above 1000 to take advantage of the asymptotically more efficient FFT algorithm. In fact, the baseline direct multiplication is much faster than FFT for $n \leqslant 512$ at that precision. Taking into account very simple parallelisation strategies that can be applied (this method is *embarrassingly parallel*), the advantages of direct multiplication are even greater.

On the other hand, as expected from the operations count, the recursive economic multiplication from [16] is more efficient than the baseline for almost all values of n and b. It is also superior to FFT-based multiplication for large b and moderate $n < 8192$, although for smaller b, FFT is the winner. Tables I and II allow one to select the most efficient approach based on the values of n and b. For moderate matrix sizes and high accuracy, the recursive multiplication is the method of choice, while FFT performs better with larger matrices and not as high precision. Parallelisation of the recursive method from [16] can also be performed in a relatively straightforward way.

The question of losses of accuracy is quite important. The losses of accuracy were evaluated using two scenarios: randomly chosen components of A and x and the precomputed values of θ_i, the coefficients of the Taylor series of $\bar{\xi}$ presented in Section 2, which correspond to our motivating application. Neither case is the worst case scenario, as obviously one can design matrix entries that would lead to much more drastic losses of accuracy. Our aim here is to illustrate the behaviour of the methods we studied in a realistic setting and appreciate the accuracy losses of each method relative to the others.

When the components of the matrix were generated randomly (in Table I), the three methods we focus on suffered relatively minor and comparable losses of accuracy. However, when we used systematically generated matrices based on (6), the behaviour of the algorithms changed.

DOI: 10.1002/nla

Table I. The mean execution time (seconds) and the accuracy retained (bits) of four methods of Hankel matrix–vector multiplication using randomly generated matrix entries. The smallest time is in boldface.

Matrix size	Length of mantissa	Schoolbook		FFT		Recursive		Decomposition	
		bits	CPU	bits	CPU	bits	CPU	bits	CPU
128	128	119	**0.0017**	114	0.0060	114	0.0030	92	0.0052
128	512	503	**0.0031**	503	0.012	502	0.0036	92	0.18
128	2048	2040	0.016	2039	0.051	2036	**0.0095**	92	0.81
128	8192	8184	0.12	8181	0.49	8181	**0.051**	92	15
128	32 768	32 760	0.94	32 759	5.2	32 759	**0.37**	92	388
512	128	117	0.030	116	**0.024**	114	0.028	66	0.22
512	512	501	0.050	501	0.044	497	**0.033**	66	0.80
512	2048	2036	0.26	2037	0.18	2034	**0.091**	66	3.4
512	8192	8179	1.9	8179	1.65	8178	**0.47**	67	62
512	32 768	32 756	15.1	32 756	21	32 755	**3.5**	67	1546
1024	128	115	0.12	116	**0.05**	112	0.08	43	0.45
1024	512	495	0.19	497	**0.09**	495	0.1	44	1.7
1024	2048	2035	1.0	2035	0.35	2032	**0.31**	41	7.2
1024	8192	8178	7.6	8177	3.2	8176	**1.5**	44	126
1024	32 768	32 755	60	32 755	42	32 753	**11**	41	3102
2048	128	113	0.55	115	**0.10**	112	0.25	4	0.95
2048	512	498	0.77	498	**0.18**	495	0.30	4	3.4
2048	2048	2033	4.2	2034	**0.71**	2030	0.93	4	14
2048	8192	8178	30	8178	6.4	8176	**4.4**	4	254
2048	32 768	32 754	241	32 755	85	32 752	**32**	4	6202
8192	128	108	11	113	**0.42**	107	2.4	0	4.1
8192	512	494	12	495	**0.76**	494	2.8	0	14
8192	2048	2031	68	2033	**2.9**	2027	8.5	0	63
8192	8192	8175	488	8176	**26**	8173	40	0	1033
8192	32 768	32 750	3860	32 750	343	32 748	**288**	0	25 081
16 384	128	107	51	111	**0.89**	105	7	0	8.5
16 384	512	494	50	495	**1.6**	491	8	0	30
16 384	2048	2029	273	2033	**6**	2026	25	0	130
16 384	8192	8173	1953	8174	**52**	8170	122	0	2094
16 384	32 768	32 748	15 465	32 752	**697**	32 746	867	0	50 811

FFT, fast Fourier transform.

Table II reveals that both FFT and the recursive method suffer greater accuracy losses than the direct schoolbook multiplication. The losses in FFT and the recursive method are roughly the same and are significant enough to make these algorithms struggle when the working precision is at the lower end of the spectrum. To some degree, the inferior accuracy compared with the schoolbook multiplication can be compensated in both FFT and recursive method by slightly increasing the working precision (by about 200 bits), which does not drastically affect CPU time. However, when these algorithms are used for matrix–vector product repeatedly within another algorithm, such as Lanczos, the accumulations in the round-off errors could be significant, as illustrated later.

The second set of experiments involves running the Lanczos tridiagonalisation method (and the subsequent QR iterations) using the multiplication methods we studied. The accuracy was evaluated by using the gold standard – the eigenvalues computed with four times the accuracy – and also by computing the determinant of the matrix using the product of its eigenvalues and by using the extended continuant of the tridiagonal matrix. In case of $M_{0,m}$ in (6), the determinant is 1, which also serves as the gold standard to test the accuracy. The Lanczos method was compared with a direct method of calculating eigenvalues implemented in Matlab in multiprecision arithmetics using the ARPACK library and multiprecision computing toolbox [14]. These computations were performed on Intel Xeon 64 bit workstations with 64 GB of RAM running Linux.

DOI: 10.1002/nla

Table II. The mean execution time (seconds) and the accuracy retained (bits) of four methods of Hankel matrix–vector multiplication using systematically generated matrix entries. The smallest time is in boldface.

Matrix	Length of	Schoolbook		FFT		Recursive		Decomposition	
size	mantissa	bits	CPU	bits	CPU	bits	CPU	bits	CPU
128	128	124	**0.0017**	108	0.0058	106	0.0030	92	0.0052
128	512	508	**0.0031**	492	0.011	490	0.0036	92	0.18
128	2048	2044	0.016	2028	0.050	2027	**0.0095**	92	0.81
128	8192	8188	0.12	8171	0.49	8171	**0.051**	92	15
128	32 768	32 763	0.94	32 474	5.2	32 746	**0.37**	92	388
512	128	122	0.030	82	**0.024**	79	0.028	66	0.22
512	512	506	0.049	466	0.043	463	**0.033**	66	0.80
512	2048	2043	0.26	2003	0.17	1999	**0.091**	66	3.4
512	8192	8187	1.9	8146	1.64	8142	**0.47**	67	62
512	32 768	32 763	15.1	32 723	21	32 718	**3.5**	67	1546
1024	128	117	0.12	59	**0.05**	51	0.08	43	0.45
1024	512	503	0.19	440	**0.09**	436	0.1	44	1.7
1024	2048	2038	1.0	1977	0.35	1972	**0.3**	41	7.2
1024	8192	8182	7.7	8123	3.2	8116	**1.5**	44	126
1024	32 768	32 759	61	32 698	42	32 692	**10**	41	3102
2048	128	120	0.51	20	**0.10**	16	0.25	4	0.95
2048	512	505	0.75	402	**0.18**	400	0.30	4	3.4
2048	2048	2040	4.3	1940	**0.71**	1933	0.93	4	14
2048	8192	8185	31	8082	6.4	8077	**4.4**	4	254
2048	32 768	32 760	241	32 661	85	32 654	**32**	4	6202
8192	128	115	11	0	**0.42**	0	2.3	0	4.1
8192	512	498	12	287	**0.75**	336	2.9	0	14
8192	2048	2036	68	1823	**2.9**	1872	8.5	0	63
8192	8192	8177	490	7967	**26**	8017	40	0	1033
8192	32 768	32 755	3857	32 543	343	32 593	**287**	0	25 081
16 384	128	114	46	0	**0.88**	0	7	0	8.5
16 384	512	497	49	341	**1.6**	355	8	0	30
16 384	2048	2035	271	1879	**6**	1893	25	0	130
16 384	8192	8176	1956	8023	**52**	8036	121	0	2094
16 384	32 768	32 754	15 451	32 598	**696**	32 611	865	0	50 811

FFT, fast Fourier transform.

Our computations revealed that QR iterations affected the losses of accuracy only marginally (e.g. the product of eigenvalues and the extended continuant differed by at most four decimal places in all experiments), and hence, most of the accuracy losses are attributed to the Lanczos algorithm.

In Table III, we present the average CPU time and the number of correct mantissa bits when running Lanczos and direct algorithms with matrices $M_{l,n}$, $l = 0, 1, \ldots, 10$ and given values of the matrix size n (the missing entries were not computed because of excessive CPU time). We observe that the round-off errors accumulate, and even the working precision of $b = 32\,768$ mantissa bits is hardly sufficient to find eigenvalues of a matrix 512×512. This situation worsens when finding eigenvalues of complex valued matrices (when using Taylor series coefficients for various Dirichlet L-functions rather than Riemann zeta function as θ_i in (6)), on which we report only briefly here: even the working precision of 30 000 decimal places (over 10^5 mantissa bits) was not sufficient to obtain the eigenvalues with a reasonable accuracy for $n = 1024$.

The numerical instability of the Lanczos method is known, and various approaches to improve its behaviour, such as full or partial re-orthogonalisation, were suggested [28, 29]. In the context of our studies, we implemented partial re-orthogonalisation from [28], which drastically improved the accuracy of eigenvalues computation, but at a price of a significant increase of RAM requirements and extra computations. The computational cost of full re-orthogonalisation is $O(n^3)$; however, partial re-orthogonalisation costs less than 20% of that as reported in [28]. Furthermore, re-orthogonalisation is embarrassingly parallel, and full advantage of this fact was made on a multi-

DOI: 10.1002/nla

Table III. The mean execution time (seconds) and the accuracy retained (bits) for Lanczos tridiagonalisation algorithm based on three methods of Hankel matrix–vector multiplication and the execution time of the subsequent QR iteration algorithm applied to the tridiagonal matrix. The smallest time is in boldface.

Matrix size	Length of mantissa	Schoolbook		FFT		Recursive		QR iterations	Direct method
		bits	CPU	bits	CPU	bits	CPU	CPU	
128	8192	8211	9	8218	46	8215	**5**	7	18
128	16 384	16 199	30	16 199	151	16 197	**10**	21	52
128	32 768	32 672	81	32 664	485	32 667	**26**	60	156
128	65 536	65 448	223	65 440	1202	65 444	**73**	226	491
256	8192	1200	74	3246	147	3232	**26**	23	109
256	16 384	16 199	206	16 199	504	16 197	**60**	83	336
256	32 768	32 596	560	32 586	1520	32 588	**205**	247	942
256	65 536	65 363	1621	—	—	65 362	**388**	883	46 318
512	8192	0	636	0	760	0	**120**	97	818
512	16 384	0	1781	0	1830	0	**320**	302	13 465
512	32 768	12 570	4530	12 566	5120	12 570	**842**	1081	—
512	65 536	—	—	—	—	65 189	**2451**	3168	—
768	32 768	—	—	0	11 826	0	**2820**	2223	—
768	65 536	—	—	—	—	24 944	**7680**	6748	—
1024	131 072	—	—	—	—	94 395	**42 804**	37 579	—

The CPU time for a multiprecision direct method is shown for comparison.
FFT, fast Fourier transform.

core architecture. With $b = 16\,384$ bits working precision, we were able to compute the eigenvalues of real and complex Hankel matrices to at least 6000 decimal places for $n \leqslant 8000$ by using both FFT and recursive multiplication. The latter method was numerically superior to the FFT for n up to 6000. When we parallelised the recursive multiplication for multicore shared memory architecture, which unlike FFT was very straightforward, the difference in numerical performance increased sixfold, making that method a clear winner.

When compared with the direct method (the last column in Table III), its cost was significantly higher even for relatively small matrices when the working precision was above $b = 16\,384$ bits. As expected, its computational cost rose as the cube of the matrix size. Furthermore, even though the Matlab implementation of this method did take advantage of multicore parallelisation for small working precision (up to $b = 256$ on our system), it was not the case for higher working precision we are interested in.

Our numerical results prompted the following practical considerations. Large matrix sizes require very large working precision when performing tridiagonalisation by Lanczos method. In particular, for n in the range 500–1000, the accuracy of $b = 65\,536$ and over seems to be warranted (unless re-orthogonalisaion methods are employed). For all combinations of n and b reported here, the most efficient matrix–vector product method is not the FFT but the recursive method from [16], with the schoolbook multiplication being the distant second. While the FFT (with a fixed working precision) performs better for larger matrices, Lanczos method for larger matrices requires an even higher accuracy, which in turn makes FFT less efficient than the recursive method, and even the direct multiplication. This endless loop suggests that it is the recursive method rather than FFT that is the most practically efficient multiplication method to be used in Lanczos algorithm for the stated combinations of matrix sizes and working precision, at least in the context of research on Riemann's hypothesis presented in [8, 9].

Another consideration is that the direct eigenvalue algorithm was unable to take advantage of the special structure of Hankel matrices and was not competitive in terms of the CPU time. Lanczos method for Hankel matrices compares favourably in terms of its computational complexity, and even if partial re-orthogonalisation is employed to improve the accuracy, its overall CPU time is still significantly lower than that of a direct method.

DOI: 10.1002/nla

6. CONCLUSIONS

We have presented and experimentally compared several alternative approaches to Lanczos tridiagonalisation of Hankel matrices in multiprecision arithmetics. The key step is matrix–vector multiplication, which needs to be accurate and efficient. The FFT-based multiplication, traditionally used as a fast matrix–vector multiplication method with Hankel and Toeplitz matrices with $b = 53$ bits precision, is numerically efficient only for quite large matrices when the number of mantissa bits b is large. On the other hand, direct schoolbook matrix–vector multiplication offers practical advantages of simplicity and trivial parallelisation.

One of our approaches based on decomposing multiprecision numbers into sums, and converting multiprecision multiplication to standard precision FFT-based matrix multiplication, was not successful, as the accuracy of the end result was lost, and CPU time was also excessive because of overheads. However, another method of recursive economical Karatsuba-like matrix multiplication significantly outperformed all other methods we studied in the range of values for n and b we were interested in. This method has complexity $\Theta(n^{\log_2 3})$ and is relatively easy to parallelise. Therefore, we recommend this approach for multiplications with Hankel and Toeplitz matrices when using high accuracy data types.

Numerical experiments with Lanczos method revealed that for matrix sizes in the range 256–1024, a sufficiently high working precision of 5000 to 40 000 decimal places was needed. The recursive economical multiplication method significantly outperformed the alternatives including FFT while delivering the same accuracy of the output. When combined with partial reorthogonalisation and multicore shared memory parallelisation, this method turned out to be sixfold to 30-fold more efficient than FFT (for $n \leqslant 8000$ and $b = 16\,384$) and allowed us to compute highly accurate eigenvalues of real and complex Hankel matrices that appeared in [8, 9]. Our method also outperformed a competitive direct method by a large margin.

The developed multiprecision Lanczos method was successfully applied to the motivating eigenvalues problem, and accurate results were obtained efficiently. Some recent results are available from [30].

REFERENCES

1. Golub G, Van Loan C. *Matrix Computations*. The John Hopkins University Press: Baltimore, 1996.
2. Bai Z, Demmel J, Dongarra J, Ruhe A, van der Vorst H. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM: Philadelphia, 2000.
3. Saad Y. *Numerical Methods for Large Eigenvalue Problems*. SIAM: Philadelphia, 2011.
4. Heinig G, Rost K. Fast algorithms for Toeplitz and Hankel matrices. *Linear Algebra and its Applications* 2011; **435**:1–59.
5. Wang X, Jituan Z. A fast eigenvalue algorithm for Pascal matrices. *Applied Mathematics and Computation* 2006; **183**:711–716.
6. Luk FT, Qiao S. A fast eigenvalue algorithm for Hankel matrices. *Linear Algebra and its Applications* 2000; **316**: 171–182.
7. Luk FT, Qiao S. Analysis of a fast Hankel eigenvalue algorithm. *Proceedings of the SPIE 3807, Advanced Signal Processing Algorithms, Architectures, and Implementations IX*, Denver, USA, 1999; 324–333. DOI: 10.1117/12.367649.
8. Matiyasevich Yu. *Hidden life of Riemann's zeta function 2. Electrons and trains*, 2007. arXiv:0709.0028v2.
9. Matiyasevich Yu. The Riemann hypothesis and eigenvalues of related Hankel matrices. *Preprint 03/2014 of Steklov Institute of Mathematics at St.Petersburg*, 2014. (Available from: http://www.pdmi.ras.ru/preprint/2014/14-03.html) [Accessed 1 July 2014].
10. GNU. GNUMP library (Available from: http://gmplib.org) [Accessed 10 June 2013].
11. Fousse L, Hanrot G, Lefèvre V, Pélissier P, Zimmermann P. MPFR: a multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software* 2007; **33:** Article 13. (Availablefrom:http://doi.acm.org/10.1145/1236463.1236468) [Accessed 1 July 2014].
12. Johansson F. Arb library (Available from: http://fredrikj.net/arb/) [Accessed 20 Jan 2014].
13. Bailey D. Arprec library (Available from: http://crd-legacy.lbl.gov/~dhbailey/mpdist/) [Accessed 10 June 2013].
14. Advanpix. Matlab multiprecision computing toolbox, (Available from: http://www.advanpix.com/) [Accessed 20 Jan 2014].
15. Borgerding M. Kiss FFT. (Available from: http://sourceforge.net/projects/kissfft/) [Accessed 20 Jan 2014].
16. Beliakov G. *On fast matrix–vector multiplication with a Hankel matrix in multiprecision arithmetics*, 2014. Arxiv: 1402:5287.

DOI: 10.1002/nla

17. Borwein P, Choi S, Rooney B, Weirathmueller A (eds.) *The Riemann Hypothesis: A Resource for the Afficionado and Virtuoso Alike.* CMS Books in Mathematics, Springer: New York, 2008.
18. Clay Institute. *The millennium prize problems*, 2013. (Available from: http://www.claymath.org/millennium/) [Accessed 1 July 2014].
19. Matiyasevich Yu. *Hidden life of Riemann's zeta function. 1. Arrow, bow, and targets*, 2007. Arxiv: 0707.1983.
20. Beliakov G, Matiyasevich Yu. Approximation of Riemann's zeta function by finite Dirichlet series: a multiprecision numerical approach. *Experimental Mathematics* 2015; **24**:1–12.
21. Beliakov G, Matiyasevich Yu. A parallel algorithm for calculation of the determinants and minors using arbitrary precision arithmetics. *BIT Numerical Mathematics* 2015. DOI: 10.1007/s10543-015-0547-z.
22. Arbenz P. *Numerical methods for solving large scale eigenvalue problems, online lecture notes*. (Available from: http://people.inf.ethz.ch/arbenz/ewp/lnotes.html) [Accessed 31 March 2014].
23. Gu M, Eisenstat SC. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM Journal on Matrix Analysis and Applications* 1995; **16**:172–191.
24. Demmel J. *Applied Numerical Linear Algebra*. SIAM: Philadelphia, 1997.
25. Karatsuba A, Ofman Y. Multiplication of many-digital numbers by automatic computers. *Proceedings of the USSR Academy of Sciences (Translation in the academic journal Physics Doklady, 7 (1963), pp. 595–596)* 1962; **145**: 293–294.
26. Cormen T, Leiserson C, Rivest R, Stein C. *Introduction to algorithms* (3rd edn.) MIT Press: Cambridge, MA, 2009.
27. Cariow A, Gliszczyński M. Fast algorithms to compute matrix–vector products for Toeplitz and Hankel matrices. *PRZEGLAD ELEKTROTECHNICZNY (Electrical Review)* 2012; **88**:166–171.
28. Simon H. The Lanczos algorithm with partial reorthogonalization. *Mathematics of Comping* 1984; **42**:115–142.
29. Simon H. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra and its Applications* 1984; **61**:101–132.
30. Matiyasevich Yu. Numerical study of Riemann's zeta functions via determinants. (Available from: http://logic.pdmi.ras.ru/~yumat/talks/muenchen_2014/index.php) [Accessed 2 Dec 2014].

DOI: 10.1002/nla