Generation of test matrices with specified eigenvalues using floating-point arithmetic

Katsuhisa Ozaki¹ · Takeshi Ogita²

Received: 28 February 2021 / Accepted: 6 August 2021 / Published online: 17 September 2021
© The Author(s) 2021

Abstract

This paper concerns test matrices for numerical linear algebra using an error-free transformation of floating-point arithmetic. For specified eigenvalues given by a user, we propose methods of generating a matrix whose eigenvalues are exactly known based on, for example, Schur or Jordan normal form and a block diagonal form. It is also possible to produce a real matrix with specified complex eigenvalues. Such test matrices with exactly known eigenvalues are useful for numerical algorithms in checking the accuracy of computed results. In particular, exact errors of eigenvalues can be monitored. To generate test matrices, we first propose an error-free transformation for the product of three matrices YSX . We approximate S by S' to compute $YS'X$ without a rounding error. Next, the error-free transformation is applied to the generation of test matrices with exactly known eigenvalues. Note that the exactly known eigenvalues of the constructed matrix may differ from the anticipated given eigenvalues. Finally, numerical examples are introduced in checking the accuracy of numerical computations for symmetric and unsymmetric eigenvalue problems.

Keywords Test matrices · Floating-point arithmetic · Numerical linear algebra · Eigenvalue problems

1 Introduction

This paper concerns test matrices for numerical algorithms. If exact eigenvalues are known in advance, then it is useful for checking accuracy and stability of numerical

✉ Katsuhisa Ozaki
ozaki@sic.shibaura-it.ac.jp

¹ 307 Fukasaku, Minuma-ku, Saitama-shi, Saitama 337-8570, Japan

² 2-6-1 Zempukuji, Suginami-ku, Tokyo 167-8585, Japan

methods for problems in numerical linear algebra. There are several matrices whose exact eigenvalues are known, for example, Circular, Clement, Frank, and tridiagonal Toeplitz matrices. Test matrices are well summarized in [4, Chapter 28]. This paper aims to generate a matrix with exactly known eigenvalues based on some standard forms, for example, the Schur or the Jordan normal forms, or a block diagonal form. Let $X \in \mathbb{R}^{n \times n}$ be a non-singular matrix. For a matrix $S \in \mathbb{R}^{n \times n}$, we consider a form:

$$X^{-1}SX. \quad (1)$$

If (1) is the Schur normal form, X is an orthogonal matrix, and S is an upper triangular matrix. For the Jordan normal form, $S \in \mathbb{R}^{n \times n}$ is a matrix whose diagonal blocks are Jordan matrices. It indicates that S is an upper bidiagonal matrix and the sub-diagonal elements $s_{i,i+1}$ are either 1 or 0. For both normal forms, the diagonal elements s_{ii} ($1 \leq i \leq n$) are eigenvalues of A .

Let $\mathbb{F}$ be a set of floating-point numbers as defined in IEEE 754 [5]. If we use floating-point numbers and floating-point arithmetic, then there are following problems to get a matrix with exactly known eigenvalues using (1).

Problem 1 Even if all elements in a non-singular matrix X can be represented by floating-point numbers ($X \in \mathbb{F}^{n \times n}$), elements in X^{-1} may not be represented by floating-point numbers ($X^{-1} \notin \mathbb{F}^{n \times n}$).

Problem 2 Even if $X, S, X^{-1} \in \mathbb{F}^{n \times n}$, $X^{-1}SX \in \mathbb{F}^{n \times n}$ is not always satisfied.

We first prepare a diagonal matrix S and a non-singular matrix X , next obtain the inverse matrix X^{-1} using exact computations such as symbolic or rational arithmetic. Finally, we compute $X^{-1}SX$ using exact computations, and the result $X^{-1}SX$ is rounded to a floating-point matrix A . One may think s_{ii} ($1 \leq i \leq n$) are nearly eigenvalues of A , and these are useful as numerical tests. However, the Bauer-Fike theorem [2] states

$$\min_{\lambda} |\lambda - \mu| \leq \text{cond}(X) \|E\|, \quad (2)$$

where λ and μ are eigenvalues of A and $A + E = X^{-1}SX$, respectively. Therefore, s_{ii} can be far from the exact eigenvalues of A even if A is a floating-point matrix nearest to $X^{-1}SX$. In addition, the cost of symbolic or rational arithmetic is much expensive compared to that of numerical computations.

We propose methods that produce a matrix with specified eigenvalues on the basis of an error-free transformation of matrix multiplication. For a matrix $S \in \mathbb{F}^{n \times n}$ given by a user, we approximate S by $S' \in \mathbb{F}^{n \times n}$, and obtain $A := X^{-1}S'X \in \mathbb{F}^{n \times n}$ using the proposed method. We prove that no rounding error occurs in evaluating a product of three matrices $X^{-1}S'X$ using floating-point arithmetic. Hence, the exact eigenvalue of A can be known from the matrix S' .

This paper is organized as follows: In Section 2, basic theorems for floating-point numbers are introduced. In Section 3, we propose an error-free transformation for a product of three floating-point matrices. In Section 4, several methods for generating test matrices with exactly known eigenvalues are proposed based on the theory in Section 3, and we introduce numerical examples in checking the accuracy of numerical computations.

2 Basics of floating-point arithmetic

Notation and basic lemmas on floating-point arithmetic are first introduced. Notations $\text{fl}(\cdot)$, $\text{fl}_{\nabla}(\cdot)$, and $\text{fl}_{\Delta}(\cdot)$ indicate a computed result by floating-point arithmetic with rounding to nearest mode (`roundTiesToEven`), rounding downward mode (`roundTowardNegative`), and rounding upward mode (`roundTowardPositive`), respectively. Such rounding modes are defined in IEEE 754 [5]. We omit to write $\text{fl}(\cdot)$ for each operation for simplicity of description. For example, if we write $\text{fl}((a + b) + c)$ for $a, b, c \in \mathbb{F}$, then it means $\text{fl}(\text{fl}(a + b) + c)$. Assume that neither overflow nor underflow occurs in $\text{fl}(\cdot)$ and $\text{fl}_{\Delta}(\cdot)$. Let u and u_S be a round-off unit and the smallest positive number in $\mathbb{F}$, respectively. A constant realmax is the largest floating-point number in $\mathbb{F}$. For example, $u = 2^{-53}$, $u_S = 2^{-1074}$, and $\text{realmax} = 2^{1024}(1 - u)$ for binary64 in IEEE 754. A function $\text{ufp}(a)$ for $a \in \mathbb{R}$, the unit in the first place of the binary representation of a , is defined as

$$\text{ufp}(a) = \begin{cases} 0 & a = 0 \\ 2^{\lfloor \log_2 |a| \rfloor} & \text{otherwise} \end{cases}.$$

From the definition of the ufp function, we have

$$\text{ufp}(a) > \frac{1}{2}|a| \text{ for } a \neq 0. \quad (3)$$

We introduce several basic facts of floating-point numbers and floating-point arithmetic for $a, b \in \mathbb{F}$ and $x \in \mathbb{R}$:

$$a \in 2u \cdot \text{ufp}(a)\mathbb{Z}, \quad (4)$$

$$u_S\mathbb{Z} \ni |x| \leq \text{realmax}, \exists k = 2^w, w \in \mathbb{Z}, u_S k \mathbb{Z} \ni |x| \leq k \implies x \in \mathbb{F}, \quad (5)$$

$$a + b = \text{fl}(a + b) + \delta, |\delta| \leq u \cdot \text{ufp}(a + b), \quad (6)$$

$$|a| \geq |b|, c = \text{fl}(a + b) \implies \text{fl}(c - a) = c - a, \quad (7)$$

$$\text{ufp}(a + b) \leq \text{ufp}(\text{fl}(a + b)). \quad (8)$$

The definition of the floating-point numbers by IEEE 754 immediately derives (4) and (5). The relation (6) and the proposition (7) are written in [8] and [3], respectively. In addition, the following lemmas are used in our proof.¹

Lemma 1 *For $\sigma, a \in \mathbb{F}$, if $\text{ufp}(\sigma + a) \geq \text{ufp}(\sigma)$, then $\text{fl}(\sigma + a) \in 2u \cdot \text{ufp}(\sigma)\mathbb{Z}$. In addition, $\text{fl}((\sigma + a) - \sigma) \in 2u \cdot \text{ufp}(\sigma)\mathbb{Z}$ holds true.*

Proof Using (4), $\text{ufp}(\sigma + a) \leq \text{ufp}(\text{fl}(\sigma + a))$ from (8), and the assumption $\text{ufp}(\sigma + a) \geq \text{ufp}(\sigma)$ in turn, we have

$$\text{fl}(\sigma + a) \in 2u \cdot \text{ufp}(\text{fl}(\sigma + a))\mathbb{Z} \subseteq 2u \cdot \text{ufp}(\sigma + a)\mathbb{Z} \subseteq 2u \cdot \text{ufp}(\sigma)\mathbb{Z}. \quad (9)$$

¹Thanks to Dr. Atsushi Minamihata for the discussion.

Because (9) and $\sigma \in 2u \cdot \text{ufp}(\sigma)\mathbb{Z}$ from (4), $\text{fl}((\sigma + a) - \sigma) \in 2u \cdot \text{ufp}(\sigma)\mathbb{Z}$ is satisfied.

□

Applying Theorem 3.5 in [8] and an induction of binary tree described in [6], we can immediately obtain the following lemma.

Lemma 2 For $p \in \mathbb{F}^n$, assume that $p_i \in u\sigma\mathbb{Z}$ for all i , where $\sigma \in \mathbb{F}$ is a power of two. If $\sum_{i=1}^n |p_i| \leq \sigma$, then no rounding error occurs in the sum by any orders.

We assume that divide and conquer methods by, for example, Strassen [9] and Winograd [11] are not applied for dot product, matrix-vector, and matrix-matrix multiplication.

3 General theory for error-free computations

For $Y \in \mathbb{F}^{m \times n}$, $S \in \mathbb{F}^{n \times p}$, $X \in \mathbb{F}^{p \times q}$, we obtain a matrix $S' \in \mathbb{F}^{n \times p}$ ($S' \approx S$) in order to compute $YS'X$ without a rounding error. This technique is applied to generation of test matrices with exactly known eigenvalues.

We set several constants as follows: We define ϕ_{ij} for $x_{ij} \neq 0$ and ψ_{ij} for $y_{ij} \neq 0$ such that

$$x_{ij} \in \phi_{ij}\mathbb{Z}, \quad x_{ij} \notin 2\phi_{ij}\mathbb{Z}, \quad y_{ij} \in \psi_{ij}\mathbb{Z}, \quad y_{ij} \notin 2\psi_{ij}\mathbb{Z}, \quad (10)$$

where all ϕ_{ij} and ψ_{ij} are powers of two. The constants ϕ_{ij} and ψ_{ij} imply the unit in the last non-zero bit of x_{ij} and y_{ij} , respectively. Exceptionally, we set $\phi_{ij} = 0$ for $x_{ij} = 0$ and $\psi_{ij} = 0$ for $y_{ij} = 0$. Next, we set constants $\beta, \gamma, \theta, \omega$ as follows:

$$\beta = \max_{\substack{i,j,k \\ \phi_{kj} \neq 0}} \frac{\phi_{ij}}{\phi_{kj}}, \quad \gamma = \max_{\substack{i,j \\ \phi_{ij} \neq 0}} \frac{\text{ufp}(x_{ij})}{\phi_{ij}}, \quad (11)$$

$$\theta = \max_{\substack{i,j,k \\ \psi_{ik} \neq 0}} \frac{\psi_{ij}}{\psi_{ik}}, \quad \omega = \max_{\substack{i,j \\ \psi_{ij} \neq 0}} \frac{\text{ufp}(y_{ij})}{\psi_{ij}}. \quad (12)$$

These imply

$$\phi_{ij} \leq \beta \min_w \phi_{wj}, \quad \psi_{ij} \leq \theta \min_t \psi_{it} \quad (13)$$

$\phi_{wj} \neq 0 \qquad \psi_{it} \neq 0$

for all (i, j) pairs. Note that $\beta, \gamma, \theta, \omega \geq 1$ and $\max_{k,j} \phi_{kj} > 0$, $\max_{i,k} \psi_{ik} > 0$, because both X and Y are non-singular matrices.

If $s_{ij} = 0$ for all (i, j) pairs, then it is trivial that the matrix A is the zero matrix. Hence, we assume $\max_{i,j} |s_{ij}| > 0$. Let

- n_Y : the maximum of the number of non-zero elements in row of Y ,
- n_S : the maximum of the number of non-zero elements in row of S ,
- n_X : the maximum of the number of non-zero elements in column of X .

We set $n' = \min(n_S, n_X)$. We define a constant σ as

$$\sigma = 12\text{ufp}(\alpha)\beta\gamma\theta\omega \in \mathbb{F}, \quad (14)$$

where

$$\mathbb{F} \ni \alpha \geq n_Y \cdot n' \max_{i,j} |s_{ij}|. \quad (15)$$

Then,

$$\text{ufp}(\sigma) = 8\text{ufp}(\alpha)\beta\gamma\theta\omega \quad (16)$$

is satisfied because all of β, γ, θ , and ω are powers of two. It is possible to set α as $\text{fl}_{\Delta}(n \cdot n' \cdot \max_{i,j} |s_{ij}|)$ using floating-point arithmetic. A matrix S' is computed by

$$s'_{ij} = \text{fl}((\sigma + s_{ij}) - \sigma). \quad (17)$$

We introduce several lemmas.

Lemma 3 For $S \in \mathbb{F}^{n \times n}$, σ in (14), β and γ in (11), and ψ and ω in (12),

$$|s_{ij}| \leq \frac{\text{ufp}(\sigma)}{4n_Y \cdot n' \cdot \beta\gamma\theta\omega}, \quad (18)$$

and

$$|x_{ij}| \leq 2\gamma\phi_{ij} \leq 2\beta\gamma \min_w \phi_{wj}, \quad |y_{ij}| \leq 2\omega\psi_{ij} \leq 2\theta\omega \min_t \psi_{it} \quad (19)$$

$\phi_{wj} \neq 0 \qquad \qquad \qquad \psi_{it} \neq 0$

are satisfied for all (i, j) pairs.

Proof From (16), the inequality (3), and α in (15), we have

$$\text{ufp}(\sigma) = 8\text{ufp}(\alpha)\beta\gamma\theta\omega > 4\alpha\beta\gamma\theta\omega \geq 4n_Y \cdot n' \cdot |s_{ij}|\beta\gamma\theta\omega.$$

Because $n_Y, n', \beta, \gamma, \theta, \omega \geq 1$, the inequality (18) can be obtained.

For $x_{ij} = 0$ and $y_{ij} = 0$, the inequalities (19) are trivially satisfied. Therefore, we assume $x_{ij} \neq 0$ ($\phi_{ij} \neq 0$) and $y_{ij} \neq 0$ ($\psi_{ij} \neq 0$) for the rest of the proof. From (11), (12), and (3), we have

$$\gamma = \max_{i,j} \frac{\text{ufp}(x_{ij})}{\phi_{ij}} \geq \frac{\text{ufp}(x_{ij})}{\phi_{ij}} > \frac{|x_{ij}|}{2\phi_{ij}},$$

$$\omega = \max_{i,j} \frac{\text{ufp}(y_{ij})}{\psi_{ij}} \geq \frac{\text{ufp}(y_{ij})}{\psi_{ij}} > \frac{|y_{ij}|}{2\psi_{ij}}$$

for all (i, j) pairs. Then, using (13),

$$|x_{ij}| < 2\gamma\phi_{ij} \leq 2\beta\gamma \min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj}, \quad |y_{ij}| < 2\omega\psi_{ij} \leq 2\theta\omega \min_{\substack{t \\ \psi_{it} \neq 0}} \psi_{it}$$

are satisfied for all (i, j) pairs. Therefore, (19) is proved. $\square$

Lemma 4 For $S \in \mathbb{R}^{n \times n}$ and σ in (14),

$$\text{ufp}(\sigma + s_{ij}) = \text{ufp}(\sigma)$$

is satisfied for all (i, j) pairs.

Proof Using (14) and (16) gives

$$\begin{aligned} \sigma + s_{ij} &= 8\text{ufp}(\alpha)\beta\gamma\theta\omega + 4\text{ufp}(\alpha)\beta\gamma\theta\omega + s_{ij} \\ &= \text{ufp}(\sigma) + 4\text{ufp}(\alpha)\beta\gamma\theta\omega + s_{ij}. \end{aligned} \quad (20)$$

From $\beta, \gamma, \theta, \omega \geq 1$, (3), and $\alpha \neq 0$ from $\max_{i,j} |s_{ij}| > 0$, we have

$$4\text{ufp}(\alpha)\beta\gamma\theta\omega + s_{ij} > \alpha + s_{ij} \geq 0. \quad (21)$$

Therefore, application of (20) and (21) yields

$$\sigma + s_{ij} > \text{ufp}(\sigma). \quad (22)$$

From (15) and (3), we have

$$|s_{ij}| \leq \alpha < 2\text{ufp}(\alpha)\beta\gamma\theta\omega \quad (23)$$

and using (22), (14), (23), and (16) gives

$$\text{ufp}(\sigma) < \sigma + s_{ij} \leq \sigma + |s_{ij}| < 14\text{ufp}(\alpha)\beta\gamma\theta\omega < 2\text{ufp}(\sigma).$$

Therefore,

$$\text{ufp}(\sigma + s_{ij}) = \text{ufp}(\sigma)$$

can be derived. $\square$

Lemma 5 Assume that s'_{ij} is obtained by (17). Then,

$$|s'_{ij}| \leq \frac{\text{ufp}(\sigma)}{4n_Y \cdot n' \cdot \beta\gamma\theta\omega} + u \cdot \text{ufp}(\sigma)$$

is satisfied.

Proof Using the definition of σ in (14) gives $\sigma \geq |s_{ij}|$ for all (i, j) pairs. From this and (7), we derive

$$s'_{ij} = \text{fl}((\sigma + s_{ij}) - \sigma) = \text{fl}(\sigma + s_{ij}) - \sigma = \sigma + s_{ij} + \delta - \sigma = s_{ij} + \delta,$$

where $|\delta| \leq u \cdot \text{ufp}(\sigma + s_{ij}) = u \cdot \text{ufp}(\sigma)$ from (6) and Lemma 4. Therefore, from Lemma 3

$$|s'_{ij}| = |s_{ij} + \delta| \leq |s_{ij}| + |\delta| \leq \frac{\text{ufp}(\sigma)}{4n_Y \cdot n' \cdot \beta \gamma \theta \omega} + u \cdot \text{ufp}(\sigma).$$

□

We show that rounding error never occurs in the evaluation of $YS'X$.

Theorem 1 *Let $\sigma \in \mathbb{F}$ be defined as in (14) from the matrices Y, X and S . Assume that $4n_Y \cdot n' \cdot u\beta\gamma\theta\omega \leq 1$. If a matrix S' is obtained by (17), then*

$$\text{fl}(Y(S'X)) = YS'X.$$

Proof First, we show $\text{fl}(S'X) = S'X$. If $x_{ij} = 0$, $1 \leq i \leq p$ for some j (the j -th column vector of X is the zero vector), then

$$\sum_{k=1}^p s'_{ik}x_{kj} = \text{fl}\left(\sum_{k=1}^p s'_{ik}x_{kj}\right) = 0.$$

Hence, we assume there exists $x_{ij} \neq 0$ for any j for the rest of the proof. Since $n_Y, n', \beta, \theta, \omega \geq 1$ and $4n_Y \cdot n' \cdot u\beta\gamma\theta\omega \leq 1$ from the assumption, we have

$$\frac{1}{4n_Y \cdot \theta \omega} + un'\beta\gamma < 1. \quad (24)$$

Using Lemma 1 and (10) gives

$$s'_{ik} \in 2u \cdot \text{ufp}(\sigma)\mathbb{Z}, \quad x_{kj} \in \phi_{kj}\mathbb{Z} \subseteq \left(\min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj}\right)\mathbb{Z}, \quad (25)$$

and they yield

$$s'_{ik}x_{kj} \in 2u \left(\min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj}\right) \text{ufp}(\sigma)\mathbb{Z}. \quad (26)$$

Next, we take an upper bound of $\left| \sum_{k=1}^p s'_{ik} x_{kj} \right|$ using Lemma 5, Lemma 3, and (24):

$$\begin{aligned} \left| \sum_{k=1}^p s'_{ik} x_{kj} \right| &\leq \sum_{k=1}^p |s'_{ik}| |x_{kj}| \\ &\leq n' \left(\frac{\text{ufp}(\sigma)}{4n_Y \cdot n' \cdot \beta_Y \theta \omega} + u \cdot \text{ufp}(\sigma) \right) 2\beta_Y \min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj} \\ &= 2 \left(\frac{1}{4n_Y \theta \omega} + un' \beta_Y \right) \left(\min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj} \right) \text{ufp}(\sigma) \end{aligned} \quad (27)$$

$$< 2 \left(\min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj} \right) \text{ufp}(\sigma). \quad (28)$$

Therefore, from (26) and (28), Lemma 2 derives $\text{fl}(S'X) = S'X$.

Next, we focus on $YS'X$. The (i, j) element of $YS'X$ is

$$(YS'X)_{ij} = \sum_{\ell=1}^n y_{i\ell} \left(\sum_{k=1}^p s'_{\ell k} x_{kj} \right).$$

Using (26) and (10) gives

$$s'_{\ell k} x_{kj} \in 2u \cdot \text{ufp}(\sigma) \left(\min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj} \right) \mathbb{Z}, \quad y_{i\ell} \in \psi_{i\ell} \mathbb{Z} \subseteq \min_{\substack{t \\ \psi_{it} \neq 0}} \psi_{it} \mathbb{Z}.$$

Therefore, we derive

$$(YS'X)_{ij} = \sum_{\ell=1}^n y_{i\ell} \left(\sum_{k=1}^p s'_{\ell k} x_{kj} \right) \in 2u \left(\min_{\substack{t \\ \psi_{it} \neq 0}} \psi_{it} \right) \left(\min_{\substack{w \\ \phi_{wj} \neq 0}} \phi_{wj} \right) \text{ufp}(\sigma) \mathbb{Z}. \quad (29)$$

Application of (27), Lemma 3, and $1 + 4n_Y \cdot n' \cdot \mathfrak{u}\beta\gamma\theta\omega \leq 2$ obtained from the assumption yields

$$\begin{aligned}
 & \left| \sum_{\ell=1}^n y_{i\ell} \left(\sum_{k=1}^p s'_{\ell k} x_{kj} \right) \right| \leq \sum_{\ell=1}^n |y_{i\ell}| \left(\sum_{k=1}^p |s'_{\ell k}| |x_{kj}| \right) \\
 & \leq \sum_{\ell=1}^n 2|y_{i\ell}| \left(\frac{1}{4n_Y\theta\omega} + \mathfrak{u}n'\beta\gamma \right) \left(\min_{\phi_{wj} \neq 0} \phi_{wj} \right) \mathfrak{u}\mathfrak{f}\mathfrak{p}(\sigma) \\
 & \leq 4n_Y\theta\omega \left(\min_{\psi_{it} \neq 0} \psi_{it} \right) \left(\frac{1}{4n_Y\theta\omega} + \mathfrak{u}n'\beta\gamma \right) \left(\min_{\phi_{wj} \neq 0} \phi_{wj} \right) \mathfrak{u}\mathfrak{f}\mathfrak{p}(\sigma) \\
 & \leq (1 + 4\mathfrak{u}n_Y \cdot n' \cdot \beta\gamma\theta\omega) \left(\min_{\psi_{it} \neq 0} \psi_{it} \right) \left(\min_{\phi_{wj} \neq 0} \phi_{wj} \right) \mathfrak{u}\mathfrak{f}\mathfrak{p}(\sigma) \\
 & \leq 2 \left(\min_{\psi_{it} \neq 0} \psi_{it} \right) \left(\min_{\phi_{wj} \neq 0} \phi_{wj} \right) \mathfrak{u}\mathfrak{f}\mathfrak{p}(\sigma). \tag{30}
 \end{aligned}$$

Hence, from (29) and (30), Lemma 2 verifies $\mathfrak{f}\mathfrak{l}(Y(S'X)) = YS'X$.

□

If we change the definition of n' , it is possible to verify $\mathfrak{f}\mathfrak{l}((YS')X) = YS'X$ similarly. Remark that the proposed method has reproducibility,² because there is no rounding error in any order of computations in $\mathfrak{f}\mathfrak{l}(Y(S'X))$.

4 Test matrices based on specified matrices

In this section, we introduce how to generate a matrix with exactly known eigenvalues. We prepare $S, X, Y \in \mathbb{F}^{n \times n}$ ($Y = X^{-1}$). Assume that exact eigenvalues can be known quickly from the matrix S . For example, S is a diagonal matrix, triangular matrices, and tridiagonal Toeplitz matrix. We use MATLAB R2020a on Windows 10 with Intel(R) Core(TM) i7-8665U in numerical examples in this section.

We introduce a basic algorithm for the generation of test matrices using MATLAB-like code on the basis of the discussion in Section 3. Greek alphabets cannot be used

²Reproducibility means ability to obtain always bit-wise identical results on the same input data on the same or different architectures [1].

for variables on MATLAB. However, we exceptionally accept to use Greek alphabets in the description of algorithms for consistency with the theories in this paper.

Algorithm 1 Let $S, X, Y \in \mathbb{F}^{n \times n}$ where X is non-singular and $Y = X^{-1}$. The following algorithm produces matrices A and Z whose eigenvalues are the same.

```

1: function [A,Z]=alg_basic(Y,S,X)
2: n = size(S, 2);
3: ny = max(sum(Y~= 0, 2));
4: nd = max(sum(S~= 0, 2));
5: nx = max(sum(X~= 0, 1));
6: [ $\beta, \gamma$ ] = get_bg(X);
7: [ $\theta, \omega$ ] = get_tw(Y);
8: feature('setround', Inf);
9:  $\alpha = ny * \min(nd, nx) * \max(\text{abs}(S(:)))$ ;
10: feature('setround', 0.5);
11:  $\sigma = 12 * \text{ufp}(\alpha) * \beta * \gamma * \theta * \omega$ ;
12: if  $4 * ny * \min(nd, nx) * u * \beta * \gamma * \theta * \omega > 1$ 
13:     warning('Eigenvalues of A and Z may be different')
14: end
15:  $Z = (\sigma + S) - \sigma$ ;
16:  $A = Y * Z * X$ ; %no rounding error
17: end

```

In the algorithm, $\text{ufp}(\cdot)$ at line 11 is not a built-in function in MATLAB. The calculation of $\text{ufp}(a)$ for $a \in \mathbb{F}$ involves only three floating-point operations (see Algorithm 3.6 in [7]). Therefore, the cost for the ufp function is negligible. The function get_bg at line 6 in Algorithm 1 produces β and γ according to (11). The function get_tw similarly gives θ and ω based on (12).

To set multiple eigenvalues can be possible, because if $s_{ij} = s_{kl}$, then $z_{ij} = z_{kl}$. We can also generate a rank deficient matrix with exactly known eigenvalues. However, even if $s_{ij} \neq s_{kl}$, $z_{ij} = z_{kl}$ may be satisfied. It happens for relatively small $|s_{ij}|$ compared to $\max_{i,j} |s_{ij}|$. If $u\sigma \geq |s_{ij}|$, then $z_{ij} = 0$. In the worst case, the matrix Z becomes the zero matrix, and an obtained matrix A is useless. In addition, a test matrix with the specified singular values can be generated using $Y = X^T$.

Note that we can generate a real matrix with complex eigenvalues. Let i be the imaginary unit. The basic of linear algebra derives that $a \pm bi$ are eigenvalues of the following skew-symmetric matrix:

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix} = \frac{1}{2} \begin{pmatrix} i & -i \\ 1 & 1 \end{pmatrix} \begin{pmatrix} a - bi & 0 \\ 0 & a + bi \end{pmatrix} \begin{pmatrix} -i & 1 \\ i & 1 \end{pmatrix},$$

where

$$\frac{1}{2} \begin{pmatrix} i & -i \\ 1 & 1 \end{pmatrix} \begin{pmatrix} -i & 1 \\ i & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

For a discussion of complex eigenvalues, we set 2-by-2 matrices to the diagonal block of D . For example, if $a \pm bi$, $c \pm di$, e ($a, b, c, d, e \in \mathbb{F}$) are expected eigenvalues, then we set a skew-symmetric matrix as

$$S = \begin{pmatrix} a & b & 0 & 0 & 0 \\ -b & a & 0 & 0 & 0 \\ 0 & 0 & c & d & 0 \\ 0 & 0 & -d & c & 0 \\ 0 & 0 & 0 & 0 & e \end{pmatrix}.$$

If one wants to obtain a matrix with exactly known eigenvectors, then we obtain the matrix as follows.

$$\frac{1}{2}Y = \begin{pmatrix} -i & 1 & 0 & 0 & 0 \\ i & 1 & 0 & 0 & 0 \\ 0 & 0 & -i & 1 & 0 \\ 0 & 0 & i & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Note that

$$\text{fl}((\sigma + b) - \sigma) \neq -\text{fl}((\sigma - b) - \sigma)$$

in some cases due to a problem of rounding error. Then, S' is not a skew-symmetric matrix. To solve the problem, we compute only

$$s'_{ij} = \text{fl}((\sigma + s_{ij}) - \sigma), \quad i \leq j$$

and set $s'_{ji} = -s'_{ij}$ by copy of the computed result. For

$$[A, Z] = \text{alg_basic}(Y, S, X),$$

Z is not a skew-symmetric matrix, while S is a skew-symmetric matrix.

The condition $X, Y \in \mathbb{F}^{n \times n}$ ($Y = X^{-1}$) is very special. Therefore, we use specified matrices for X in (1) from the following sections.

4.1 Using weighing matrix

Assume that $D \in \mathbb{F}^{n \times n}$ is a diagonal matrix ($n' = 1$). First, we use a weighing matrix. Let I be the identity matrix. If $W \in \mathbb{R}^{n \times n}$ with weight $c \in \mathbb{N}$ is a weighing matrix, $W^T W = cI$ and $w_{ij} \in \{-1, 0, 1\}$ for all (i, j) pairs. The weighing matrix is useful for the candidate of the matrix X , because $\beta = \gamma = \theta = \omega = 1$ in (11) and (12), and $Y = X^{-1} = \frac{1}{c} W^T$.

Let $W_{n,c} \in \mathbb{F}^{n \times n}$ be a weighing matrix such that $W_{n,c}^T W_{n,c} = cI$. The notation $\otimes$ means the Kronecker product. For $n = \prod_{i=1}^w n_i$ and $c = \prod_{i=1}^w c_i$, $n_i, c_i \in \mathbb{N}$, a weighing matrix $W_{n,c}$ can be obtained by $W_{n_1,c_1} \otimes W_{n_2,c_2} \otimes \cdots \otimes W_{n_w,c_w}$.

If we set $Y = \frac{1}{c} W_{n,c}^T = \frac{1}{c_1} W_{n_1,c_1}^T \otimes \cdots \otimes \frac{1}{c_w} W_{n_w,c_w}^T$, then the matrix Y may not be representable in floating-point numbers. It happens when c is not a power of two. For a given diagonal matrix D , we obtain σ from $\text{fl}\left(\frac{1}{c}D\right)$ instead of D and compute

$$d'_{ii} = \text{fl}((\sigma + d_{ii}/c) - \sigma).$$

Then, we can compute $W_{n,c}^T(D'W_{n,c}) =: A$ without rounding errors. Hence, cd'_{ii} is the exact eigenvalue of the matrix A . Note that cd'_{ii} may not be representable in the floating-point numbers. An error-free transformation [3] produces floating-point vectors p and q such that

$$cd' = p + q, \quad p = \text{fl}(cd'), \quad p, q \in \mathbb{F}^n.$$

It is well known that we quickly obtain p and q using fused multiply-add (FMA instruction defined in IEEE 754). The function $\text{fma}(a, b, c)$ for $a, b, c \in \mathbb{F}$ provides a nearest floating-point number of $ab + c$. Then,

$$cd' = p + q, \quad p = \text{fl}(cd'), \quad q = \text{fma}(c, d', -p).$$

Hadamard matrix, the special case of weighing matrix ($c = n$), is very useful for the candidate of X .

The following codes generate a test matrix on MATLAB. Let $d \in \mathbb{F}^n$ be a vector with expected eigenvalues. The exact eigenvalue of $A \in \mathbb{F}^{n \times n}$ is $p_i + q_i \approx d_i$. Note that the function `hadamard` on MATLAB R2020a in the following algorithm works when $n, n/12$ or $n/20$ is a power of two.³

Algorithm 2 From a given vector $d \in \mathbb{F}^n$, the following algorithm produces a matrix A whose eigenvalues are exactly $p_i + q_i (\approx d_i)$.

```

1: function [A, p, q] = alg_sym(d)
2:   n = length(d);
3:   H = hadamard(n);
4:   feature('setround', Inf);
5:   sigma = 12 * ufp(max(abs(d/n)));
6:   feature('setround', 0.5);
7:   if 4 * n * u > 1
8:     warning('Eigenvalues of A may not be p + q')
9:   end
10:  y = (sigma + d/n) - sigma;
11:  A = H' * (y * H); % H^T diag(y_1, y_2, ..., y_n) H;
12:  [p, q] = TwoProduct(n, y);
13: end

```

³This is not only for MATLAB R2020a, but also for more early versions.

Note that Algorithm 2 produces a real symmetric matrix. For $n \in \mathbb{F}$ and $y \in \mathbb{F}^n$, the function `TwoProduct` at line 12 transforms ny to $p + q$, $p, q \in \mathbb{F}^n$. If n is a power of two, q at line 12 is the zero vector.

At line 11, a matrix multiplication is computed. In the special cases, we can reduce the cost of the matrix multiplication. Let H_n be a Hadamard matrix with dimension n . We can generate one of $2n$ -by- $2n$ Hadamard matrices by the following manner [10]

$$H_{2n} = \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix} = H_n \otimes \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

For a diagonal matrix $D \in \mathbb{F}^{2n \times 2n}$, there exist matrices $M, N \in \mathbb{R}^{n \times n}$ such that

$$D = \begin{pmatrix} D_1 & O \\ O & D_2 \end{pmatrix}, \quad DH_{2n} = \begin{pmatrix} M & M \\ N & -N \end{pmatrix}, \quad M = D_1 H_n, \quad N = D_2 H_n$$

where D_1 and D_2 are n -by- n diagonal matrices. Then, the matrices M and N can quickly be obtained, because D_1 and D_2 are diagonal matrices. Finally, we compute $H_{2n}^T DH_{2n}$ by

$$H_{2n}^T DH_{2n} = \begin{pmatrix} H_n^T & H_n^T \\ H_n^T & -H_n^T \end{pmatrix} \begin{pmatrix} M & M \\ N & -N \end{pmatrix} = \begin{pmatrix} H_n^T M + H_n^T N & H_n^T M - H_n^T N \\ H_n^T M - H_n^T N & H_n^T M + H_n^T N \end{pmatrix}.$$

Therefore, we only compute $H_n^T M = H_n^T D_1 H_n$ and $H_n^T N = H_n^T D_2 H_n$ as matrix multiplications to obtain $H_{2n}^T DH_{2n}$. The cost of matrix multiplications can be reduced using this technique recursively.

Let $r \cdot n$ be a matrix size for the output, where $r, n \in \mathbb{N}$ and r is a power of two.

$$C = \begin{pmatrix} C_{11} & C_{12} & \dots & C_{1r} \\ C_{21} & C_{22} & \dots & C_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ C_{r1} & C_{r2} & \dots & C_{rr} \end{pmatrix} = H_{rn}^T D H_{rn}, \quad D = \begin{pmatrix} D_1 & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & D_2 & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \dots & D_r \end{pmatrix}$$

where $D_i \in \mathbb{F}^{n \times n}$, $1 \leq i \leq r$ are diagonal matrices, and $\mathbf{O}$ is the zero matrix.

Then, we only compute the following r matrix multiplications:

$$H_n^T D_1 H_n, \quad H_n^T D_2 H_n, \quad \dots, \quad H_n^T D_r H_n,$$

and C_{ij} is obtained by

$$C_{ij} = \sum_{k=1}^r s_{ki} s_{kj} H_n^T D_k H_n, \quad S = H_r.$$

The total cost becomes $2kn^3 + \mathcal{O}(n^2 + k)$ floating-point operations for r -by- r block division. Note that the original cost without block computations is $2(kn)^3 + \mathcal{O}((kn)^2)$, that is much expensive compared to block computations for large k .

We introduce an algorithm producing a matrix with eigenvalues based on the block computations.

Algorithm 3 From a vector $d \in \mathbb{R}^n$, the following algorithm produces a matrix A whose eigenvalues are exactly $p_i + q_i$ ($\approx d_i$). A is obtained by r -by- r block computations. Assume that n is dividable by r .

```

1: function [A, p, q] = alg1_sym_div(d, r)
2: n = length(d);
3: n2 = n/r;
4: H = hadamard(n2);
5: H2 = hadamard(r);
6: d = d/n;
7: t = ufp(max(abs(d)));
8: feature('setround', Inf);
9: t = 12 * n * t;
10: feature('setround', 0.5);
11: y = (t + d) - t;
12: A = zeros(n);
13: for i = 1:r, T{i} = H * (y((i - 1) * n2 + 1 : i * n2) .* H); end
14: for i = 1:r
15:     i2 = (i - 1) * n2 + 1 : i * n2;
16:     for j = 1:r
17:         j2 = (j - 1) * n2 + 1 : j * n2;
18:         S = zeros(n2);
19:         for k = 1:r
20:             S = S + H2(i, k) * H2(k, j) * T{k};
21:         end
22:         A(i2, j2) = S;
23:     end
24: end
25: [p, q] = TwoProduct(y, n);
26: end

```

We show numerical examples for the performance using this technique. Figures 1 and 2 show computing times with r -by- r block computations. $r = 1$ means a computing time for Algorithm 2. Computing times for solving linear systems (`linsolve`) and eigenvalue problems (`eig`) using MATLAB are shown for comparison. Note that `eig` in this example produces only approximate eigenvalues such that $d = \text{eig}(A)$. The figures indicate that computing time for the generation of test matrices is much shorter than that for solving linear systems and that for eigenvalue problems.

Remark 1 We mentioned that MATLAB's `hadamard` function works in the case that n , $n/12$, or $n/20$ is the power of two. Even if n does not satisfy that restriction, the test matrices can be generated. Assume that H_k exists and $k < n$. Let $P \in \mathbb{R}^{(n-k) \times (n-k)}$ be a permutation matrix. We generate

$$\begin{pmatrix} H_n & O \\ O & P \end{pmatrix}$$

Fig. 1 Comparison of computing times ($n = 4096$)

Fig. 2 Comparison of computing times ($n = 16384$)

instead of the Hadamard matrix. Then, D' is obtained by

$$d'_{ii} = \begin{cases} \text{fl}((\sigma + d_{ii}/k) - \sigma), & i \leq k \\ \text{fl}((\sigma + d_{ii}) - \sigma), & i > k. \end{cases}$$

The exact eigenvalue is kd'_{ii} for $1 \leq i \leq k$ and d'_{ii} for $k + 1 \leq i \leq n$.

We checked the accuracy of approximate eigenvalues obtained by `eig`. The vector $d \in \mathbb{F}^n$, $n = 4096$ in Algorithm 2 is set as follows.

Ex.1: $d_1 = 1$, $d_n = 10^{10}$, intermediate values d_i , $2 \leq i \leq n - 1$, are geometrically distributed.

Ex.2: $d_i = 1$, $d_n = 10^{10}$, $1 \leq i \leq n - 1$.

We compare accuracy of eigenvalues by $d = \text{eig}(A)$ and $[V, D] = \text{eig}(A)$ in MATLAB. Let $\hat{\lambda}_1 \leq \hat{\lambda}_2 \leq \dots \leq \hat{\lambda}_n$ be computed eigenvalues. Figures 3 and 4 show relative errors

$$\frac{|p_i + q_i - \hat{\lambda}_i|}{|p_i + q_i|} \quad (31)$$

for all i , where p and q are obtained by Algorithm 2. From Fig. 3 for Ex.1, the accuracy of relatively small eigenvalue tends to be worse. Besides, approximate eigenvalues from $[V, D] = \text{eig}(A)$ are better than those from $d = \text{eig}(A)$ in terms of accuracy. In the case of Ex.2, we can see from Fig. 4 that some of the eigenvalues are not so accurate, and several computed eigenvalues are exact.

Using the Hadamard matrix, we introduce an algorithm generating a test matrix based on the Jordan normal form ($n' = 2$).

Fig. 3 The quantity (31) for Ex.1

Fig. 4 The quantity (31) for Ex.2

Algorithm 4 The following algorithm produces a test matrix A with exactly known eigenvalues $p_i + q_i \approx v$ based on Jordan normal form. $v \in \mathbb{F}^n$ is a vector and $w \in \mathbb{F}^{n-1}$ keeps information of sub-diagonal elements in a Jordan matrix D , i.e., $w_i = d_{i,i+1}$, $1 \leq i \leq n-1$.

```

1: function [A, p, q] = alg_jordan(v, w)
2: n = length(v);
3: H = hadamard(n);
4: feature('setround', Inf)
5: s = 24 * n * ufp(max(abs([v; 1]))/n); %n' = 2 in (2)
6: feature('setround', 0.5)
7: y = (s + v/n) - s;
8: A = [0; w].* H;
9: A = H * (A([n, 1:n-1], :) + y.* H);
10: [p, q] = TwoProduct(n, y);
11: end
    
```

We show numerical examples using Algorithm 4. The vectors $v \in \mathbb{F}^n$ and $w \in \mathbb{F}^{n-1}$ for $n = 4096$ as arguments of Algorithm 4 are set as follows:

Ex.3: $v_1 = 1$, $v_i = 10^5$, $2 \leq i \leq n$, $w = (1, 1, \dots, 1)^T$.

Ex.4: $v_i = 1$, $1 \leq i \leq n-1$, $v_n = 10^5$, $w = (1, 1, \dots, 1)^T$.

Then, we execute $[A, p, q] = \text{alg_jordan}(v, w)$. Figures 5 and 6 show the quantity (31) for all i , respectively. Approximate eigenvalues for these numerical

Fig. 5 The quantity (31) for Ex.3

examples have small imaginary parts so that we checked the accuracy of the real part of the computed eigenvalues. In both cases, the matrices have $n - 1$ multiple eigenvalues, but there are differences in the numerical examples. From Fig. 6, computed eigenvalues for $n - 1$ small eigenvalues are very inaccurate.

Fig. 6 The quantity (31) for Ex.4

Algorithm 5 From a vector $d \in \mathbb{F}^n$, $n \leq u^{-1}$, the following algorithm produces a matrix A and its exact eigenvalue $w_i \approx d_i$. The sparsity of L and U is controlled by a constant $k \in \mathbb{N}$.

```

1: function [A, r] = alg_intmat(d, k)
2: n = length(d);
3: L = spones(tril(sprandn(n, n, k)));
4: L(1:n+1:n*n) = 1;
5: U = spones(triu(sprandn(n, n, k)));
6: U(1:n+1:n*n) = 1;
7: X = L * U;
8: Y = (eye(n)/U) * (eye(n)/L);
9: feature('setround', -Inf);
10: Td = Y * X - eye(n);
11: feature('setround', Inf);
12: Tu = Y * X - eye(n);
13: if max(max(Tu - Td))~ = 0 && norm(Td, Inf) == 0
14:     error('Y is not inverse of X');
15: end
16: feature('setround', 0.5);
17: [β, γ] = get_bg(X);
18: [θ, ω] = get_tw(Y);
19: feature('setround', Inf);
20: α = n * max(abs(d));
21: feature('setround', 0.5);
22: σ = 12 * ufp(α) * β * γ * θ * ω;
23: if 4 * n * n * u * β * γ * θ * ω > 1,
24:     error('generation failed');
25: end
26: r = (σ + d) - σ;
27: A = Y * (r * X); %A = Y * diag(r) * X
28: end

```

4.2 Eigenvector matrix X using random integer matrices

We use the LU decomposition and generate integer matrices⁴ for X . Let L and U be a unit lower triangular matrix and a unit upper triangular matrix, respectively. We generate $X = LU$, where we set L and U as 0-1 matrices in order to obtain small constants β , γ , θ , and ω . Then, $Y = X^{-1}$ is also an integer matrix. Because L and U are 0-1 matrices, there is no rounding error in $\text{fl}(LU)$ up to $n = u^{-1}$. On the other hand, the problem is that the exact $Y = X^{-1}$ can be obtained or not. Let $\tilde{Y}$ be a numerically obtained result for X^{-1} . If $\mathbf{O} = \text{fl}_{\nabla}(\tilde{Y}X - I) = \text{fl}_{\Delta}(\tilde{Y}X - I)$, where

⁴We appreciate Prof. Siegfried M. Rump's advice for this discussion.

Table 1 Comparison of properties for k for $n = 1,000$

k	$\beta\gamma\theta\omega$	$\text{cond}(X)$	$\max_i \frac{ d_i - r_i }{ d_i }$	Density of A
0.001	2.80e+00	2.18e+02	8.70e-09	3.56e-03
0.002	4.00e+00	2.02e+03	3.25e-08	1.19e-02
0.003	4.80e+00	1.22e+04	1.88e-07	3.17e-02
0.004	4.00e+00	6.04e+04	3.80e-08	6.89e-02
0.005	8.00e+00	1.38e+05	2.68e-08	1.27e-01
0.006	8.80e+00	6.76e+05	1.95e-07	2.08e-01
0.007	1.60e+01	1.76e+06	1.63e-07	3.14e-01
0.008	1.92e+01	5.63e+06	4.87e-08	4.22e-01
0.009	2.24e+01	1.81e+07	8.84e-08	5.37e-01
0.0010	4.48e+01	4.41e+07	2.18e-07	6.47e-01

I is the identity matrix, then $\tilde{Y} = X^{-1}$. Using MATLAB notation, L and U are given as

$$L = \text{spones}(\text{tril}(\text{sprandn}(n, n, k))); \quad L(1:n+1:n*n) = 1; \quad (32)$$

$$U = \text{spones}(\text{triu}(\text{sprandn}(n, n, k))); \quad U(1:n+1:n*n) = 1; \quad (33)$$

where $0 \leq k \leq 1$ is density of a matrix. The sparsity of L and U is controlled by $k \in \mathbb{N}$.

The following algorithm generates A using random integer matrices.

We introduce numerical examples for Algorithm 5. We set $n = 1,000, 5,000$, and $d \in \mathbb{F}^n$ is generated by $\text{randn}(n, 1)$ using MATLAB R2020a. Table 1 shows

- The product of four constants $\beta\gamma\theta\omega$
- The 2-norm condition number of X , i.e., $\|X^{-1}\|_2 \|X\|_2 = \|Y\|_2 \|X\|_2$
- The relative change between d_i and r_i , i.e., $\max_i \frac{|d_i - r_i|}{|d_i|}$

Table 2 Comparison of properties for k for $n = 5,000$

k	$\beta\gamma\theta\omega$	$\text{cond}(X)$	$\max_i \frac{ d_i - r_i }{ d_i }$	Density of A
0.001	4.00e+00	5.56e+05	1.91e-06	2.83e-02
0.002	1.92e+01	1.41e+08	2.11e-04	2.37e-01
0.003	2.88e+01	3.80e+10	3.92e-05	6.14e-01
0.004	6.40e+01	8.14e+12	5.63e-05	8.57e-01
0.005	1.02e+02	1.29e+15	3.43e-05	9.48e-01
0.006	1.79e+02	2.57e+17	2.00e-05	9.78e-01
0.007	2.56e+02	2.71e+20	2.13e-05	9.90e-01
0.008	2.56e+02	1.04e+21	1.50e-05	9.95e-01

- The density of the matrix A , i.e., the number of non-zero elements of A divided by n^2

for each k . The numbers in Tables 1 and 2 are averages of 10 examples. If k is small, we can obtain small $\beta\gamma\theta\omega$. It makes the relative change for v small. If k becomes large, then

- The relative change for a small absolute value,
- The condition number of X , and
- Density of A

also become large.

5 Conclusion

We proposed methods generating matrices with specified eigenvalues based on the error-free transformation of floating-point numbers. They are useful for checking the accuracy and stability of numerical algorithms. Finally, we provided several possibilities for an eigenvector matrix and showed how to generate a real matrix with complex eigenvalues. The discussion can be extended to the generation of test matrices with complex eigenvalues and eigenvectors.

Acknowledgements This research is partially supported by JST / CREST and JSPS Kakenhi (20H04195). We sincerely express our thanks to the reviewer for the fruitful comments and Dr. Takeshi Terao for his kind advice of simplified codes of MATLAB.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Ahrens, P., Nguyen, H.D., Demmel, J.: Efficient Reproducible Floating Point Summation and BLAS, EECS Department, UC Berkeley, Technical Report No. UCB/EECS2016-121 Berkeley (2016)
2. Bauer, F.L., Fike, C.T.: Norms and exclusion theorems. *Numer. Math.* **2**, 137–141 (1960)
3. Dekker, T.J.: A floating-point technique for extending the available precision. *Numer. Math.* **18**, 224–242 (1971)
4. Higham, N.J. *Accuracy and Stability of Numerical Algorithms*, 2nd edn. SIAM, Philadelphia (2002)
5. IEEE Standard for Floating-Point Arithmetic: Std 754-2019 (2019)
6. Jeannerod, C.-P., Rump, S.M.: Improved error bounds for inner products in floating-point arithmetic. *SIAM J. Matrix Anal. & Appl. (SIMAX)* **34**, 338–344 (2013)
7. Rump, S.M.: Error estimation of floating-point summation and dot product. *BIT Numer. Math.* **52**, 201–220 (2012)
8. Rump, S.M., Ogita, T., Oishi, S.: Accurate floating-point summation part I: faithful rounding. *SIAM J. Sci. Comput.* **31**, 189–224 (2008)

9. Strassen, V.: Gaussian elimination is not optimal. *Numer. Math.* **13**, 354–356 (1969)
10. Sylvester, J.J.: Thoughts on inverse orthogonal matrices, simultaneous sign successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers. *Philos. Mag.* **34**, 461–475 (1867)
11. Winograd, S.: On multiplication of 2×2 matrices. *Linear Algebra Its Appl.* **4**, 381–388 (1971)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.