6-2-2017

Fast Algorithms on Random Matrices and Structured Matrices

Liang Zhao
The City University of New York

Fast Algorithms on Random Matrices and Structured Matrices

by

Liang Zhao

A dissertation submitted to the Graduate Faculty in Mathematics in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

2017

This manuscript has been read and accepted for the Graduate Faculty in Mathematics in satisfaction of the dissertation requirements for the degree of Doctor of Philosophy.

(required signature)

Date — Chair of Examining Committee

(required signature)

Date — Executive Officer

Victor Y. Pan

Christina Sormani

Christina Zamfirescu

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Fast Algorithms on Random Matrices and Structured Matrices

by

Liang Zhao

Advisor: Victor Y. Pan

Randomization of matrix computations has become a hot research area in the big data era. Sampling with randomly generated matrices has enabled fast algorithms to perform well for some most fundamental problems of numerical algebra with probability close to 1. The dissertation develops a set of algorithms with random and structured matrices for the following applications:

1) We prove that using random sparse and structured sampling enables rank-r approximation of the average input matrix having numerical rank r.

2) We prove that Gaussian elimination with no pivoting (GENP) is numerically safe for the average nonsingular and well-conditioned matrix preprocessed with a nonsingular and well-conditioned f-Circulant or another

structured multiplier. This can be an attractive alternative to the customary Gaussian elimination with partial pivoting (GEPP).

3) By using structured matrices of a large family we compress large-scale neural networks while retaining high accuracy. The results of our extensive are in good accordance with those of our theoretical study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Structured matrices (such as Toeplitz, Hankel, Vandermonde, and Cauchy matrices, see Table 1) are also called Data Sparse because such an $n - by - n$ matrix can be represented with $O(n)$ parameters, that is, linear number in dimension n, rather than with n^2 entries.

Furthermore computations with such matrices require much fewer arithmetic operations than the same computations with general matrices. The most fundamental operations of matrix computations are multiplication of a matrix by a vector, multiplication of pairs of matrices, and the solution of a linear system of equations. When these operations are performed with general matrices, they require quadratic arithmetic time (for multiplication by a vector), or cubic time, but these bounds are decreased to nearly linear time in the case of structured matrices (see Table 1). This means dramatic saving of computer time and memory in modern large scale computations,

Table 1.0.1: FOUR CLASSES OF STRUCTURED MATRICES

TOEPLITZ $(t_{i-j})_{i,j=0}^{n-1}$ $\begin{pmatrix} t_0 & t_{-1} & \cdots & t_{1-n} \\ t_1 & t_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{pmatrix}$	HANKEL $(h_{i+j})_{i,j=0}^{n-1}$ $\begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_1 & h_2 & \iddots & h_n \\ \vdots & \iddots & \iddots & \vdots \\ h_{n-1} & h_n & \cdots & h_{2n-2} \end{pmatrix}$
VANDERMONDE $\left(s_i^j\right)_{i,j=0}^{n-1}$ $\begin{pmatrix} 1 & s_0 & \cdots & s_0^{n-1} \\ 1 & s_1 & \cdots & s_1^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & s_{n-1} & \cdots & s_{n-1}^{n-1} \end{pmatrix}$	CAUCHY $\left(\frac{1}{s_i - t_j}\right)_{i,j=0}^{n-1}$ $\begin{pmatrix} \frac{1}{s_0-t_0} & \cdots & \frac{1}{s_0-t_{n-1}} \\ \frac{1}{s_1-t_0} & \cdots & \frac{1}{s_1-t_{n-1}} \\ \vdots & & \vdots \\ \frac{1}{s_{n-1}-t_0} & \cdots & \frac{1}{s_{n-1}-t_{n-1}} \end{pmatrix}$

allowing to perform routinely computations that previously were unfeasible.

The intensive study in this field has produced efficient algorithms applied to various areas of computing, but this study is relatively recent, less than 50 years old, and for a while was mostly restricted to the important classes of Toeplitz and Hankel matrices, although already that study has involved

Unstructured	**Toeplitz/Henkel**	**Vandermonde**	**Cauchy**
Memory			
n^2	$2n$	n	$2n$
Matrix-Vector Multiplication			
$O(n^2)$	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$
Matrix Multiplication			
$O(n^{2.8})$	$O(n^2 \log n)$	$O(n^2 \log^2 n)$	$O(n^2 \log^2 n)$
Linear Solving			
$O(n^3)$	$O(n^2 \log n)$	$O(n^2 \log^2 n)$	$O(n^2 \log^2 n)$

nontrivial mathematical tools.

It turned out that a much wider class of structured matrices is highly important. New progress requires more advanced mathematical techniques, and the class of highly important applications to modern computing is growing fast. Together with the number and sophistication of mathematical techniques involved, the current study of this subject is flourishing.

In this thesis, we present some sample applications of sparse and structured matrices to some fundamental and highly popular problems of modern computations, where we demonstrate the power of this approach, show some advanced technical tools, and variety of its applications to modern computations for sciences, engineering, signal and image processing, and data mining and analysis.

In particular application of sparse and structured matrices enables us to enhance the power of randomization which is another recent technique extensively applied in order to advance fundamental matrix computations.

1.1 Low Rank Approximation of a Matrix

Low-rank approximation of a matrix has a variety of applications to the most fundamental matrix computations [HMT11] and numerous problems of data mining and analysis, "ranging from term document data to DNA SNP data"

[M11]. Classical solution algorithms use SVD or rank-revealing factorization, but the alternative solution by means of random sampling is numerically reliable, robust, and computationally and conceptually simple and has become highly and increasingly popular in the last decade (see [HMT11], [M11], [GL13, Section 10.4.5], [CMMa] for surveys and ample bibliography). Random sampling with Gaussian multipliers is likely to produce low-rank approximation with a probability close to 1,[1] but empirically the same algorithms work as efficiently with various random structured multipliers [HMT11, Sections 4.6 and 10.2].

Adequate formal support for this empirical evidence has been elusive so far, but based on our new insight we provide such a support, which opens new opportunities for enhancing the efficiency of this approach. Namely, an $n \times l$ Gaussian sampling is likely to yield rank-r approximation to any $m \times n$ matrix having numerical rank r if $l \geq r + 20$ [HMT11, Section 10.2], but by virtue of our dual theorem, sampling with any well-conditioned $n \times l$ multiplier of full rank l (we call such multipliers *promising*) supports rank-r approximation of the average $m \times n$ matrix having numerical rank r if $l \geq r$ and if, in the spirit of the Central Limit Theorem, the average matrix is

[1] Here and hereafter we refer to "standard Gaussian random" variables just as "*Gaussian*", and *"likely"* means "with a probability close to 1".

defined under the Gaussian probability distribution.

Our result should embolden search for new multipliers, to be generated and applied to an input matrix at a substantially lower computational cost than the structured multipliers of [HMT11], and we propose new families of such multipliers and some policies of managing the unlikely failure of sampling with such multipliers in Sections 2.5 and 2.6. We consistently succeeded in our extensive numerical tests of our recipes for benchmark inputs from [HMT11] as well as inputs generated according to [H02, Section 28.3].

By applying similar techniques we obtain similar progress for Gaussian elimination with no pivoting, block Gaussian elimination, and Least Squares Regression in [PZ17], [PSZa], and [PZa]; by extending these techniques we dramatically accelerate low-rank approximation of the average matrix in [PSZb].

1.2 Deep Neural Network Model Compression

Neural networks, especially large-scale deep neural networks, have made remarkable success in various applications such as computer vision, natural language processing, etc. [11][16]. However, large-scale neural networks are both memory-intensive and computation-intensive, thereby posing severe chal-

lenges when deploying those large-scale neural network models on memory-constrained and energy-constrained embedded devices. Various techniques have been proposed in order to overcome these limitations, by means of reducing the size of large-scale (deep) neural networks, for example, connection pruning [8][7], low rank approximation [6][9], sparsity regularization [17][13] etc.,

LDR Construction and LDR Neural Networks: Among those efforts, *low displacement rank (LDR) construction* is a type of structure- imposing technique for network model reduction and computational complexity reduction. By regularizing the weight matrices of neural networks using the format of LDR matrices (when weight matrices are square) or the composition of multiple LDR matrices (when weight matrices are non-square), a *strong structure* is naturally imposed to the construction of neural networks. Since an LDR matrix typically requires $O(n)$ independent parameters and supports fast matrix operation algorithms [14], an immense space for network model and computational complexity reduction can be enabled. Pioneering work in this direction [3][15] applied LDR matrices of specific classes, such as circulant matrices and Toeplitz matrices, for weight representation. Other types of LDR matrices exist such as Cauchy matrices, Vandermonde matrices, etc., as shown in Figure 1.

Figure 1.1: Examples of commonly used LDR (structured) matrices, i.e., circulant, Cauchy, Toeplitz, Hankel, and Vandermonde matrices.

Benefits of LDR Neural Networks: Compared with other types of network compression approaches, the LDR construction shows several unique advantages. First, unlike heuristic weight-pruning methods [8][7] that produce irregular pruned networks, the LDR construction approach always guarantees the strong structure of the trained network, thereby avoiding large overhead of its storage space and computation time incurred by the complicated indexing process. Second, as a "train from scratch" technique, LDR construction does not need extra re-training, and hence eliminates the additional complexity to the training process. Third, the reduction in space complexity and computational complexity by using the structured weight matrices are significant. Different from other approaches to network compression that can only provide a heuristic compression factor, the LDR construction enables the model reduction and computational complexity reduction in Big-O complexity: The storage requirement is reduced from $O(n^2)$ to $O(n)$, and the computational complexity can be reduced from $O(n^2)$ to $O(n \log n)$ or $O(n \log^2 n)$ because of the existence of fast matrix-vector multiplication algorithm [14][2] for LDR matrices. For example, when applying structured matrices to the fully-connected layers of AlexNet using ImageNet dataset [5],

the storage requirement can be reduced by a factor of more than 4,000 while incurring only negligible degradation in overall accuracy [3].

Motivation of This Work: Because of its inherent structure-imposing characteristic, convenient re-training-free training process and unique capability of simultaneous Big-O complexity reduction in storage and computation, LDR construction is a promising approach to achieve high compression ratio and high speedup for a broad category of network models. However, since imposing the structure to weight matrices results in substantial reduction of weight storage from $O(n^2)$ to $O(n)$, cautious researchers need to know whether the neural networks with LDR construction, to which we refer as LDR neural networks, consistently yields similar accuracy compared to uncompressed networks. Although [3][15] have already shown that using LDR construction still results in the same accuracy or allows only its minor degradation on various datasets, such as ImageNet [5], CIFAR [10] etc., the *theoretical analysis*, which can provide the mathematically solid proofs that the LDR neural networks can converge to the same "effectiveness" as the uncompressed neural networks, is still very necessary in order to promote the wide application of LDR neural networks for emerging and larger-scale applications.

Technical Preview and Contributions: To achieve these goals we

provide solid theoretical foundation for the ability of LDR neural networks to approximate an arbitrary continuous function, to estimate error bounds for function approximation, and to elaborate upon applications on shallow and deep neural networks, etc. More specifically, the main contributions of this chapter include:

- We prove the *universal approximation property* for LDR neural networks, which states that the LDR neural networks can approximate an arbitrary continuous function with arbitrary accuracy given enough parameters/neurons. In other words, the LDR neural network has the same "effectiveness" as the classical neural networks that work without compression. This property serves as the theoretical foundation of the potential broad applications of LDR neural networks.

- We show that, for LDR matrices defined by $O(n)$ parameters, the corresponding LDR neural networks are still capable of achieving integrated squared error of order $O(1/n)$, which is identical to the error bound of neural networks based on unstructured weight matrices. Thus there is essentially no loss for restricting to the weight matrices to LDR matrices.

- We develop a universal training process for LDR neural networks with

computational complexity reduction compared with backward propagation process for classical neural networks. The proposed algorithm is the generalization of the training process in [3][15] that restricts the structure of weight matrices to circulant matrices or Toeplitz matrices.

1.3 Gaussian Elimination without Pivoting

The history of Gaussian elimination can be traced back some 2000 years [G11]. Its modern version, called *Gaussian elimination with partial pivoting* (hereafter we use the acronym *GEPP*), is performed routinely, millions times per day around the world, being a cornerstone for computations in linear algebra. For some samples of extensive and intensive applications of GEPP to Sciences, Technology, and Signal and Image Processing, the survey [DDF14] refers to fusion reactor modeling, aircraft design, acoustic scattering, antenna design, and radar cross-section studies and then recalls that, e.g., in simulating fusion reactors, GEPP is applied to solving dense linear systems of equations with over half a million unknowns.

For an $n \times n$ input matrix, partial pivoting, that is, row interchange, involves only $(n-1)n/2$ comparisons, versus about $\frac{2}{3}n^3$ arithmetic operations involved into elimination. Progress in computer technology, however, has made partial pivoting the bottleneck of Gaussian elimination. Here is a

relevant citation from [BCD14]: "The traditional metric for the efficiency of a numerical algorithm has been the number of arithmetic operations it performs. Technological trends have long been reducing the time to perform an arithmetic operation, so it is no longer the bottleneck in many algorithms; rather, communication, or moving data, is the bottleneck."

Pivoting is communication intensive and in modern computer environment takes quite a heavy toll: it interrupts the stream of arithmetic operations with foreign operations of comparison, involves book-keeping, compromises data locality, impedes parallelization of the computations, and increases communication overhead and data dependence.

According to [BDHT13], "pivoting can represent more than 40% of the global factorization time for small matrices, and although the overhead decreases with the size of the matrix, it still represents 17% for a matrix of size 10,000". Because of the heavy use of GEPP, even its limited improvement is valuable.

Gaussian elimination with no pivoting (hereafter we keep using the acronym *GENP*) is an attractive alternative to the customary GEPP, but it can fail or produce a corrupted output more readily than GEPP. To specify this, call an input matrix *unsafe* and *numerically unsafe* for Gaussian elimination with or without pivoting if this algorithm applied to that matrix runs into a division

by 0 or into numerical problems, respectively. A singular input matrix is unsafe for both GENP and GEPP, and an ill-conditioned matrix is numerically unsafe for both of them (see the end of Section **??** on the concepts of ill- and well-conditioned matrices). Nonsingular matrices are safe for GEPP, and a very small fraction of them is unsafe for GENP. The matrices of only a small subclass of nonsingular and well-conditioned matrices are numerically unsafe for GENP, and even more rarely they are numerically unsafe for GEPP.

Since [W61], numerical safety or stability of Gaussian elimination has been tied to the growth factor $\rho = \max_{i,j} |u_{ij}| / \max_{i,j} |a_{ij}|$ in the PLUP' factorization (with or without pivoting) of a matrix $A = (a_{ij})_{i,j} = PLUP'$ where $L = (l_{ij})_{i,j}$ and $U = (u_{ij})_{i,j}$ are lower and upper triangular factors, respectively, and P and P' are permutation matrices. In the case of GENP, both of P and P' turn into the identity matrix, $P = P' = I_n$. In the case of GEPP, $P' = I_n$, and we can choose any permutation matrix P, defining row interchange. Both P and P' are our unrestricted choice in Gaussian elimination with *complete pivoting*, hereafter referred to as *GECP*, that is, in GECP we can interchange any pair of rows and columns.

For the worst case input, ρ is unbounded in the case of GENP, is as large as 2^{n-1} in the case of GEPP, and is in $O(n^{(1+\log(\sqrt{n}))/2})$ in the case of GECP. Nevertheless GEPP has been universally preferred by the user in most cases.

Its average growth factor is only $n^{2/3}$, which is just slightly larger than $n^{1/2}$ for GECP, and its pivoting is simpler than that of GECP, whereas GENP is considered numerically unsafe in practice and is little used, even though its average growth factor is just $n^{3/2}$, that is, only marginally larger than in the case of GEPP (cf. [TS90], [YC97]).

Block matrix algorithms are highly important resource for enhancing the efficiency of matrix algorithms [GL13], but *block Gaussian elimination* (hereafter we keep using the acronym *BGE*) is impeded because pivoting cannot helps us to avoid numerical stability problems for this algorithm. We can see in Section 4.1, however, that BCG is safe (resp., numerically safe) for any input matrix for which GENP is safe (resp., numerically safe). Thus we have yet another major motivation for studying GENP.

1.3.1 Preprocessing versus pivoting

Preprocessing $A \to FA$, $A \to AH$, and $A \to FAH$, for nonsingular matrices F and H, is a natural resource for supporting GENP and BGE because $A^{-1} = (FA)^{-1}F$, $A^{-1} = G(AG)^{-1}$, and $A^{-1} = G(FAG)^{-1}F$ and because empirically the matrices FA, AH, and FAH tend to be safe and numerically safe for GENP and BGE when the matrix A is nonsingular and well-conditioned and when the matrices F and H are random. The above observations and

empirical data have recently convinced some leading experts in numerical matrix computations to implement GENP with randomized preprocessing (see [BBD12], [BDHT13], [BBBDD14], and [BLR15]). The resulting algorithm noticeably improves GEPP, even though preprocessing in the cited papers relies on the ad hoc multipliers from unpublished Technical Reports of 1995 by Parker and by Parker and Pierce. This leaves us with the challenge of finding even more efficient multipliers and preprocessing policies.

In this chapter we reexamine the state of the art, the role of randomization, and the benefits of using sparse and structured multipliers. We supply formal probabilistic analysis based on our nonstandard and more general techniques of analysis. Our techniques are much simpler (the proof of our basic Theorem 4.2.2 occupies just ten lines), but much more general. Our results are consistent with the ones based on estimating the growth factor of LU factorization, but unlike them cover BGE as well and are extended readily to the study of alternative methods of preprocessing of Gaussian elimination by means of augmentation and additive preprocessing as well as to the study of the highly important algorithms for low-rank approximation of a matrix.

We provide some new insights into the subject and new recipes for preprocessing such as *incomplete GENP and BGE* in Remark 4.2.2 and *testing multipliers by action, successively or concurrently* in Section 4.2.4. We spec-

ify some useful concepts (such as *strongly well-conditioned matrices* in Section 4.2.1, *universal* and *random universal sets of multipliers* in the abstract and Section 4.2.4, *primal and dual randomization* in Section 4.2.4, *the dilemmas of random versus fixed preprocessing* in Section 4.2.4 and of *random versus fixed sampling* in Section 2.4.3, and *subcirculant multipliers* in Remark C.0.3. We also point out some promising basic choices of multipliers and their combinations in Section 4.3. Some of the basic choices are new, some go back to the 1990s (c.f. [BP94, Section 2.13], entitled "Regularization of a Matrix via Preconditioning with Randomization") or extend our more recent study of randomized matrix algorithms, including GENP and BGE with preprocessing (cf. [PGMQ, Section 12.2], [PIMR10], [PQ10], [PQ12], [PQY14], [PQY15], and [PQZ13]).

1.3.2 Extensions of our study and its empirical support

All our results and recipes for GENP also apply to BGE. In Section 4.2.3 we point out a sample amelioration of GENP and BGE. In Section **??** we show that our alternatives of randomized augmentation and additive preprocessing imply some benefits for solving linear systems of equations by means of randomized GENP, BGE, and other methods.

In Section 2.4.3 we follow the lead of [PQY15] and extend our study

of GENP and BGE to low-rank approximation of a matrix by means of random sampling. This highly popular subject has been studied intensively and successfully in the last decade. Application areas include some of the most fundamental matrix computations [HMT11] as well as "data analysis, ranging from term document data to DNA SNP data" [M11]. We refer the reader to [HMT11], [M11], and [GL13, Section 10.4.5] for surveys and ample bibliography, and to [GZT97], [GTZ97], [T00], [FKV98/04], and [DKM06] for sample early works. Technical similarity of this approach to preprocessing for GENP and BGE (apparently never observed until [PQY15]) enables us to provide new insights into this subject and some novel recipes. We also discuss the related topic of low-rank representation of a matrix.

The results of our formal analysis are in good accordance with the data from our tests, presented in Section 4.5. In particular GENP with our new preprocessing (unlike the case of the known ones in [BDHT13] and [BBBDD14]) produced accurate outputs even with no iterative refinement.

Chapter 2

Low-rank Approximation: New Insights

2.1 Some Basic Definitions

- Typically we use the concepts "large", "small", "near", "close", "approximate", "ill-conditioned" and "well-conditioned" quantified in the context, but we specify them quantitatively as needed.

- Hereafter "$\ll$" means "much less than"; *"flop"* stands for "floating point arithmetic operation".

- I_s is the $s \times s$ identity matrix. $O_{k,l}$ is the $k \times l$ matrix filled with zeros. $\mathbf{o}$ is a vector filled with zeros.

- $(B_1 \mid B_2 \mid \ldots \mid B_h)$ denotes a $1 \times h$ block matrix with h blocks $B_1, B_2, \ldots, B_h$.

- $\mathrm{diag}(B_1, B_2, \ldots, B_h)$ denotes a $h \times h$ block diagonal matrix with h diagonal blocks $B_1, B_2, \ldots, B_h$.

- $\mathrm{rank}(W)$, $\mathrm{nrank}(W)$, and $||W||$ denote the *rank*, the *numerical rank*, and the *spectral norm* of a matrix W, respectively. $||W||_F$ denotes its *Frobenius norm*.

- W^T and W^H denote its transpose and Hermitian transpose, respectively.

- An $m \times n$ matrix W is *unitary* (in the real case also *orthogonal*) if $W^H W = I_n$ or if $W W^H = I_m$.

- $W = S_{W,\rho} \Sigma_{W,\rho} T_{W,\rho}^T$ is *compact SVD* of a matrix W of rank ρ with the unitary matrices of its singular vectors $S_{W,\rho}$ and $T_{W,\rho}$ and the diagonal matrix $\Sigma_{W,\rho} = \mathrm{diag}(\sigma_j(W))_{j=1}^{\rho}$ of its singular values.

- $W^+ = T_{W,\rho} \Sigma_{W,\rho}^{-1} S_{W,\rho}^T$ is it *Moore–Penrose pseudo inverse*. [Recall that $||W^+|| = \frac{1}{\sigma_\rho(W)}$.]

- $\kappa(W) = \sigma_1(W)/\sigma_\rho(W) = ||W||\ ||W^+|| \geq 1$ denotes the *condition number* of a matrix W. A matrix is called *ill-conditioned* if its condition number is large in context and is called *well-conditioned* if this number $\kappa(W)$ is reasonably bounded.

2.2 Randomized Matrix Computations (Definitions, Basic Results)

- The acronym "*i.i.d.*" stands for "independent identically distributed", and we refer to "standard Gaussian random" variables just as "*Gaussian*".

- We call an $m \times n$ matrix *Gaussian* and denote it $G_{m,n}$ if all its entries are i.i.d. Gaussian variables.

- $\mathcal{G}^{m\times n}$, $\mathbb{R}^{m\times n}$, and $\mathbb{C}^{m\times n}$ denote the classes of $m \times n$ Gaussian, real, or complex matrices, respectively.

- $\mathcal{G}_{m,n,r}$, $\mathbb{R}_{m,n,r}$, and $\mathbb{C}_{m,n,r}$, for $1 \le r \le \min\{m, n\}$, denote the classes of $m \times n$ matrices $M = UV$ (of rank at most r) where $U \in \mathbb{C}^{m\times r}$ and $V \in \mathbb{C}^{r\times n}$ are Gaussian, real, and complex, respectively.

- If $U \in \mathcal{G}^{m\times r}$ and $V \in \mathcal{G}^{r\times n}$, then we call $M = UV$ an $m \times n$ *factor-Gaussian matrix of expected rank* r. (In this case the matrices U, V and M have rank r with probability 1 by virtue of Theorem A.1.1.)

Next we recall some basic results for randomized matrix computations (see proofs in [PSZa, Section 3]).

Theorem 2.2.1. *Suppose that A is an $m \times n$ matrix of full rank $k = \min\{m, n\}$, F and H are $r \times m$ and $n \times r$ matrices, respectively, for $r \leq k$, and the entries of these two matrices are non-constant linear combinations of finitely many i.i.d. random variables $v_1, \dots, v_h$.*

Then the matrices F, FA, H, and AH have full rank r

(i) with probability 1 if $v_1, \dots, v_h$ are Gaussian variables and

(ii) with a probability at least $1 - r/|\mathcal{S}|$ if they are random variables sampled under the uniform probability distribution from a finite set $\mathcal{S}$ having cardinality $|\mathcal{S}|$.

Lemma 2.2.1. *(*Orthogonal invariance of a Gaussian matrix.*) Suppose that k, m, and n are three positive integers, $k \geq \min\{m, n\}$, G is an $m \times n$ Gaussian matrix, and S and T are $k \times m$ and $n \times k$ orthogonal matrices, respectively. Then SG and GT are Gaussian matrices.*

We state the following estimates for real matrices, but similar estimates in the case of complex matrices can be found, e.g., in [CD05]:

Definition 2.2.1. Norms of random matrices and expected value of a random variable. *Write $\nu_{m,n} = ||G||$, $\nu^+_{m,n} = ||G^+||$, and $\nu^+_{m,n,F} = ||G^+||_F$, for a Gaussian $m \times n$ matrix G, and write $\mathbb{E}(v)$ for the expected value of a random variable v. ($\nu_{m,n} = \nu_{n,m}$, $\nu^+_{m,n} = \nu^+_{n,m}$, and $\nu_{F,m,n} = \nu_{F,n,m}$, for all pairs of m*

and n.)

Theorem 2.2.2. *Suppose that m and n are positive integers, $t \geq 0$, and $h = \max\{m, n\}$. Then*

(i) Probability$\{\nu_{m,n} > t + \sqrt{m} + \sqrt{n}\} \leq \exp(-t^2/2)$ *and*

(ii) $\mathbb{E}(\nu_{m,n}) \leq \sqrt{m} + \sqrt{n}$.

Theorem 2.2.3. *Let $\Gamma(x) = \int_0^\infty \exp(-t)t^{x-1}dt$ denote the Gamma function and let $x > 0$. Then*

(i) Probability $\{\nu_{m,n}^+ \geq m/x^2\} < \frac{x^{m-n+1}}{\Gamma(m-n+2)}$ *for $m \geq n \geq 2$,*

(ii) Probability $\{\nu_{n,n}^+ \geq x\} \leq 2.35\sqrt{n}/x$ *for $n \geq 2$,*

(iii) $\mathbb{E}(\nu_{m,n}^+) \leq e\sqrt{m}/|m-n|$, *provided that $m \neq n$ and $e = 2.71828\ldots$.*

The probabilistic upper bounds of Theorem A.2.2 on $\nu_{m,n}^+$ are reasonable already in the case of square matrices, that is, where $m = n$, but are strengthened very fast as the difference $|m - n|$ grows from 1. Theorems A.2.1 and A.2.2 combined imply that an $m \times n$ Gaussian matrix is well-conditioned unless the integer $m + n$ is large or the integer $|m - n|$ is close to 0. With some grain of salt we can still consider such a matrix well-conditioned even where the integer $|m - n|$ is small or vanishes provided that the integer m is not large.

2.3 The Basic Algorithm, Primal and Dual Support

A matrix M can be represented (respectively, approximated) by a product UV of two matrices $U \in \mathbb{C}^{m\times r}$ and $V \in \mathbb{C}^{r\times n}$ if and only if $r \geq \text{rank}(M)$ (respectively, $r \geq \text{nrank}(M)$), and our goal is the computation of such a representation or approximation.

We rely on the following basic algorithm. for the *fixed rank problem* where we assume that we are given $r = \text{nrank}(M)$ or $r = \text{rank}(M)$. Otherwise we could have computed it by means of binary search based on recursive application of the algorithm or proceeded as in our Algorithm 2.6.1 of Section 2.6.2.

Algorithm 2.3.1. Range Finder (See [HMT11, Algorithms 4.1 and 4.2]).

INPUT: *An $m \times n$ matrix M, a tolerance $\tau \geq 0$, and $r = \text{nrank}(M)$ such that $0 < r \ll \min\{m, n\}$.*

INITIALIZATION: *Fix an integer l such that $r \leq l \ll \min\{m, n\}$ and an $n \times l$ matrix B.*

COMPUTATIONS: *1. Compute the $m \times l$ matrix MB. Remove its columns that have small norms.*

2. *Orthogonalize its remaining columns (cf. [GL13, Theorem 5.2.3]), compute and output the resulting* $m \times \bar{l}$ *matrix* $U = U(MB)$ *where* $\bar{l} \le l$.

3. *Estimate the error norm* $\Delta = ||\tilde{M} - M||$ *for* $\tilde{M} = UU^T M$.

   If $\Delta \le \tau$, *output SUCCESS; otherwise FAILURE.*

We call a pair of matrices $[M; B]$ *hard* for Algorithm 2.3.1 if the algorithm fails for that pair. For any fixed B_0 and any fixed M_0 we can readily define hard pairs $[M; B_0]$ and $[M_0; B]$, but next we prove that such pairs $[M; B_0]$ and $[M_0; B]$ are unlikely to be hard for Algorithm 2.3.1 if M and B are Gaussian matrices.

Theorem 2.3.1. *Let Algorithm 2.3.1 be applied with a Gaussian multiplier* $B \in \mathcal{G}^{n \times l}$. *Then*

(i) $\tilde{M} = M$ *with probability 1 if* $l \ge r = \text{rank}(M)$ *(cf. Theorem 2.4.1) and*

(ii) it is likely that $\tilde{M} \approx M$ *if* $\text{nrank}(M) = r \le l$, *and the probability that* $\tilde{M} \approx M$ *approaches 1 fast as* l *increases from* $r + 1$ *(cf. Theorem 4.4.1).*

An $n \times l$ Gaussian matrix B is defined by its nl random entries, and can be pre-multiplied by a vector in $(2n - 1)l$ flops, but an $n \times l$ structured

matrix B of *subsample random Fourier or Hadamard transform*[1] is defined by by $n+l$ random variables and can be pre-multiplied by a vector by using $O(n\log(l))$ flops (see [HMT11, Sections 4.6 and 11], [M11, Section 3.1], and [T11]). SRFT and SRHT multipliers B are *universal*, like Gaussian ones: Algorithm 2.3.1 applied with such a multiplier is likely to approximate closely a matrix M having numerical rank at most r, although the estimated failure probability $3\exp(-p)$, for $p=l-r\geq 4$ with Gaussian multipliers increases to order of $1/l$ in the case of SRFT and SRHT multipliers (cf. [HMT11, Theorems 10.9 and 11.1], [M11, Section 5.3.2], and [T11]).

Empirically Algorithm 2.3.1 with SRFT and SRHT multipliers fails very rarely even for $l=r+20$, although for some special input matrices M it is likely to fail if $l=o(r\log(r))$ (cf. [HMT11, Remark 11.2] or [M11, Section 5.3.2]). Researchers have consistently observed similar empirical behavior with various other multipliers (see [HMT11], [M11], [W14], [PQY15], and the references therein), but so far no adequate formal support for that empirical observation has appeared in the huge bibliography on this highly popular subject. Our *Dual* Theorem 2.3.2 below, however, provides such a formal support. The theorem reverses the assumptions of our *Primal* Theorem 4.2.1 that a multiplier B is Gaussian, while a matrix M is fixed.

[1]Hereafter we use the acronyms *SRFT* and *SRHT*.

Theorem 2.3.2. *Let $M - E \in \mathcal{G}_{m,n,r}$ and $||E||_2 \approx 0$ (in which case we have* $\text{nrank}(M) \le r$ *and, with a probability close to 1,* $\text{nrank}(M) = r$*). Furthermore let $B \in \mathbb{R}^{n \times l}$ and* $\text{nrank}(B) = l$*. Then*

(i) Algorithm 2.3.1 outputs a rank-r representation of a matrix M with probability 1 if $E = 0$ and if $l = r$ (cf. Theorem 2.4.1), and

(ii) it outputs a rank-l approximation of that matrix with a probability close to 1 if $l \ge r$ and approaching 1 fast as the integer l increases from $r + 1$ (cf. Theorem 2.4.4).

Claim (ii) implies that *Algorithm 2.3.1 succeeds for the average input matrix M that has a small numerical rank $r \le l$* (and thus in a sense to most of such matrices) if the multiplier B is a well-conditioned matrix of full rank and if the average matrix is defined under the Gaussian probability distribution. The former provision, that $\text{nrank}(B) = l$, is natural for otherwise we could have replaced the multiplier B by an $n \times l_-$ matrix for some integer $l_- < l$. The latter provision is customary in view of the Central Limit Theorem.

2.4 Proof of the Primal and Dual Theorems

2.4.1 Computation of low-rank representation: proof

Hereafter $\mathcal{R}(W)$ denotes the range of W. Our next theorem implies claims (i) of Theorems 4.2.1 and 2.3.2.

Theorem 2.4.1. *(i) For an $m \times n$ input matrix M of rank $r \le n \le m$, its rank-r representation is given by the products $R(R^TR)^{-1}R^TM = U(R)U(R)^TM$ provided that R is an $n \times r$ matrix such that $\mathcal{R}(R) = \mathcal{R}(M)$ and that $U(R)$ is a matrix obtained by means of column orthogonalization of R.*

(ii) $\mathcal{R}(R) = \mathcal{R}(M)$, for $R = MB$ and an $n \times r$ matrix B, with probability 1 *if B is Gaussian, and*

(iii) with a probability at least $1 - r/|S|$ if an $n \times r$ matrix B has i.i.d. random entries sampled uniformly from a finite set $\mathcal{S}$ of cardinality $|S|$.

Proof. Readily verify claim (i) (cf. [S98, pages 60–61]). Then note that $\mathcal{R}(MB) \subseteq \mathcal{R}(M)$, for an $n \times r$ multiplier B. Hence $\mathcal{R}(MB) = \mathcal{R}(M)$ if and only if $\text{rank}(MB) = r$, and therefore if and only if a multiplier B has full rank r. Now claims (ii) and (iii) follow from Theorem A.1.1. □

2.4.2 Analysis of low-rank approximation: a basic step

In our proofs of Theorems 2.6.1, 4.2.1, and 2.3.2 we rely on the following lemma and theorem.

Lemma 2.4.1. *(Cf. [GL13, Theorem 2.4.8].) For an integer r and an $m \times n$ matrix M where $m \ge n > r > 0$, set to 0 the singular values $\sigma_j(M)$, for $j > r$, let M_r denote the resulting matrix, which is a closest rank-r approximation*

of M, and write $M = M_r + E$. Then

$$||E|| = \sigma_{r+1}(M) \text{ and } ||E||_F^2 = \sum_{j=r+1}^{n} \sigma_j^2 \le \sigma_{r+1}(M)^2(n-r).$$

Theorem 2.4.2. The error norm in terms of $||(M_rB)^+||$. *Assume dealing with the matrices M and $\tilde{M}$ of Algorithm 2.3.1, M_r and E of Lemma 2.4.1, and $B \in \mathbb{C}^{n\times l}$ of rank l. Let* $\text{rank}(M_rB) = r$ *and write $E' = EB$ and $\Delta = ||\tilde{M} - M||$. Then*

$$||E'||_F \le ||B||_F \; ||E||_F \le ||B||_F \; \sigma_{r+1}(M) \; \sqrt{n-r}, \tag{2.4.1}$$

$$|\Delta - \sigma_{r+1}(M)| \le \sqrt{8} \; ||(M_rB)^+|| \; ||E'||_F + O(||E'||_F^2). \tag{2.4.2}$$

Proof. Lemma 2.4.1 implies bound (2.4.1).

Next apply claim (i) of Theorem 2.4.1 for matrix M_r replacing M, recall that $\text{rank}(M_rB) = l$, and obtain

$$U(M_rB)U(M_rB)^TM_r = M_r, \; \mathcal{R}(U(M_rB)) = \mathcal{R}(M_rB) = \mathcal{R}(M_r).$$

Furthermore $U(M_rB)^T(M - M_r) = O_{n,n}$. Hence$U(M_rB)U(M_rB)^TM = U(M_rB)U(M_rB)^TM_r = M_r$.

Consequently, $M - U(M_rB)U(M_rB)^TM = M - M_r = E$, and so (cf. Lemma 2.4.1)

$$||M - U(M_rB)U(M_rB)^TM|| = \sigma_{r+1}(M). \tag{2.4.3}$$

Apply [PQY15, Corollary C.1], for $A = M_rB$ and E replaced by $E' = (M - M_r)B$, and obtain

$$\begin{aligned} &||U(MB)U(MB)^T - U(M_rB)U(M_rB)^T|| \\ \leq &\sqrt{8}||(M_rB)^+||\ ||E'||_F + O(||E'||_F^2). \end{aligned} \tag{2.4.4}$$

Combine this bound with (2.4.3) and obtain (2.4.2). □

By combining bounds (2.4.1) and (2.4.2) obtain

$$|\Delta - \sigma_{r+1}(M)| \leq \sqrt{8(n-r)}\ \sigma_{r+1}(M)\ ||B||_F||(M_rB)^+|| + O(\sigma_{r+1}^2(M)). \tag{2.4.5}$$

In our applications the value $\sqrt{8(n-r)}\ \sigma_{r+1}(M)||B||_F$ is small, and so the value $|\Delta - \sigma_{r+1}(M)|$ is small unless the norm $||(M_rB)^+||$ is large.

2.4.3 Detailed estimates for primal and dual low-rank approximation

The following theorem, proven in the next subsection, bounds the approximation errors and the probability of success of Algorithm 2.3.1 for $B \in \mathcal{G}^{n\times l}$. Together these bounds imply claim (ii) of Theorem 4.2.1.

Theorem 2.4.3. *Suppose that Algorithm 2.3.1 has been applied to an $m \times n$ matrix M having numerical rank r and that the multiplier B is an $n \times l$ Gaussian matrix.*

(i) Then the algorithm outputs an approximation $\tilde{M}$ *of a matrix* M *by a rank-*l *matrix within the error norm bound* Δ *such that* $|\Delta - \sigma_{r+1}(M)| \le f\sigma_{r+1}(M)/\sigma_r(M) + O(\sigma^2_{r+1}(M))$ *where* $f = \sqrt{8(n-r)}\,\nu_{F,n,l}\nu^+_{r,l}$ *and* $\nu_{F,n,l}$ *and* $\nu^+_{r,l}$ *are random variables of Definition 2.2.1.*

(ii) $\mathbb{E}(f) < (1 + \sqrt{n} + \sqrt{l})\frac{e}{p}\,\sqrt{8(n-r)rl}$, *for* $p = l - r > 0$ *and* $e = 2.71828\ldots$.

Remark 2.4.1. $\sigma_{r+1}(M)$ *is the optimal upper bound on the norm* Δ, *and the expected value* $\mathbb{E}(f)$ *is reasonably small even for* $p = 1$. *If* $p = 0$, *then* $\mathbb{E}(f)$ *is not defined, but the random variable* Δ *estimated in Theorem 4.4.1 is still likely to be reasonably close to* $\sigma_{r+1}(M)$ *(cf. claim (ii) of Theorem A.2.2).*

In Section 2.4.5 we prove the following elaboration upon dual Theorem 2.3.2.

Theorem 2.4.4. *Suppose that Algorithm 2.3.1, applied to a small-norm perturbation of an* $m \times n$ *factor-Gaussian matrix with expected rank* $r < m$, *uses an* $n \times l$ *multiplier* B *such that* $\mathrm{nrank}(B) = l$ *and* $l \ge r$.

(i) Then the algorithm outputs a rank-l *matrix* $\tilde{M}$ *that approximates the matrix* M *within the error norm bound* Δ *such that* $|\Delta - \sigma_{r+1}(M)| \le f_d\sigma_{r+1}(M) + O(\sigma^2_{r+1}(M))$, *where* $f_d = \sqrt{8(n-r)l}\;\nu^+_{r,l}\nu^+_{m,r}\kappa(B)$, $\kappa(B) =$

$||B||\ ||B^+||$, *and* $\nu^+_{m,r}$ *and* $\nu^+_{r,l}$ *are random variables of Definition 2.2.1.*

(ii) $\mathbb{E}(f_d) < e^2\sqrt{8(n-r)l}\ \kappa(B)\frac{r}{(m-r)p}$, *for* $p = l - r > 0$ *and* $e = 2.71828\ldots$.

Remark 2.4.2. *The expected value* $\mathbb{E}(\nu^+_{m,r}) = \frac{e\sqrt{r}}{m-r}$ *converges to 0 as* $m \to \infty$ *provided that* $r \ll m$. *Consequently the expected value* $\mathbb{E}(\Delta) = \sigma_{r+1}(M)\mathbb{E}(f_d)$ *converges to the optimal value* $\sigma_{r+1}(M)$ *as* $\frac{m}{r\sqrt{nl}} \to \infty$ *provided that* B *is a well-conditioned matrix of full rank and that* $1 \le r < l \ll n \le m$.

Remark 2.4.3. *[HMT11, Theorem 10.8] also estimates the norm* Δ, *but our estimate in Theorem 4.4.1, in terms of random variables* $\nu_{F,n,l}$ *and* $\nu^+_{r,l}$, *is more compact, and our proof is distinct and shorter than one in [HMT11], which involves the proofs of [HMT11, Theorems 9.1, 10.4 and 10.6].*

Remark 2.4.4. *By virtue of Theorems A.1.1,* $\mathrm{rank}(M_rB) = r$ *with probability 1 if the matrix* B *or* M *is Gaussian, (as in Theorems 4.4.1 and 2.4.4), and we deduced (2.4.5) from the equation* $\mathrm{rank}(M_rB) = r$.

Remark 2.4.5. The Power Scheme of increasing the output accuracy of Algorithm 2.3.1. *See [RST09], [HMST11]. Define the Power Iterations* $M_i = (M^TM)^iM$, $i = 1, 2, \ldots$. *Then* $\sigma_j(M_i) = (\sigma_j(M))^{2i+1}$ *for all* i *and* j *[HMT11, equation (4.5)]. Therefore, at a reasonable computational cost,*

one can dramatically decrease the ratio $\frac{\sigma_{r+1}(M)}{\sigma_r(M)}$ *and thus decrease the bounds of Theorems 4.4.1 and 2.4.4 accordingly.*

In the next two subsections we deduce reasonable bounds on the norm $||(M_rB)^+||$ in both cases where M is a fixed matrix and B is a Gaussian matrix and where B is fixed matrix and M is a factor Gaussian matrix (cf. Theorems 2.4.5 and 2.4.6). The bounds imply Theorems 4.4.1 and 2.4.4.

2.4.4 Primal theorem: completion of the proof

Theorem 2.4.5. *For* $M \in \mathbb{R}^{m\times n}$, $B \in \mathcal{G}^{m\times l}$, *and* $\nu^+_{r,l}$ *of Definition 2.2.1, it holds that*

$$||(M_rB)^+|| \le \nu^+_{r,l}/\sigma_r(M). \tag{2.4.6}$$

Proof. Let $M_r = S_r\Sigma_r T_r^T$ be compact SVD. By applying Lemma A.2.1, deduce that T_r^TB is a $r \times l$ Gaussian matrix. Denote it $G_{r,l}$ and obtain $M_rB = S_r\Sigma_r T_r^TB = S_r\Sigma_r G_{r,l}$.

Write $H = \Sigma_r G_{r,l}$ and let $H = S_H\Sigma_H T_H^T$ be compact SVD where S_H is a $r \times r$ unitary matrix.

It follows that $S = S_rS_H$ is an $m \times r$ unitary matrix.

Hence $M_rB = S\Sigma_H T_H^T$ and $(M_rB)^+ = T_H(\Sigma_H)^+S^T$ are compact SVDs of the matrices M_rB and $(M_rB)^+$, respectively. Therefore $||(M_rB)^+|| = ||(\Sigma_H)^+|| = ||(\Sigma_r G_{r,l})^+|| \le ||G^+_{r,l}||\ ||\Sigma_r^{-1}||$.

Substitute $||G^+_{r,l}|| = \nu^+_{r,l}$ and $||\Sigma_r^{-1}|| = 1/\sigma_r(M)$ and obtain the theorem.

□

Combine bounds (2.4.5), (2.4.6), and equation $||B||_F = \nu_{F,n,l}$ and obtain claim (i) of Theorem 4.4.1. Combine that claim with claims (ii) of Theorem A.2.1 and (iii) of Theorem A.2.2 and obtain claim (ii) of Theorem 4.4.1.

2.4.5 Dual theorem: completion of the proof

Theorem 2.4.6. *Suppose that* $U \in \mathbb{R}^{m\times r}$, $V \in \mathcal{G}^{r\times n}$, $\text{rank}(U) = r \le \min\{m, n\}$, $M = UV$, *and* B *is a well-conditioned* $n \times l$ *matrix of full rank* l *such that* $m \ge n > l \ge r$ *and* $||B||_F = 1$. *Then*

$$||(MB)^+|| \le ||B^+||\ \nu^+_{r,l}\ ||U^+||. \tag{2.4.7}$$

If in addition $U \in \mathcal{G}^{m\times r}$, *that is, if* M *is an* $m \times n$ *factor-Gaussian matrix with expected rank* r, *then*

$$||(MB)^+|| \le ||B^+||\nu^+_{m,r}\ \nu^+_{r,l}.$$

Proof. Combine compact SVDs $U = S_U\Sigma_U T_U^T$ and $B = S_B\Sigma_B T_B^T$ and obtain $UVB = S_U\Sigma_U T_U^T V S_B\Sigma_B T_B^T$. Here U, V, B, S_U, Σ_U, T_U, S_B, Σ_B, and T_B are matrices of the sizes $m \times r$, $r \times n$, $n \times l$, $m \times r$, $r \times r$, $r \times r$, $n \times l$, $l \times l$, and $l \times l$, respectively.

Now observe that $G_{r,l} = T_U^T V S_B$ is a $r \times l$ Gaussian matrix, by virtue of Lemma A.2.1 (since V is a Gaussian matrix). Therefore $UVB = S_U F T_B^T$, for $F = \Sigma_U G_{r,l} \Sigma_B$.

Let $F = S_F \Sigma_F T_F^T$ denote compact SVD where $\Sigma_F = \mathrm{diag}(\sigma_j(F))_{j=1}^r$ and S_F and T_F^T are unitary matrices of sizes $r \times r$ and $r \times l$, respectively.

Both products $S_U S_F \in \mathbb{R}^{m \times r}$ and $T_F^T T_B^T \in \mathbb{R}^{r \times l}$ are unitary matrices, and we obtain compact SVD $MB = UVB = S_{MB} \Sigma_{MB} T_{MB}^T$ where $S_{MB} = S_U S_F$, $\Sigma_{MB} = \Sigma_F$, and $T_{MB}^T = T_F^T T_B^T$. Therefore

$$||(MB)^+|| = ||\Sigma_{MB}^+|| = ||\Sigma_F^+|| = ||F^+||.$$

Σ_B and Σ_V are square nonsingular diagonal matrices, and so

$$F^+ = \Sigma_B^{-1} G_{r,l}^+ \Sigma_U^{-1}$$

and

$$||(MB)^+|| = ||F^+|| \le ||\Sigma_B^{-1}||\ ||G_{r,l}^+||\ ||\Sigma_U^{-1}|| = ||B^+||\nu_{r,l}^+||U^+||. \qquad \square$$

Combine (2.4.5), (2.4.7) and $||B||_F \le ||B||\ \sqrt{l}$ and obtain Theorem 2.4.4 provided that M is a factor-Gaussian matrix UV with expected rank r. By applying [S98, Corollary 1.4.19] for $P = -C^{-1}E$ extend the results to the case where $M = UV + E$ and the norm $||E||$ is small, completing the proof of Theorem 2.4.4.

Remark 2.4.6. *If* $U \in \mathcal{G}^{m\times r}$*, for* $m-r \geq 4$*, then it is likely that* $\mathrm{nrank}(U) = r$ *by virtue of Theorem A.2.2, and our proof of bound (2.4.7) applies even if we assume that* $\mathrm{nrank}(U) = r$ *rather than* $U \in \mathcal{G}^{m\times r}$*.*

2.5 Generation of multipliers. Counting flops and random variables

In our tests we have consistently succeeded by using multipliers from a *limited family* of very sparse and highly structured orthogonal matrices (cf. classes 13–17 of Section 2.7.3), but in this section we also cover a greater variety of other sparse and structured matrices, which form an *extended family* of multipliers.

We proceed in the following order. Given two integers l and n, $l \ll n$, we first generate four classes of very sparse primitive $n\times n$ unitary matrices, then combine them into some basic families of $n \times n$ matrices (we denote them $\widehat{B}$ in this section), and finally define multipliers B as $n\times l$ submatrices made up of l columns, which can be fixed (e.g., leftmost) or chosen at random. The matrix B is unitary if so is the matrix $\widehat{B}$, and more generally $\kappa(B) \leq \kappa(\widehat{B})$ (cf. [GL13, Theorem 8.6.3]).

2.5.1 $n \times n$ matrices of four primitive types

1. A fixed or random *permutation matrix* P. Their block submatrices form the class of CountSketch matrices from the data stream literature (cf. [W14, Section 2.1], [CCF04], [TZ12]).

2. A *diagonal matrix* $D = \mathrm{diag}(d_i)_{i=0}^{n-1}$, with fixed or random diagonal entries d_i such that $|d_i| = 1$ for all i (and so all n entries d_i lie on the unit circle $\{x : |z| = 1\}$, being either nonreal or ± 1).

3. An *f-circular shift matrix* $Z_f = \begin{pmatrix} \mathbf{0}^T & f \\ I & \mathbf{0} \end{pmatrix}$ and its transpose Z_f^T for a scalar f such that either $f = 0$ or $|f| = 1$. We write $Z = Z_0$, call Z *unit down-shift matrix*, and call the special permutation matrix Z_1 the *unit circulant matrix*.

4. A $2s \times 2s$ *Hadamard primitive matrix* $H^{(2s)} = \left(\begin{smallmatrix} I_s & I_s \\ I_s & -I_s \end{smallmatrix}\right)$ for a positive integer s (cf. [M11], [W14]).

All our primitive $n \times n$ matrices are very sparse and can be pre-multiplied by a vector in at most $2n$ flops. Except for the matrix Z, they are unitary or real orthogonal, and so is any $n \times l$ submatrix of Z of full rank l. Next we combine primitives 1–4 into families of $n \times n$ sparse and/or structured multipliers B.

2.5.2 Family (i): multipliers based on the Hadamard and Fourier processes

At first we recall the following recursive definition of dense and orthogonal (up to scaling by constants) $n \times n$ matrices H_n of *Walsh-Hadamard transform* for $n = 2^k$ (cf. [M11, Section 3.1] and our Remark 2.5.1):

$$H_{2q} = \begin{pmatrix} H_q & H_q \\ H_q & -H_q \end{pmatrix} \tag{2.5.1}$$

for $q = 2^h$, $h = 0, 1, \dots, k-1$, and the Hadamard primitive matrix $H_2 = H^{(2)} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$ of type 4 for $s = 1$. E.g., $H_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$, but for larger dimensions n, recursive representation (2.5.1) enables much faster pre-multiplication of a matrix H_n by a vector, namely it is sufficient to use nk additions and subtractions for $n = 2^k$.

Next we sparsify this matrix by defining it by a shorter recursive process, that is, by fixing a *recursion depth* d, $1 \le d < k$, and applying equation (2.5.1) where $q = 2^h s$, $h = k-d, k-d+1, \dots, k-1$, and $H_s I_s$ for $n = 2^d s$. For two positive integers d and s, we denote the resulting $n \times n$ matrix $H_{n,d}$ and for $1 \le d < k$ call it *d–Abridged Hadamard (AH) matrix*. In particular,

$$H_{n,1} = \begin{pmatrix} I_s & I_s \\ I_s & -I_s \end{pmatrix}, \text{for } n = 2s; \; H_{n,2} = \begin{pmatrix} I_s & I_s & I_s & I_s \\ I_s & -I_s & I_s & -I_s \\ I_s & I_s & -I_s & -I_s \\ I_s & -I_s & -I_s & I_s \end{pmatrix}, \text{ for } n = 4s,$$

$$H_{n,3} = \begin{pmatrix} I_s & I_s & I_s & I_s & I_s & I_s & I_s & I_s \\ I_s & -I_s & I_s & -I_s & I_s & -I_s & I_s & -I_s \\ I_s & I_s & -I_s & -I_s & I_s & I_s & -I_s & -I_s \\ I_s & -I_s & -I_s & I_s & I_s & -I_s & -I_s & I_s \\ I_s & I_s & I_s & I_s & -I_s & -I_s & -I_s & -I_s \\ I_s & -I_s & I_s & -I_s & -I_s & I_s & -I_s & I_s \\ I_s & I_s & -I_s & -I_s & -I_s & -I_s & I_s & I_s \\ I_s & -I_s & -I_s & I_s & -I_s & I_s & I_s & -I_s \end{pmatrix}, \text{ for } n = 8s.$$

For a fixed d, the matrix $H_{n,d}$ is still orthogonal up to scaling, has $q = 2^d$ nonzero entries in every row and column, and hence is sparse unless $k - d$ is a small integer. Then again, for larger dimensions n, we can pre-multiply such a matrix by a vector much faster if we represent it via recursive process (2.5.1), by using just dn additions/subtractions and allows efficient parallel implementation (cf. Remark 2.5.4).

We similarly obtain sparse matrices by shortening a recursive process of the generation of the $n \times n$ matrix Ω_n of *discrete Fourier transform (DFT)* at n points, for $n = 2^k$:

$$\begin{aligned} &\Omega_n = (\omega_n^{ij})_{i,j=0}^{n-1}, \text{ for } n = 2^k \text{ and a primitive } n\text{th root of unity} \\ &\omega_n = \exp(\frac{2\pi}{n}\sqrt{-1}). \text{ [E.g., } \Omega_2 = H^{(2)}.] \end{aligned} \tag{2.5.2}$$

The matrix Ω_n is unitary up to scaling by $\frac{1}{\sqrt{n}}$. We can pre-multiply it by a vector by using $1.5nk$ flops, and we can efficiently parallelize this computation if, instead of representation by entries, we apply following recursive

representation (cf. [P01, Section 2.3] and our Remark 2.5.1):[2]

$$\Omega_{2q} = \widehat{P}_{2q} \begin{pmatrix} \Omega_q & \Omega_q \\ \Omega_q \widehat{D}_q & -\Omega_q \widehat{D}_q \end{pmatrix}, \ \widehat{D}_q = \mathrm{diag}(\omega_n^i)_{i=0}^{n-1}. \tag{2.5.3}$$

Here $\widehat{P}_{2q}$ is the matrix of odd/even permutations such that $\widehat{P}_{2^h}(\mathbf{u}) = \mathbf{v}$, $\mathbf{u} = (u_i)_{i=0}^{2^h-1}$, $\mathbf{v} = (v_i)_{i=0}^{2^h-1}$, $v_j = u_{2j}$, $v_{j+2^{h-1}} = u_{2j+1}$, $j = 0, 1, \dots, 2^{h-1} - 1$; $q = 2^h$, $h = 0, 1, \dots, k$, and $\Omega_1 = (1)$ is the scalar 1.

We can sparsify this matrix by defining it by a shorter recursive process, that is, by fixing a recursion depth d, $1 \le d < k$, replacing Ω_s for $s = n/2^d$ by the identity matrix I_s, and then applying equation (4.3.3) for $q = 2^h$, $h = k - d, k - d + 1, \dots, k - 1$. For $1 \le d < k$ and $n = 2^d s$, we denote the resulting $n \times n$ matrix $\Omega_{n,d}$ and call it *d-Abridged Fourier (AF) matrix*. It is also unitary (up to scaling), has $q = 2^d$ nonzero entries in every row and column, and thus is sparse unless $k - d$ is a small integer. We can represent such a matrix by its entries, but then again its pre-multiplication by a vector involves just $1.5dn$ flops and allows highly efficient parallel implementation if we rely on recursive representation (4.3.3).

By applying fixed or random permutation and scaling to AH matrices $H_{n,d}$ and AF matrices $\Omega_{n,d}$, we obtain the families of *d–Abridged Scaled and*

[2]This is a representation of FFT, called decimation in frequency (DIF) radix-2 representation. Transposition turns it into an alternative representation of FFT, called decimation in time (DIT) radix-2 representation.

Permuted Hadamard (ASPH) matrices, PDH_n, and *d–Abridged Scaled and Permuted Fourier (ASPF)* $n \times n$ matrices, $PD\Omega_n$ where P and D are two matrices of permutation and diagonal scaling of primitive classes 1 and 2, respectively. Likewise we define the families of ASH, ASF, APH, and APF matrices, $DH_{n,d}$, $D\Omega_{n,d}$, $PH_{n,d}$, and $P\Omega_{n,d}$, respectively. Each random permutation or scaling contributes up to n random parameters.

Remark 2.5.1. *The following equations are equivalent to (2.5.1) and (4.3.3):*

$$H_{2q} = \operatorname{diag}(H_q, H_q)H^{(2q)} \text{ and } \Omega_{2q} = \widehat{P}_{2q} \operatorname{diag}(\Omega_q, \Omega_q \widehat{D}_q)H^{(2q)}.$$

Here $H^{(2q)}$ denotes a $2q \times 2q$ Hadamard's primitive matrix of type 4. By extending the latter recursive representation we can define matrices that involve more random parameters. Namely we can recursively incorporate random permutations and diagonal scaling as follows:

$$\widehat{H}_{2q} = P_{2q}D_{2q} \operatorname{diag}(\widehat{H}_q, \widehat{H}_q)H^{(2q)} \text{ and } \widehat{\Omega}_{2q} = P_{2q}D_{2q} \operatorname{diag}(\Omega_q, \Omega_q \widehat{D}_q)H^{(2q)}. \tag{2.5.4}$$

Here P_{2q} are $2q \times 2q$ random permutation matrices of primitive class 1 and D_{2q} are $2q \times 2q$ random matrices of diagonal scaling of primitive class 2, for all q. Then again we define d–abridged matrices $\widehat{H}_{n,d}$ and $\widehat{\Omega}_{n,d}$ by applying only d recursive steps (2.5.4) initiated at the primitive matrix I_s, for $s = n/2^d$.

With these recursive steps we can pre-multiply matrices $\widehat{H}_{n,d}$ and $\widehat{\Omega}_{n,d}$ by a vector by using at most $2dn$ additions and subtractions and at most $2.5dn$ flops, respectively, provided that 2^d divides n.

2.5.3 f-circulant, sparse f-circulant, and uniformly sparse matrices

An *f-circulant matrix* $Z_f(\mathbf{v}) = \sum_{i=0}^{n-1} v_i Z_f^i$ for the matrix Z_f of f-circular shift, is defined by a scalar $f \neq 0$ and by the first column $\mathbf{v} = (v_i)_{i=0}^{n-1}$ and is called *circulant* if $f = 1$ and *skew-circulant* if $f = -1$. Such a matrix is nonsingular with probability 1 (see Theorem A.1.1) and is likely to be well-conditioned [PSZ15] if $|f| = 1$ and if the vector $\mathbf{v}$ is Gaussian.

Remark 2.5.2. *One can compute the product of an $n \times n$ circulant matrix with an $n \times n$ Toeplitz or Toeplitz-like matrix by using $O(n \log(n))$ flops (see [P01, Theorem 2.6.4 and Example 4.4.1]).*

FAMILY (ii) of *sparse f-circulant matrices* $\widehat{B} = Z_f(\mathbf{v})$ is defined by a fixed or random scalar f, $|f| = 1$, and by the first column having exactly q nonzero entries, for $q \ll n$. The positions and the values of nonzeros can be randomized (and then the matrix would depend on up to $2n + 1$ random values).

Such a matrix can be pre-multiplied by a vector by using at most $(2q-1)n$

flops or, in the real case where $f = \pm 1$ and $v_i = \pm 1$ for all i, by using at most qn additions and subtractions.

The same cost estimates apply in the case of the generalization of $Z_f(\mathbf{v})$ to a *uniformly sparse matrix* with exactly q nonzeros entries, ± 1, in every row and in every column for $1 \leq q \ll n$. Such a matrix is the sum $\widehat{B} = \sum_{i=1}^{q} \widehat{D}_i P_i$ for fixed or random matrices P_i and $\widehat{D}_i$ of primitive types 1 and 2, respectively.

2.5.4 Abridged f-circulant matrices

First recall the well-known expression $Z_g(\mathbf{v}) = \sum_{i=0}^{n-1} v_i Z_g^i = D_f^{-1} \Omega_n^H D \Omega_n D_f$ where $g = f^n$, $D_f = \mathrm{diag}(f^i)_{i=0}^{n-1}$, $\mathbf{v} = (v_i)_{i=0}^{n-1} = (\Omega_n D_f)^{-1}\mathbf{u}$, $\mathbf{u} = (u_i)_{i=0}^{n-1}$, and $D = \mathrm{diag}(u_i)_{i=0}^{n-1}$ (cf. [P01, Theorem 2.6.4]). For $f = 1$, the expression is simplified: $g = 1$, $D_f = I_n$, and $Z_g(\mathbf{v})$ is a circulant matrix:

$$Z_1(\mathbf{v}) = \Omega_n^H D \Omega_n, \ D = \mathrm{diag}(u_i)_{i=0}^{n-1}, \ \text{for } \mathbf{u} = (u_i)_{i=0}^{n-1} = \Omega_n \mathbf{v}. \tag{2.5.5}$$

Pre-multiplication of an f-circulant matrix by a vector is reduced to pre-multiplication of each of the matrices Ω and Ω^H by a vector and in addition to performing $4n$ flops (or $2n$ flops in case of a circulant matrix). This involves $O(n \log(n))$ flops overall and then again allows highly efficient parallel implementation.

For a fixed scalar f and $g = f^n$, we can define the matrix $Z_g(\mathbf{v})$ by any

of the two vectors $\mathbf{u}$ or $\mathbf{v}$. The matrix is unitary (up to scaling) if $|f| = 1$ and if $|u_i| = 1$ for all i and is defined by $n+1$ real parameters (or by n such parameters for a fixed f), which we can fix or choose at random.

Now suppose that $n = 2^d s$, $1 \le d < k$, d and k are integers, and substitute a pair of AF matrices of recursion length d for two factors Ω_n in the above expressions. Then the resulting *abridged f-circulant matrix $Z_{g,d}(\mathbf{v})$ of recursion depth d* is still unitary (up to scaling), defined by $n+1$ or n parameters u_i and f, is sparse unless the positive integer $k-d$ is small, and can be pre-multiplied by a vector by using $(3d+3)n$ flops. Instead of AF matrices, we can substitute a pair of ASPF, APF, ASF, AH, ASPH, APH, or ASF matrices for the factors Ω_n. All such matrices form **FAMILY (iii)** of *d–abridged f-circulant matrices*.

Remark 2.5.3. *Recall that an $n \times l$ SRFT and SRHT matrices are the products $\sqrt{n/l}\ D\Omega_n R$ and $\sqrt{n/l}\ DH_n R$, respectively, where H_n and Ω_n are the matrices of (2.5.1) and (2.5.2), $D = \mathrm{diag}(u_i)_{i=0}^{n-1}$, u_i are i.i.d. variables uniformly distributed on the circle $\{u : |u| = \sqrt{n/l}\}$, and R is the $n \times l$ submatrix formed by l columns of the identity matrix I_n chosen uniformly at random. Equation (2.5.5) shows that we can obtain a SRFT matrix by pre-multiplying a circulant matrix by the matrix Ω_n and post-multiplying it by the above matrix R.*

2.5.5 Inverses of bidiagonal matrices

FAMILY (iv) is formed by the *inverses of* $n \times n$ *bidiagonal matrices* $\widehat{B} = (I_n + DZ)^{-1}$ or $\widehat{B} = (I_n + Z^T D)^{-1}$ for a matrix D of primitive type 2 and the down-shift matrix Z.

We pre-multiply a matrix $\widehat{B} = (I_n + DZ)^{-1}$ by a vector $\mathbf{v}$ by solving the linear system $(I_n + DZ)\mathbf{x} = \mathbf{v}$ in $2n - 1$ flops. In the real case we use just $n - 1$ additions and subtractions. We randomize this matrix $\widehat{B}$ by choosing up to $n-1$ random diagonal entries of the matrix D (its leading entry makes no impact on $\widehat{B}$).

Finally, $||\widehat{B}|| \le \sqrt{n}$ because nonzero entries of the lower triangular matrix $\widehat{B} = (I_n + DZ)^{-1}$ have absolute values 1, and clearly $||\widehat{B}^{-1}|| = ||I_n + DZ|| \le \sqrt{2}$. Hence $\kappa(\widehat{B}) = ||\widehat{B}||\ ||\widehat{B}^{-1}||$ (the spectral condition number of $\widehat{B}$) cannot exceed $\sqrt{2n}$ for $\widehat{B} = (I_n + DZ)^{-1}$, and the same bound holds for $\widehat{B} = (I_n + Z^T D)^{-1}$.

2.5.6 Summary of estimated numbers of flops and random variables involved

Table 2.5.1 shows upper bounds on

(a) the numbers of random variables involved into the $n \times n$ matrices $\widehat{B}$ of the four families (i)–(iv) and

(b) the numbers of flops for pre-multiplication of such a matrix by a vector.[3]

For comparison, using a Gaussian $n \times n$ multiplier involves n^2 random variables and $(2n-1)n$ flops.

One can readily extend the estimates to $n \times l$ submatrices B of the matrices $\widehat{B}$.

Table 2.5.1: The numbers of random variables and flops

family	(i) AH	(i) ASPH	(ii)	(iii)	(iv)
random variables	0	$2n$	$2q+1$	n	$n-1$
flops complex	dn	$(d+1)n$	$(2q-1)n$	$(3d+2)n$	$2n-1$
flops in real case	dn	$(d+1)n$	qn	*	$n-1$

Remark 2.5.4. *Other observations besides flop estimates can be decisive. E.g., a special recursive structure of an ARSPH matrix $H_{2^k,d}$ and an ARSPF matrix $\Omega_{2^k,d}$ allows highly efficient parallel implementation of their pre-multiplication by a vector based on Application Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs), incorporating Butterfly Circuits [DE].*

[3]The asterisks in the table show that the matrices of families (i) AF, (i) ASPF, and (iii) involve nonreal roots of unity.

2.5.7 Other basic families

There is a number of other interesting basic matrix families. According to [HMT11, Remark 4.6], "among the structured random matrices .... one of the strongest candidates involves sequences of random Givens rotations". They are dense unitary matrices $\frac{1}{\sqrt{n}}D_1G_1D_2G_2D_3\Omega_n$, for the DFT matrix Ω_n, three random diagonal matrices D_1, D_2 and D_3 of primitive type 2, and two chains of Givens rotations G_1 and G_2, each of the form $G(\theta_1,\dots,\theta_{n-1}) = P\prod_{i=1}^{n-1}G(i,i+1,\theta_i)$ for a random permutation matrix P,

$$G(i,i+1,\theta_i) = \mathrm{diag}(I_{i-1}, \begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix}, I_{n-i-1}),\ c_i = \cos\theta_i,\ s_i = \sin\theta_i,\ c_i^2+s_i^2 = 1.$$

Here $\theta_1,\dots,\theta_{n-1}$ denote $n-1$ random angles of rotation uniformly distributed in the range $0 \le \phi < 2\pi$.

The DFT factor Ω_n makes the resulting matrices dense, but we can sparsify them by replacing that factor by an AF, ASF, APF, or ASPF matrix having recursion depth $d < \log_2(n)$. This would also decrease the number of flops involved in pre-multiplication of such a multiplier by a vector from order $n\log_2(n)$ to $1.5dn + O(n)$. We can turn Givens sequences into distinct candidate families of efficient multipliers by replacing either or both of the Givens products with sparse matrices of Householder reflections matrices of the form $I_n - \frac{2\mathbf{h}\mathbf{h}^T}{\mathbf{h}^T\mathbf{h}}$ for fixed or random sparse vectors $\mathbf{h}$ (cf. [GL13, Section

5.1]).

We can obtain a variety of multiplier families by combining matrices of basic families (i)–(iv) and the above matrices. Besides linear combinations, we can apply block representation as in the following real 2×2 block matrix $\frac{1}{\sqrt{n}} \begin{pmatrix} Z_1(\mathbf{u}) & Z_1(\mathbf{v}) \\ Z_1(\mathbf{v}) & -Z_1(\mathbf{u}) \end{pmatrix} D$ for two vectors $\mathbf{u}$ and $\mathbf{v}$ and a matrix D of primitive class 2.

The reader can find other useful families of multipliers in our Section 2.7. E.g., according to our tests in Section 2.7, it turned out to be efficient to use nonsingular well-conditioned (rather than unitary) diagonal factors in the definition of some of our basic matrix families.

2.6 Managing the unlikely failure of Algorithm 2.3.1

2.6.1 Some basic observations

Theorem 2.6.1. *Given an $m \times n$ matrix M with* $\operatorname{nrank}(M) = r$ *and a reasonably small positive tolerance τ, Algorithm 2.3.1 outputs SUCCESS if and only if* $\operatorname{nrank}(MB) = r$.

Proof. If $\operatorname{rank}(MB) = r_- < r$, then $\operatorname{rank}(\tilde{M}) \le r_- < r$, $\Delta \ge \sigma_{r_-}(M)$. In this case Δ is not small because $\operatorname{nrank}(M) = r > r_-$, and so Algorithm 2.3.1 applied to M with the multiplier B outputs FAILURE.

If $\text{rank}(MB) = r > \text{nrank}(MB) = r_-$, then $\text{rank}(MB - E) = r_- < r$ for a small-norm perturbation matrix E. Hence $\Delta \geq \sigma_{r_-}(M) - O(||E||)$, and then again Algorithm 2.3.1 applied to M with the multiplier B outputs FAILURE. This proves the "only if" claim of the claim of Theorem 2.6.1.

Now let $\text{nrank}(MB) = r$ and assume that we scale the matrix B so that $||B||_F = 1$. Then $\text{rank}(MB) = r$ (and so we can apply bound (2.4.5)), and furthermore $\text{nrank}(M_rB) = \text{nrank}(MB) = r$. Equation (2.4.5) implies that $\Delta \approx 8\sqrt{8(n-r)}\sigma_{r+1}||(M_rB)^+||$. Therefore Δ is a small positive value because $\text{nrank}(M) = r$. Thus the value $|\sigma_{r+1}|$ is small, and claim "if" of Theorem 2.6.1 follows. □

Definition 2.6.1. *For two integers l and n, $0 < l \leq n$, and any fixed $n \times l$ multiplier B, partition the set of $m \times n$ matrices M with* $\text{nrank}(M) = r$ *into the set $\mathcal{M}_B = \mathcal{M}_{B,\text{good}}$ of "B-good" matrices such that* $\text{nrank}(MB) = r$ *and the set $\mathcal{M}_{B,\text{bad}}$ of "B-bad" matrices such that* $\text{nrank}(MB) < r$.

The following simple observations should be instructive.

Theorem 2.6.2. *(Cf. Remark 2.6.2.) Consider a vector* $\mathbf{v}$ *of dimension n, an $n \times n$ unitary matrix U, and an $n \times l$ unitary matrix B, so that $n \times l$ matrix UB is unitary. Then*

(i) $\mathcal{M}_{UB} = (\mathcal{M}_B)U$, that is, the map $B \to UB$ multiplies the class $\mathcal{M}_B$

of B-good $m \times n$ matrices by the unitary matrix U,

(ii) $\mathcal{M}_B \subseteq \mathcal{M}_{(B \mid \mathbf{v})}$, that is, appending a column to a multiplier B can only expand the class $\mathcal{M}_B$, and

(iii) this class fills the whole space $\mathbb{C}_{m,n,r}$ or $\mathbb{R}_{m,n,r}$ if $l = n$.

Proof. The relationships $(MU)B = M(UB)$ and

$$\operatorname{nrank}(MB) \le \operatorname{nrank}(M(B \mid \mathbf{v}))$$

imply claims (i) and (ii), respectively. Claim (iii) follows because

$$\operatorname{nrank}(MB) = \operatorname{nrank}(M)$$

if B is an $n \times n$ unitary matrix. □

Remark 2.6.1. *In view of claim (i) of Theorem 2.6.2, the results for the classes of SRFT and SRHT matrices also hold for the products of these classes with any unitary matrix, in particular for the class of $n \times l$ submatrices of an $n \times n$ circulant matrix, each made up of l randomly chosen columns (see Remark 2.5.3).*

2.6.2 A recursive algorithm

Based on Theorem 2.6.2 we devise the following algorithm where $\operatorname{nrank}(M)$ is not known.

Algorithm 2.6.1. Recursive low-rank approximation. *See Figure 1 and cf. [HMT11, Algorithm 4.2].*

INPUT: *An $m \times n$ matrix M and a nonnegative tolerance τ.*

COMPUTATIONS: *1. Generate an $n \times n$ unitary matrix $\widehat{B}$.*

2. Fix positive integers $l_1, \dots, l_h$ such that $l_1 + \cdots + l_h = n$ (in particular $l_j = 1$ for all j if $h = n$) and represent the matrix $\widehat{B}$ as a block vector $(B_1 \mid B_2 \mid \dots \mid B_h)$ where the block B_i has size $n \times l_i$ for $i = 1, \dots, h$.

3. Recursively, for $i = 1, 2, \dots,$ apply Algorithm 2.3.1 to the matrix M by substituting $l^{(i)} = \sum_{j=1}^{i} l_j$ for l and $B^{(i)} = (B_1 \mid \dots \mid B_i)$ for B. Stop If the algorithm outputs SUCCESS, output a rank-r approximation of M by applying [HMT11, Algorithm 5.2].

Figure 2.1: Matrices of Algorithm 2.6.1

Remark 2.6.2. *One can readily extend the algorithm by using a nonsingular and well-conditioned (rather than unitary) $n \times n$ matrix $\widehat{B}$. Then all multipliers $B^{(i)}$ and their blocks B_j are also well-conditioned matrices of full rank; moreover $\kappa(B^{(i)}) \le \kappa(B)$ and $\kappa(B_j) \le \kappa(B)$ for all i and j (cf. [GL13, Corollary 8.6.3]).*

The complexity of the algorithm is dominated at Stage 1, which uses from $(2n-1)ml$ flops for generic matrices M and B to order of mn for sparse and structured multipliers B even where $l = n$.

Our conflicting goals and simple recipes. We try to decrease:

(i) the cost of the generation of a multiplier B and of the computation of the matrix MB,

(ii) the chances for the failure of Algorithm 2.3.1, and

(iii) the rank of the computed approximation of a matrix M.

Towards goal (i) we apply sparse and structured $n \times l$ multipliers. Towards goal (ii) we can expect to succeed whenever integer parameter l exceeds $r+1$, but our chances for success grow fast as l increases. Such an increase is in conflict with our goal (iii), but we can alleviate the problem by using the following heuristic technique, which consistently worked in our extensive tests for benchmark inputs in Section 2.7.

Heuristic Compression Algorithm (linear combination of failed multipliers): if the first h recursive steps of Algorithm 2.6.1 have failed for $h > 1$, then apply Algorithm 2.3.1 with a multiplier $B = \sum_{j=1}^{h} c_j B_j$ where $c_j = \pm 1$ for all j and for a fixed or random choice of the signs $\pm$. (More generally, one can choose complex values c_j on the unit circle, letting $|c_j| = 1$ for all j.)

2.7 Numerical Tests

Numerical experiments have been performed by Xiaodong Yan for Tables 4.6.1–4.6.3 and by John Svadlenka and Liang Zhao for the other tables. The report [PSZa] displays graphs of the same data. The tests have been run by using MATLAB in the Graduate Center of the City University of New York on a Dell computer with the Intel Core 2 2.50 GHz processor and 4G memory running Windows 7; in particular the standard normal distribution function randn of MATLAB has been applied in order to generate Gaussian matrices.

We calculated the ξ-rank, i.e., the number of singular values exceeding ξ, by applying the MATLAB function "svd()". We have set $\xi = 10^{-5}$ in Sections 2.7.1 and 2.7.2 and $\xi = 10^{-6}$ in Section 2.7.3.

2.7.1 Tests for inputs generated via SVD

In the tests of this subsection we generated $n \times n$ input matrices M by extending the customary recipes of [H02, Section 28.3]. Namely, we first generated matrices S_M and T_M by means of the orthogonalization of $n \times n$ Gaussian matrices. Then we defined $n \times n$ matrices M by their compact SVDs, $M = S_M \Sigma_M T_M^T$, for $\Sigma_M = \mathrm{diag}(\sigma_j)_{j=1}^n$; $\sigma_j = 1/j$, $j = 1, \dots, r$, $\sigma_j = 10^{-10}$, $j = r+1, \dots, n$, and $n = 256, 512, 1024$. (Hence $||M|| = 1$ and

$\kappa(M) = ||M||\ ||M^{-1}|| = 10^{10}$.)

Table 2.7.1 shows the average output error norms Δ over 1000 tests of Algorithm 2.3.1 applied to these matrices M for each pair of n and r, $n = 256, 512, 1024$, $r = 8, 32$, and each of the following three groups of multipliers: 3-AH multipliers, 3-ASPH multipliers, both defined by Hadamard recursion (4.3.3), for $d = 3$, and dense multipliers $B = B(\pm 1, 0)$ having i.i.d. entries ± 1 and 0, each value chosen with probability 1/3.

Table 2.7.1: Error norms for SVD-generated inputs and 3-AH, 3-ASPH, and $B(\pm 1, 0)$ multipliers

n	r	3-AH	3-ASPH	$B(\pm 1, 0)$
256	8	2.25e-08	2.70e-08	2.52e-08
256	32	5.95e-08	1.47e-07	3.19e-08
512	8	4.80e-08	2.22e-07	4.76e-08
512	32	6.22e-08	8.91e-08	6.39e-08
1024	8	5.65e-08	2.86e-08	1.25e-08
1024	32	1.94e-07	5.33e-08	4.72e-08

Tables 4.6.1–4.6.3 show the mean and maximal values of such an error norm in the case of (a) real Gaussian multipliers B and dense real Gaussian subcirculant multipliers B, for $q = n$, each defined by its first column filled with either (b) i.i.d. Gaussian variables or (c) random variables ± 1. Here and hereafter in this section we assigned each random signs $+$ or $-$ with probability 0.5.

Table 2.7.2: Error norms for SVD-generated inputs and Gaussian multipliers

r	n	**mean**	**max**
8	256	7.54×10^{-8}	1.75×10^{-5}
8	512	4.57×10^{-8}	5.88×10^{-6}
8	1024	1.03×10^{-7}	3.93×10^{-5}
32	256	5.41×10^{-8}	3.52×10^{-6}
32	512	1.75×10^{-7}	5.57×10^{-5}
32	1024	1.79×10^{-7}	3.36×10^{-5}

Table 2.7.3: Error norms for SVD-generated inputs and Gaussian subcirculant multipliers

r	n	**mean**	**max**
8	256	3.24×10^{-8}	2.66×10^{-6}
8	512	5.58×10^{-8}	1.14×10^{-5}
8	1024	1.03×10^{-7}	1.22×10^{-5}
32	256	1.12×10^{-7}	3.42×10^{-5}
32	512	1.38×10^{-7}	3.87×10^{-5}
32	1024	1.18×10^{-7}	1.84×10^{-5}

Table 4.6.4 displays the average error norms in the case of multipliers B of eight kinds defined below, all generated from the following Basic Sets 1, 2 and 3 of $n \times n$ multipliers:

Basic Set 1: 3-APF multipliers defined by three Fourier recursive steps of equation (4.3.3), for $d = 3$, with no scaling, but with a random column permutation.

Basic Set 2: Sparse real circulant matrices $Z_1(\mathbf{v})$ of family (ii) of Section

Table 2.7.4: Error norms for SVD-generated inputs and random subcirculant multipliers filled with ± 1

r	n	**mean**	**max**
8	256	7.70×10^{-9}	2.21×10^{-7}
8	512	1.10×10^{-8}	2.21×10^{-7}
8	1024	1.69×10^{-8}	4.15×10^{-7}
32	256	1.51×10^{-8}	3.05×10^{-7}
32	512	2.11×10^{-8}	3.60×10^{-7}
32	1024	3.21×10^{-8}	5.61×10^{-7}

2.5.3 (for $q = 10$) having the first column vectors $\mathbf{v}$ filled with zeros, except for ten random coordinates filled with random integers ± 1.

Basic Set 3: Sum of two scaled inverse bidiagonal matrices. We first filled the main diagonals of both matrices with the integer 101 and their first subdiagonals with ± 1. Then we multiplied each matrix by a diagonal matrix $\mathrm{diag}(\pm 2^{b_i})$, where b_i were random integers uniformly chosen from 0 to 3.

For multipliers B we used the $n \times r$ western (leftmost) blocks of $n \times n$ matrices of the following classes:

1. a matrix from Basic Set 1;

2. a matrix from Basic Set 2;

3. a matrix from Basic Set 3;

4. the product of two matrices of Basic Set 1;

5. the product of two matrices of Basic Set 2;

6. the product of two matrices of Basic Set 3;

7. the sum of two matrices of Basic Sets 1 and 3, and

8. the sum of two matrices of Basic Sets 2 and 3.

The tests produced the results similar to the ones of Tables 2.7.1–4.6.3.

In sum, for all classes of input pairs M and B and all pairs of integers n and r, Algorithm 2.3.1 with our pre-processing has consistently output approximations to rank-r input matrices with the average error norms ranged from 10^{-7} or 10^{-8} to about 10^{-9} in all our tests.

Table 2.7.5: Error norms for SVD-generated inputs and multipliers of eight classes

n	r	class 1	class 2	class 3	class 4	class 5	class 6
256	8	5.94e-09	4.35e-08	2.64e-08	2.20e-08	7.73e-07	5.15e-09
256	32	2.40e-08	2.55e-09	8.23e-08	1.58e-08	4.58e-09	1.36e-08
512	8	1.11e-08	8.01e-09	2.36e-09	7.48e-09	1.53e-08	8.15e-09
512	32	1.61e-08	4.81e-09	1.61e-08	2.83e-09	2.35e-08	3.48e-08
1024	8	5.40e-09	3.44e-09	6.82e-08	4.39e-08	1.20e-08	4.44e-09
1024	32	2.18e-08	2.03e-08	8.72e-08	2.77e-08	3.15e-08	7.99e-09

2.7.2 Tests for inputs generated via the discretization of a Laplacian operator and via the approximation of an inverse finite-difference operator

Next we present the test results for Algorithm 2.3.1 applied to input matrices for computational problems of two kinds, both replicated from [HMT11], namely, the matrices of

(i) the discretized single-layer Laplacian operator and

(ii) the approximation of the inverse of a finite-difference operator.

Input matrices (i). We considered the Laplacian operator $[S\sigma](x) = c\int_{\Gamma_1}\log|x-y|\sigma(y)dy, x\in\Gamma_2$, from [HMT11, Section 7.1], for two contours $\Gamma_1 = C(0,1)$ and $\Gamma_2 = C(0,2)$ on the complex plane. Its dscretization defines an $n\times n$ matrix $M = (m_{ij})_{i,j=1}^n$ where $m_{i,j} = c\int_{\Gamma_{1,j}}\log|2\omega^i - y|dy$ for a constant c such that $||M|| = 1$ and for the arc $\Gamma_{1,j}$ of the contour Γ_1 defined by the angles in the range $[\frac{2j\pi}{n}, \frac{2(j+1)\pi}{n}]$.

We applied Algorithm 2.3.1 supported by three iterations of the Power Scheme of Remark 2.4.5 and used with multipliers B being the $n\times r$ leftmost submatrices of $n\times n$ matrices of the following five classes:

- Gaussian multipliers,
- Gaussian Toeplitz multipliers $T = (t_{i-j})_{i=0}^{n-1}$ for i.i.d. Gaussian variables $t_{1-n},\dots,t_{-1},t_0,t_1,\dots,t_{n-1}$.

- Gaussian circulant multipliers $\sum_{i=0}^{n-1} v_i Z_1^i$, for i.i.d. Gaussian variables $v_0, \dots, v_{n-1}$ and the unit circular matrix Z_1 of Section 2.5.1.
- Abridged permuted Fourier (3-APF) multipliers, and
- Abridged permuted Hadamard (3-APH) multipliers.

As in the previous subsection, we defined each 3-APF and 3-APH matrix by applying three recursive steps of equation (4.3.3) followed by a single random column permutation.

We applied Algorithm 2.3.1 with multipliers of all five listed classes. For each setting we repeated the test 1000 times and calculated the mean and standard deviation of the error norm $||\tilde{M} - M||$.

Input matrices (ii). We similarly applied Algorithm 2.3.1 to the input matrix M being the inverse of a large sparse matrix obtained from a finite-difference operator of [HMT11, Section 7.2] and observed similar results with all structured and Gaussian multipliers.

We performed 1000 tests for every class of pairs of $n \times n$ or $m \times n$ matrices of classes (i) or (ii), respectively, and $n \times r$ multipliers for every fixed triple of m, n, and r or pair of n and r.

Tables 2.7.6 and 2.7.7 display the resulting mean values and standard deviation of the error norms.

Table 2.7.6: Low-rank approximation of Laplacian matrices

n	multiplier	r	mean	std
200	Gaussian	3.00	1.58e-05	1.24e-05
200	Toeplitz	3.00	1.83e-05	7.05e-06
200	Circulant	3.00	3.14e-05	2.30e-05
200	3-APF	3.00	8.50e-06	5.15e-15
200	3-APH	3.00	2.18e-05	6.48e-14
400	Gaussian	3.00	1.53e-05	1.37e-06
400	Toeplitz	3.00	1.82e-05	1.59e-05
400	Circulant	3.00	4.37e-05	3.94e-05
400	3-APF	3.00	8.33e-06	1.02e-14
400	3-APH	3.00	2.18e-05	9.08e-14
2000	Gaussian	3.00	2.10e-05	2.28e-05
2000	Toeplitz	3.00	2.02e-05	1.42e-05
2000	Circulant	3.00	6.23e-05	7.62e-05
2000	3-APF	3.00	1.31e-05	6.16e-14
2000	3-APH	3.00	2.11e-05	4.49e-12
4000	Gaussian	3.00	2.18e-05	3.17e-05
4000	Toeplitz	3.00	2.52e-05	3.64e-05
4000	Circulant	3.00	8.98e-05	8.27e-05
4000	3-APF	3.00	5.69e-05	1.28e-13
4000	3-APH	3.00	3.17e-05	8.64e-12

2.7.3 Tests with additional classes of multipliers

In this subsection we display the mean values and standard deviations of the error norms observed when we repeated the tests of the two previous subsections for the same three classes of input matrices (that is, SVD-generated, Laplacian, and matrices obtained by discretization of finite difference operators), but now we applied Algorithm 2.3.1 with seventeen additional classes

Table 2.7.7: Low-rank approximation of the matrices of discretized finite-difference operator

m	n	multiplier	r	mean	std
88	160	Gaussian	5.00	1.53e-05	1.03e-05
88	160	Toeplitz	5.00	1.37e-05	1.17e-05
88	160	Circulant	5.00	2.79e-05	2.33e-05
88	160	3-APF	5.00	4.84e-04	2.94e-14
88	160	3-APH	5.00	4.84e-04	5.76e-14
208	400	Gaussian	43.00	4.02e-05	1.05e-05
208	400	Toeplitz	43.00	8.19e-05	1.63e-05
208	400	Circulant	43.00	8.72e-05	2.09e-05
208	400	3-APF	43.00	1.24e-04	2.40e-13
208	400	3-APH	43.00	1.29e-04	4.62e-13
408	800	Gaussian	64.00	6.09e-05	1.75e-05
408	800	Toeplitz	64.00	1.07e-04	2.67e-05
408	800	Circulant	64.00	1.04e-04	2.67e-05
408	800	3-APF	64.00	1.84e-04	6.42e-12
408	800	3-APH	64.00	1.38e-04	8.65e-12

of multipliers (besides its control application with Gaussian multipliers). We tested Algorithm 2.3.1 applied to 1024×1024 SVD-generated input matrices having numerical nullity $r = 32$, to 400×400 Laplacian input matrices having numerical nullity $r = 3$, and to 408×800 matrices having numerical nullity $r = 64$ and representing finite-difference inputs.

Then again we repeated the tests 1000 times for each class of input matrices and each size of an input and a multiplier, and we display the resulting average error norms in Table 2.7.3.

We used multipliers defined as the seventeen sums of $n \times r$ matrices of the following basic families:

- 3-ASPH matrices
- 3-APH matrices
- Inverses of bidiagonal matrices
- Random permutation matrices

We defined every 3-APH matrix by three Hadamard's recursive steps (2.5.1) followed by random permutation, and we similarly defined every 3-ASPH matrix, but also applied random scaling with a diagonal matrix $D = \mathrm{diag}(d_i)_{i=1}^{n}$ choosing the values of random i.i.d. variables d_i uniformly from the set $\{1/4, 1/2, 1, 2, 4\}$. We permuted all inverses of bidiagonal matrices except for Class 5 of multipliers.

Describing our multipliers we use the following acronyms and abbreviations: "IBD" for "the inverse of a bidiagonal", "MD" for "the main diagonal", "SB" for "subdiagonal", and "SP" for "superdiagonal". We write "MDi", "kth SBi" and "kth SPi" in order to denote that the main diagonal, the kth subdiagonal, or the kth superdiagonal of a bidiagonal matrix, respectively, was filled with the integer i.

- Class 0: Gaussian

- Class 1: Sum of a 3-ASPH and two IBD matrices:

  B1 with MD−1 and 2nd SB−1 and B2 with MD+1 and 1st SP+1

- Class 2: Sum of a 3-ASPH and two IBD matrices:

  B1 with MD+1 and 2nd SB−1 and B2 with MD+1 and 1st SP−1

- Class 3: Sum of a 3-ASPH and two IBD matrices:

  B1 with MD+1 and 1st SB−1 and B2 with MD +1 and 1st SP−1

- Class 4: Sum of a 3-ASPH and two IBD matrices:

  B1 with MD+1 and 1st SB+1 and B2 with MD+1 and 1st SP−1

- Class 5: Sum of a 3-ASPH and two IBD matrices:

  B1 with MD+1 and 1st SB+1 and B2 with MD+1 and 1st SP−1

- Class 6: Sum of a 3-ASPH and three IBD matrices:

  B1 with MD−1 and 2nd SB−1, B2 with MD+1 and 1st SP+1 and B3 with MD+1 and 9th SB+1

- Class 7: Sum of a 3-ASPH and three IBD matrices:

  B1 with MD+1 and 2nd SB−1, B2 with MD+1 and 1st SP−1, and B3 with MD+1 and 8th SP+1

- Class 8: Sum of a 3-ASPH and three IBD matrices:

  B1 with MD+1 and 1st SB−1, B2 with MD+1 and 1st SP−1, and B3 with MD+1 and 4th SB+1

- Class 9: Sum of a 3-ASPH and three IBD matrices:

  B1 with MD+1 and 1st SB+1, B2 with MD+1 and 1st SP−1, and B3 with MD−1 and 3rd SP+1

- Class 10: Sum of three IBD matrices:

  B1 with MD+1 and 1st SB+1, B2 with MD+1 and 1st SP−1, and B3 with MD−1 and 3rd SP+1

- Class 11: Sum of a 3-APH and three IBD matrices:

  B1 with MD+1 and 2nd SB−1, B2 with MD+1 and 1st SP−1, and B3 with MD+1 and 8th SP+1

- Class 12: Sum of a 3-APH and two IBD matrices:

  B1 with MD+1 and 1st SB−1 and B2 with MD+1 and 1st SP−1

- Class 13: Sum of a 3-ASPH and a permutation matrix

- Class 14: Sum of a 3-ASPH and two permutation matrices

- Class 15: Sum of a 3-ASPH and three permutation matrices

- Class 16: Sum of a 3-APH and three permutation matrices
- Class 17: Sum of a 3-APH and two permutation matrices

	Random Matrices		Laplacian Matrices		FD Matrices	
Class No.	Mean	Std	Mean	Std	Mean	Std
Class 0	3.54E-09	3.28E-09	4.10E-14	2.43E-13	1.61E-06	1.35E-06
Class 0	1.07E-08	3.82E-09	2.05E-13	1.62E-13	4.58E-06	9.93E-07
Class 1	1.16E-08	6.62E-09	6.07E-13	5.20E-13	4.67E-06	1.04E-06
Class 2	1.23E-08	5.84E-09	1.69E-13	1.34E-13	4.52E-06	1.01E-06
Class 3	1.25E-08	1.07E-08	2.46E-13	3.44E-13	4.72E-06	9.52E-07
Class 4	1.13E-08	6.09E-09	1.93E-13	1.48E-13	4.38E-06	8.64E-07
Class 5	1.12E-08	8.79E-09	9.25E-13	2.64E-12	5.12E-06	1.29E-06
Class 6	1.16E-08	7.42E-09	5.51E-13	5.35E-13	4.79E-06	1.12E-06
Class 7	1.33E-08	1.00E-08	1.98E-13	1.30E-13	4.60E-06	9.52E-07
Class 8	1.08E-08	4.81E-09	2.09E-13	3.60E-13	4.47E-06	8.57E-07
Class 9	1.18E-08	5.51E-09	1.87E-13	1.77E-13	4.63E-06	9.28E-07
Class 10	1.18E-08	6.23E-09	1.78E-13	1.42E-13	4.55E-06	9.08E-07
Class 11	1.28E-08	1.40E-08	2.33E-13	3.44E-13	4.49E-06	9.67E-07
Class 12	1.43E-08	1.87E-08	1.78E-13	1.61E-13	4.74E-06	1.19E-06
Class 13	1.22E-08	1.26E-08	2.21E-13	2.83E-13	4.75E-06	1.14E-06
Class 14	1.51E-08	1.18E-08	3.57E-13	9.27E-13	4.61E-06	1.08E-06
Class 15	1.19E-08	6.93E-09	2.24E-13	1.76E-13	4.74E-06	1.09E-06
Class 16	1.26E-08	1.16E-08	2.15E-13	1.70E-13	4.59E-06	1.12E-06
Class 17	1.31E-08	1.18E-08	1.25E-14	5.16E-14	1.83E-06	1.55E-06

Table 2.7.8: Relative Error Norm in Tests with Multipliers of Additional Classes

The outputs were quite accurate even where we used very sparse multipliers of classes 13–17.

2.8 A Brief Summary

In this chapter we

(i) supplied a missing *formal support* for well-known empirical observations,

(ii) defined *new efficient policies of generation and application of multipliers* for low-rank approximation,

(iii) *successfully tested our policies numerically*, and

(iv) *extended our progress* to other important areas of matrix computations.

Bibliography

[CCF04] M. Charikar, K. Chen, M. Farach-Colton, Finding Frequent Items in Data Streams, *Theoretical Computer Science*, **312, 1**, 3–15, 2004.

[CD05] Z. Chen, J. J. Dongarra, Condition Numbers of Gaussian Random Matrices, *SIAM. J. on Matrix Analysis and Applications*, **27**, 603–620, 2005.

[CMMa] Michael B. Cohen, Cameron Musco, Christopher Musco, Input Sparsity Time Low-rank Approximation via Ridge Leverage Score Sampling, arXiv 1511.07263 [cs.DS], October 2016.

[CW13] K. L. Clarkson, D. P. Woodruff, Low Rank Approximation and Regression in Input Sparsity Time, *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing (STOC '13)*, 81–90, ACM New York, NY, USA, 2013. Full Version in arXiv:1207.6365.

[DE] *Dillon Engineering*, http://www.dilloneng.com/fft_ip/parallel-fft.

[GL13] G. H. Golub, C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, 2013 (fourth edition).

[H02] N. J. Higham, *Accuracy and Stability in Numerical Analysis*, SIAM, Philadelphia, 2002.

[HMST11] N. Halko, P. G. Martinsson, Y. Shkolnisky, M. Tygert, An Algorithm for the Principal Component Analysis of Large Data Sets, *SIAM J. Scientific Computation*, **33, 5**, 2580–2594, 2011.

[HMT11] N. Halko, P.G. Martinsson, J.A. Tropp, Finding Structure with Randomness: Probabilistic Algorithms for Approximate Matrix Decompositions, *SIAM Review*, **53, 2**, 217–288, 2011.

[M11] M. W. Mahoney, Randomized Algorithms for Matrices and Data, *Foundations and Trends in Machine Learning*, NOW Publishers, **3, 2**, 2011. Preprint: arXiv:1104.5557 (2011)

[P01] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*,
Birkhäuser/Springer, Boston/New York, 2001.

[PQY15] V. Y. Pan, G. Qian, X. Yan, Random Multipliers Numerically Stabilize Gaussian and Block Gaussian Elimination: Proofs and an Extension to Low-rank Approximation, *Linear Algebra and Its Applications*, **481**, 202–234, 2015.

[PSZ15] V. Y. Pan, J. Svadlenka, L. Zhao, Estimating the Norms of Circulant and Toeplitz Random Matrices and Their Inverses, *Linear Algebra and Its Applications*, **468**, 197–210, 2015.

[PSZa] V. Y. Pan, J. Svadlenka, L. Zhao, Low-rank Approximation: New Insight and Opportunities, arXiv:1510.06142, 21 Oct 2015, revised December 2016.

[PSZb] V.Y.Pan, L.Zhao, Fast CUR Approximation of Average Matrix and Extensions, preprint,2016.

[PZ16] V. Y. Pan, L. Zhao, Low-rank Approximation of a Matrix: Novel Insights, New Progress, and Extensions, *Lecture Notes in Computer Science*, **9691**, 352–366, Springer, 2016.

[PZ17] V. Y. Pan, L. Zhao, New Studies of Randomized Augmentation and Additive Preprocessing, *Linear Algebra and Its Applications*, 2017, http://dx.doi.org/10.1016/j.laa.2016.09.035.

[PZa] V. Y. Pan, L. Zhao, Numerically Safe Gaussian Elimination with No Pivoting, arxiv 1501.05385 CS, submitted on January 22, 2015, revised in June 2016.

[RST09] V. Rokhlin, A. Szlam, M. Tygert, A Randomized Algorithm for Principal Component Analysis, *SIAM Journal on Matrix Analysis and Applications*, **31, 3**, 1100–1124, 2009.

[S98] G. W. Stewart, *Matrix Algorithms, Vol I: Basic Decompositions*, SIAM, Philadelphia, 1998.

[T11] J. A. Tropp, Improved Analysis of the Subsampled Randomized Hadamard Transform, *Adv. Adapt. Data Anal.*, **3, 1–2**, 115–126, 2011. Also arXiv math.NA 1011.1595.

[TZ12] M. Thorup, Y. Zhang, Tabulation-based 5-independent Hashing with Applications to Linear Probing and Second Moment Estimation, *SIAM J. on Computing*, **41, 2**, 293–331, 2012.

[W14] D.P. Woodruff, Sketching As a Tool for Numerical Linear Algebra, *Foundations and Trends in Theoretical Computer Science*, **10, 1–2**, 1–157, 2014.

Chapter 3

LDR Neural Network

3.0.1 LDR Neural Networks

In this chapter we study application of LDR matrices in neural networks. Without loss of generality, we focus on a feed-forward neural network with one fully-connected (hidden) layer, which is similar to the network setup in [4]. Here the input layer (with n neurons) and the hidden layer (with kn neurons)[1] are assumed to be fully connected with a weight matrix $\mathbf{W} \in \mathbb{R}^{n \times kn}$ of displacement rank at most r corresponding to displacement operators $(\mathbf{A}, \mathbf{B})$, where $r \ll n$. The domain for the input vector $\mathbf{x}$ is the n-dimensional hypercube $I^n := [0,1]^n$, and the output layer only contains one neuron. The neural network can be expressed as:

$$y = G_{\mathbf{W},\boldsymbol{\theta}}(\mathbf{x}) = \sum_{j=1}^{kn} \alpha_j \sigma({\mathbf{w}_j}^T \mathbf{x} + \theta_j). \tag{3.0.1}$$

[1]Please note that this assumption does not sacrifice any generality because the n-by-m case can be transformed to n-by-kn format with the nearest k by using zero padding [3].

Here $\sigma(\cdot)$ is the activation function, $\mathbf{w}_j \in \mathbb{R}^n$ denotes the j-th column of the weight matrix $\mathbf{W}$, and $\alpha_j, \theta_j \in \mathbb{R}$ for $j = 1, ..., kn$. When the weight matrix $\mathbf{W} = [\mathbf{w}_1|\mathbf{w}_2|\cdots|\mathbf{w}_{kn}]$ has a low-rank displacement, we call it an LDR neural network. Matrix displacement techniques ensure that LDR neural network has much lower space requirement and higher computational speed comparing to classical neural networks of similar size.

3.0.2 Problem Statement

In this chapter, we aim at providing theoretical support on the accuracy of function approximation using LDR neural networks, which represents the "effectiveness" of LDR neural networks compared with the original neural networks. Given a continuous function $f(\mathbf{x})$ defined on $[0, 1]^n$, we study the following tasks:

- For any $\epsilon > 0$, find an LDR weight matrix $\mathbf{W}$ such that the function defined by equation (4) satisfies

$$\max_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - G_{\mathbf{W},\boldsymbol{\theta}}(\mathbf{x})| < \epsilon. \tag{3.0.2}$$

- Fix a positive integer n, find an upper bound ϵ so that for any continuous function $f(\mathbf{x})$ there exists a bias vector $\boldsymbol{\theta}$ and an LDR matrix with at most n rows satisfying equation (3.0.2).

- Find a multi-layer LDR neural network that achieves error bound (3.0.2) but with fewer parameters.

The first task is handled in Section 3.1, which is the *universal approximation property* of LDR neural networks. It states that the LDR neural networks could approximate any continuous function arbitrarily well, and this has widespread applications. The error bounds for shallow and deep neural networks are derived in Section 5. In addition, we derived explicit back-propagation expressions for LDR neural networks in Section 3.3.

3.1 The Universal Approximation Property of LDR Neural Networks

In this section we first prove a theorem for *matrix displacements*. Based on it, we prove the universal approximation property of neural networks utilizing only LDR matrices.

Theorem 3.1.1. *Let* $\mathbf{A}$, $\mathbf{B}$ *be two* $n \times n$ *non-singular diagonalizable matrices satisfying:*

i) $\mathbf{A}^q = a\mathbf{I}$ *for some positive integer* $q \leq n$ *and a scalar* $a \neq 0$*; ii)* $(\mathbf{I} - a\mathbf{B}^q)$ *is nonsingular; iii) the eigenvalues of* $\mathbf{B}$ *have distinct absolute values.*

Define S *as the set of matrices* $\mathbf{M}$ *such that* $\Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M})$ *has rank* 1*, i.e.,*

$$S_{\mathbf{A},\mathbf{B}} = \{\mathbf{M} \in \mathbb{R}^{n \times n} | \exists \mathbf{g}, \mathbf{h} \in \mathbb{R}^n, \Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M}) = \mathbf{g}\mathbf{h}^T\}. \tag{3.1.1}$$

Then for any vector $\mathbf{v} \in \mathbb{R}^n$*, there exists a matrix* $\mathbf{M} \in S_{\mathbf{A},\mathbf{B}}$ *and an index* $v \in \{1, ..., n\}$ *such that the i-th column of* $\mathbf{M}$ *equals vector* $\mathbf{v}$*.*

Proof. By the property of Stein displacement, any matrix $\mathbf{M} \in S$ can be expressed in terms of $\mathbf{A}$, $\mathbf{B}$, and its displacement as follows:

$$\mathbf{M} = \sum_{k=0}^{q-1} \mathbf{A}^k \Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M}) \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1}. \tag{3.1.2}$$

Here we use the property that $\Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M})$ has rank 1, and thus it can be written as $\mathbf{g} \cdot \mathbf{h}^T$. Since $\mathbf{A}$ is diagonalizable, one can write its eigen-decomposition as

$$\mathbf{A} = \mathbf{Q}^{-1} \mathbf{\Lambda} \mathbf{Q}, \tag{3.1.3}$$

where $\mathbf{\Lambda} = \mathbf{diag}(\lambda_1, ..., \lambda_n)$ is a diagonal matrix generated by the eigenvalues of $\mathbf{A}$. Now define $\mathbf{e}_j$ to be the j-th unit column vector for $j = 1, ..., n$. Write

$$\begin{aligned} \mathbf{Q}\mathbf{M}\mathbf{e}_j =& \mathbf{Q} \sum_{k=0}^{q-1} \mathbf{A}^k \Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M}) \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1} \mathbf{e}_j \\ =& \mathbf{Q} \sum_{k=0}^{q-1} (\mathbf{Q}^{-1} \mathbf{\Lambda} \mathbf{Q})^k \mathbf{g} \mathbf{h}^T \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1} \mathbf{e}_j \\ =& \Big(\sum_{k=0}^{q-1} s_{\mathbf{h},j} \mathbf{\Lambda}^k \Big) \mathbf{Q}\mathbf{g}. \end{aligned} \tag{3.1.4}$$

Here $s_{\mathbf{h},j}$ denotes the scalar value $\mathbf{h}^T \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1} \mathbf{e}_j$ for $k = 1, ..., n$. Define $\mathbf{T} := (\mathbf{I} - a\mathbf{B}^q)^{-1}$. In order to prove the theorem, we need to show that there exists a vector $\mathbf{h}$ and an index k such that the matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j} \mathbf{\Lambda}^k$

is nonsingular. In order to distinguish scalar multiplication from matrix multiplication, we write $a \circ \mathbf{M}$ to denote the product of a scalar a and a matrix M. Rewrite the expression as

$$\begin{aligned}
&\sum_{k=0}^{q-1} s_{\mathbf{h},j}\mathbf{\Lambda}^k \\
=&\sum_{k=0}^{q-1} \mathbf{h}^T \cdot \big(\mathbf{B}^k\mathbf{T}\mathbf{e}_j \circ \mathbf{diag}(\lambda_1^k, ..., \lambda_n^k)\big) \\
=&\sum_{k=0}^{q-1} \mathbf{diag}(\mathbf{h}^T \cdot \mathbf{B}^k \cdot \mathbf{T} \cdot [\lambda_1^k\mathbf{e}_j| \cdots |\lambda_n^k\mathbf{e}_j]) \\
=&\mathbf{diag}\bigg(\mathbf{h}^T \cdot \Big(\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_1^k\mathbf{e}_j\Big), ..., \mathbf{h}^T \cdot \Big(\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_n^k\mathbf{e}_j\Big)\bigg).
\end{aligned}$$

The diagonal matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j}\Lambda^k$ is nonsingular if and only if all of its diagonal entries are nonzero. Let $\mathbf{b}_{ij}$ denote the column vector $\sum_{k=0}^{q-1} \mathbf{B}\mathbf{T}^k\lambda_i^k\mathbf{e}_j$. Unless for every j there is an index i_j such that $\mathbf{b}_{i_jj} = \mathbf{0}$, we can choose a vector $\mathbf{h}$ such that the resulting diagonal matrix is nonsingular. Next we prove by contradiction that the former case is not possible. Assume that there is a column $\mathbf{b}_{i_jj} = \mathbf{0}$ for every $j = 1, 2, \cdots, n$, we must have:

$$\begin{aligned}
\mathbf{0} =&[\mathbf{b}_{i_11}|\mathbf{b}_{i_22}| \cdots |\mathbf{b}_{i_nn}] \\
=&\Big[\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_{i_1}^k\mathbf{e}_1| \cdots |\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_{i_n}^k\mathbf{e}_n\Big] \\
=&\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T} \cdot \mathbf{diag}(\lambda_{i_1}^k, ..., \lambda_{i_n}^k).
\end{aligned}$$

Since $\mathbf{B}$ is diagonalizable, we write $\mathbf{B} = \mathbf{P}^{-1}\mathbf{\Pi}\mathbf{P}$, where $\mathbf{\Pi} = \mathbf{diag}(\eta_1, ..., \eta_n)$. Also we have $\mathbf{T} = (\mathbf{I} - a\mathbf{B}^q)^{-1} = \mathbf{P}^{-1}(\mathbf{I} - a\mathbf{\Pi}^q)^{-1}\mathbf{P}$. Then

$$\begin{aligned}
\mathbf{0} &= \sum_{k=0}^{q-1} \mathbf{B}\mathbf{T}^k \mathbf{diag}(\lambda_{i_1}^k, ..., \lambda_{i_n}^k) \\
&= \mathbf{P}^{-1}\Big[\sum_{k=0}^{q-1} \mathbf{\Pi}^k (\mathbf{I} - a\mathbf{\Pi}^q)^{-1} \mathbf{diag}(\lambda_{i_1}^k, ..., \lambda_{i_n}^k)\Big]\mathbf{P} \\
&= \mathbf{P}^{-1} \sum_{k=0}^{q-1} \mathbf{diag}\Big((\lambda_{i_1}\eta_1)^k, ..., (\lambda_{i_n}\eta_n)^k\Big)(\mathbf{I} - a\mathbf{\Pi}^q)^{-1}\mathbf{P} \\
&= \mathbf{P}^{-1}\mathbf{diag}\Big(\sum_{k=0}^{q-1}(\lambda_{i_1}\eta_1)^k, ..., \sum_{k=0}^{q-1}(\lambda_{i_n}\eta_n)^k\Big)(\mathbf{I} - a\mathbf{\Pi}^q)^{-1}\mathbf{P}.
\end{aligned}$$

This implies that $\lambda_{i_1}\eta_1, ..., \lambda_{i_n}\eta_n$ are solutions to the equation

$$1 + x + x^2 + \cdots + x^{q-1} = 0. \tag{3.1.5}$$

By assumption of matrix $\mathbf{B}$, $\eta_1, ..., \eta_k$ have different absolute values, and so are $\lambda_{i_1}\eta_1, ..., \lambda_{i_1}\eta_1$, since all λ_k have the same absolute value because $\mathbf{A}^q = a\mathbf{I}$. It would follow that there are q distinct solutions of equation (3.1.5), but this is impossible, and so the matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j}\mathbf{\Lambda}^k$ cnnot be singular for all $\mathbf{h} \in \mathbb{R}^n$. With this property proven, given any vector $\mathbf{v} \in \mathbb{R}^n$, one can take the following procedure to find a matrix $\mathbf{M} \in S$ and an index j such that the j-th column of $\mathbf{M}$ equals $\mathbf{v}$:

i) Find a vector $\mathbf{h}$ and a index j such that matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j}\Lambda^k$ is non-singular;

ii) By equation (3.1.4), find

$$\mathbf{g} := \mathbf{Q}^{-1}\Big(\sum_{k=0}^{q-1} s_{\mathbf{h},j}\mathbf{\Lambda}^k\Big)^{-1}\mathbf{QTv};$$

iii) Construct $\mathbf{M} \in S$ with $\mathbf{g}$ and $\mathbf{h}$ defined by equation (3.1.2). Then the j-th column of M is equal to $\mathbf{v}$.

With the above construction, we have shown that for any vector $\mathbf{v} \in \mathbb{R}^n$ one can find a matrix $\mathbf{M} \in S$ and an index j such that the j-th column of $\mathbf{M}$ equals $\mathbf{v}$; thus the theorem is proved. □

Our main goal of this section is to show that neural networks with many types of LDR matrices (LDR neural networks) can approximate continuous functions arbitrarily well. In particular, we are going to show that Toeplitz and circulant matrices, being specific cases of LDR matrices, have this property. In order to do so, we need to introduce the following definition of a *discriminatory* function and a key property (cf. [4])

Definition 3.1.1. *A function $\sigma(u) : \mathbb{R} \to \mathbb{R}$ is called as discriminatory if the zero measure is the only measure μ that satisfies the following property:*

$$\int_{I^n} \sigma(\mathbf{w}^{\mathbf{T}}\mathbf{x} + \theta)d\mu(\mathbf{x}) = 0, \forall \mathbf{w} \in \mathbb{R}^n, \theta \in \mathbb{R}. \tag{3.1.6}$$

Lemma 3.1.1. *Any bounded, measurable sigmoidal function is discriminatory.*

Proof. The statement of this lemma and its proof is included in [4]. □

Now we are ready to present the universal approximation theorem of LDR neural networks with n-by-kn weight matrix $\mathbf{W}$:

Theorem 3.1.2 (Universal Approximation Theorem for LDR Neural Networks)**.** *Let σ be any continuous discriminatory function. For any continuous function $f(\mathbf{x})$ defined on I^n, $\epsilon > 0$, and any $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$, satisfying assumptions in Theorem 3.1.1, then there exists a function $G(\mathbf{x})$ in the form of equation* (3.0.1) *so that its weight matrix consists of k submatrices with displacement rank of* 1 *and*

$$\max_{\mathbf{x}\in I^n} |G(\mathbf{x}) - f(\mathbf{x})| < \epsilon. \tag{3.1.7}$$

Proof. Denote the i-th $n \times n$ submatrix of $\mathbf{W}$ as $\mathbf{W}_i$. Then $\mathbf{W}$ can be written as

$$\mathbf{W} = \big[\mathbf{W}_1|\mathbf{W}_2|...|\mathbf{W}_k\big]. \tag{3.1.8}$$

Let $\mathbf{M}$ be any of submatirx $\mathbf{W}_i$ with displacement rank 1. $\mathbf{M}$ can be written as

$$\Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M}) = \mathbf{M} - \mathbf{AMB} = \mathbf{g} \cdot \mathbf{h}^{\mathbf{T}}, \tag{3.1.9}$$

where $\mathbf{g}, \mathbf{h} \in \mathbb{R}^n$.

Since $\mathbf{A}^q = \mathbf{I}$, then obtain

$$\mathbf{M} = \Big[\sum_{k=0}^{q-1} \mathbf{A}^k \Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M})\mathbf{B}^k\Big](\mathbf{I} - a\mathbf{B}^q)^{-1}. \tag{3.1.10}$$

Let S_{I^n} denote the set of all continuous functions defined on I^n. Let U_{I^n} be the linear subspace of S_{I^n} that can be expressed in form of equation (3.0.1) where $\mathbf{W}$ consists of k sub-matrices with displacement rank 1. We want to show that U_{I^n} is dense in the set of all continuous functions S_{I^n}.

Suppose it is not dense. Then, by virtue of the Hahn-Banach Theorem, there exists a bounded linear functional $L \neq 0$ such that $L(\bar{U}(I^n)) = 0$. Moreover, by by virtue of the Riesz Representation Theorem, L can be written as

$$L(h) = \int_{I^n} h(\mathbf{x}) d\mu(\mathbf{x}), \forall h \in S(I^n),$$

for some measure μ.

Next we show that for any $\mathbf{y} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, the function $\sigma(\mathbf{y}^\mathbf{T}\mathbf{x} + \theta)$ belongs to the set U_{I^n}, and thus we must have

$$\int_{I^n} \sigma(\mathbf{y}^\mathbf{T}\mathbf{x} + \theta) d\mu(\mathbf{x}) = 0. \tag{3.1.11}$$

For any vector $\mathbf{y} \in \mathbb{R}^n$, Theorem 3.1.1 guarantees that there exists an $n \times n$ LDR matrix $\mathbf{M} = [\mathbf{b}_1|\cdots|\mathbf{b}_n]$ and an index j such that $\mathbf{b}_j = \mathbf{y}$. Now define

a vector $(\alpha_1, ..., \alpha_n)$ such that $\alpha_j = 1$ and $\alpha_1 = \cdots = \alpha_n = 0$. Also let the value of all bias be θ. Then the LDR neural network function becomes

$$\begin{aligned} G(\mathbf{x}) =& \sum_{i=1}^{n} \alpha_i \sigma(\mathbf{b_i^T x} + \theta) \\ =& \alpha_j \sigma(\mathbf{b_j^T x} + \theta) = \sigma(\mathbf{y^T x} + \theta). \end{aligned} \tag{3.1.12}$$

From the fact that $L(G(\mathbf{x})) = 0$, we derive that

$$\begin{aligned} 0 =& L(G(\mathbf{x})) \\ =& \int_{I^n} \sum_{i=1}^{n} \alpha_i \sigma(\mathbf{b_i^T x} + \theta) = \int_{I_n} \sigma(\mathbf{y^T x} + \theta) d\mu(\mathbf{x}). \end{aligned}$$

Since $\sigma(t)$ is a discriminatory function by Lemma 3.1.1. We can conclude that μ is the zero measure. As a result, the function defined as an integral with measure μ must be zero for any input function $h \in S(I^n)$. The last statement contradicts the property that $L \neq 0$ from the Hahn-Banach Theorem, which is obtained based on the assumption that the set U_{I^n} of LDR neural network functions are not dense in S_{I^n}. As this assumption is not true, we have the universal approximation property of LDR neural networks. □

The papers [3], [15] have utilized a circulant matrix or a Toeplitz matrix for weight representation in deep neural networks. Please notice that in the general case of n-by-m weight matrices, either the more general Block-circulant matrices should be utilized or padding extra columns or rows of zeroes are needed [3]. Circulant matrices and Topelitz matrices are both

special form of LDR matrices, and thus we could apply the above universal approximation property of LDR neural networks and provide theoretical support for the use of circulant and Toeplitz matrices in [3], [15]. Although circulant and Toeplitz matrices have displacement rank of 2 instead of 1, the property of Theorem 3.1.1 still holds, as a Toeplitz matrix is completely determined by its first row and its first column (and a circulant matrix is completely determined by its first row.) Therefore we arrive at the following corollary.

Corollary 3.1.1. *Any continuous function can be arbitrarily approximated by neural networks constructed with Toeplitz matrices or circulant matrices (with padding or using Block-circulant matrices).*

3.2 Error Bounds on LDR Neural Networks

Having the universal approximation property proved, we seek error estimates for LDR neural networks. We prove that for LDR matrices defined by $O(n)$ parameters (n represents the number of rows and has the same order as the number of columns), the corresponding structured neural network is capable of achieving integrated squared error of order $O(1/n)$, where n is the number of parameters. This result is asymptotically equivalent to Barron's aforementioned result on general neural networks, indicating that there is essentially

no loss for restricting to LDR matrices.

The functions we would like to approximate are those who are defined on a n-dimensional ball $B_r = \{\mathbf{x} \in \mathbb{R}^n : |\mathbf{x}| \leq r\}$ such that $\int_{B_r} |\mathbf{x}||f(\mathbf{x})|\mu(d\mathbf{x}) \leq C$, where μ is an arbitrary measure normalized so that $\mu(B_r) = 1$. Let us denote this set Γ_{C,B_r}.

[1] considered the following set of bounded multiples of a sigmoidal function composed with linear functions:

$$G_\sigma = \{\alpha\sigma(\mathbf{y}^\mathbf{T}\mathbf{x} + \theta) : |\alpha| \leq 2C, \mathbf{y} \in \mathbb{R}^n, \theta \in \mathbb{R}\}. \tag{3.2.1}$$

He proved the following theorem:

Theorem 3.2.1 ([1])**.** *For every function in Γ_{C,B_r}, every sigmoidal function σ, every probability measure, and every $k \geq 1$, there exists a linear combination of sigmoidal functions $f_k(\mathbf{x})$ of the form*

$$f_k(\mathbf{x}) = \sum_{j=1}^{k} \alpha_j \sigma(\mathbf{y}_j^\mathbf{T}\mathbf{x} + \theta_j), \tag{3.2.2}$$

such that

$$\int_{B_r} (f(\mathbf{x}) - f_k(\mathbf{x}))^2 \mu(d\mathbf{x}) \leq \frac{4r^2C}{k}. \tag{3.2.3}$$

Here $\mathbf{y}_j \in \mathbb{R}^n$ and $\theta_j \in \mathbb{R}$ for every $j = 1, 2, ..., N$, Moreover, the coefficients of the linear combination may be restricted to satisfy $\sum_{j=1}^{k} |c_j| \leq 2rC$.

Next we will obtain a similar result for LDR matrices. Fix operator $(\mathbf{A}, \mathbf{B})$ and define

$$\begin{aligned} S_\sigma^{kn} = \Big\{ \sum_{j=1}^{kn} \alpha_j \sigma(\mathbf{y}_j^\mathbf{T}\mathbf{x} + \theta_j) : |\alpha_j| \leq 2C, \mathbf{y}_j \in \mathbb{R}^n, \\ \theta_j \in \mathbb{R}, j = 1, 2, ..., N, \\ \textit{and } [\mathbf{y}_{(i-1)n+1}|\mathbf{y}_{(i-1)n+2}|\cdots|\mathbf{y}_{in}] \\ \textit{is an LDR matrix, } \forall i = 1, ..., k \Big\}. \end{aligned} \tag{3.2.4}$$

Moreover, let G_σ^k be the set of functions that can be expressed as a sum of no more than k terms from G_σ. Define the metric

$$||f - g||_\mu = \sqrt{\int_{B_r} (f(\mathbf{x}) - g(\mathbf{x}))^2 \mu(d\mathbf{x})}.$$

Theorem 3.2.1 essentially states that the minimal distance between a function $f \in \Gamma_{C,B}$ and G_σ^m is asymptotically $O(1/n)$. The following lemma proves that G_σ^k is in fact contained in S_σ^{kn}.

Lemma 3.2.1. *For any $k \geq 1$, $G_\sigma^k \subset S_\sigma^{kn}$.*

Proof. Any function $f_k(\mathbf{x}) \in G_\sigma^k$ can be written in the form

$$f_k(\mathbf{x}) = \sum_{j=1}^{k} \alpha_j \sigma(\mathbf{y}_j^\mathbf{T}\mathbf{x} + \theta_j). \tag{3.2.5}$$

For each $j = 1, ..., k$, define a $n \times n$ LDR matrix $\mathbf{W}_j$ such that one of its column is $\mathbf{y}_j$. Let $\mathbf{t}_{ij}$ be the i-th column of $\mathbf{W}_j$. Let i_j correspond to

the column index such that $\mathbf{t}_{ij} = \mathbf{y}_j$ for all j. Now consider the following function,

$$G(\mathbf{x}) := \sum_{j=1}^{k} \sum_{i=1}^{n} \beta_{ij} \sigma(\mathbf{t}_{ij}^{\mathbf{T}} \mathbf{x} + \theta_j), \tag{3.2.6}$$

where $\beta_{i_j j}$ equals α_j, and $\beta_{ij} = 0$ if $i \neq i_j$. Notice that we have the following equality,

$$\begin{aligned} G(\mathbf{x}) :=& \sum_{j=1}^{k} \sum_{i=1}^{n} \beta_{ij} \sigma(\mathbf{t}_{ij}^{\mathbf{T}} \mathbf{x} + \theta_j) \\ =& \sum_{j=1}^{k} \beta_{i_j j} \sigma(\mathbf{t}_{ij}^{\mathbf{T}} \mathbf{x} + \theta_j) \\ =& \sum_{j=1}^{k} \alpha_j \sigma(\mathbf{y}_j^{\mathbf{T}} \mathbf{x} + \theta_j) = f_k(\mathbf{x}). \end{aligned}$$

Notice that the matrix $\mathbf{W} = [\mathbf{W}_1 | \mathbf{W}_2 | \cdots | \mathbf{W}_k]$ consists of k LDR submatrices. Thus $f_k(\mathbf{x})$ belongs to the set S_σ^{kn}. □

By Lemma 3.2.1, we can replace G_σ^k with S_σ^{kn} in Theorem 3.2.1 and obtain the following error bound estimates on LDR neural networks:

Theorem 3.2.2. *For every disk $B_r \subset \mathbb{R}^n$, every function in Γ_{C,B_r}, every sigmoidal function σ, every normalized measure μ, and every $k \geq 1$, there exists a neural network defined by a weight matrix consisting of k LDR submatrices such that*

$$\int_{B_r} (f(\mathbf{x}) - f_{kn}(\mathbf{x}))^2 \mu(d\mathbf{x}) \leq \frac{4r^2 C}{k}. \tag{3.2.7}$$

Moreover, the coefficients of the linear combination may be restricted to satisfy $\sum_{k=1}^{N} |c_k| \leq 2rC$.

Since an $n \times n$ LDR matrix can be defined by $O(n)$ parameters, the space requirement of storing a matrix with $n \times kn$ LDR matrix is asymptotically the same as that of an $n \times k$ dense matrix. Since each $n \times n$ LDR matrix with displacement rank 1 can be determined by $2n$ parameters, function f_{kn} is defined by $2kn$ parameters, which is asymptotically the same with a classical neural network function defined by an $n \times k$ dense matrix.

The next theorem naturally extends the result from [12] to LDR neural networks, implying that LDR neural networks can benefit from parameter reduction if one uses more than one layers. More precisely, we have the following statement:

Theorem 3.2.3. *Let* f *be a continuous function on* $[0, 1]$ *and is* $2n+1$ *times differentiable in* $(0, 1)$ *for* $n = \lceil \log \frac{1}{\epsilon} + 1 \rceil \rceil$. *If* $|f^{(k)}(x)| \leq k!$ *holds for all* $x \in (0, 1)$ *and* $k \in \left[2n + 1\right]$, *then for any* $n \times n$ *matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *satisfying the conditions of Theorem 3.1.1, there exists a LDR neural network* $G_{\mathbf{A},\mathbf{B}}(x)$ *with* $O(\log \frac{1}{\epsilon})$ *layers,* $O(\log \frac{1}{\epsilon} n)$ *binary step units,* $O(\log^2 \frac{1}{\epsilon} n)$ *rectifier linear units such that*

$$\max_{x \in (0,1)} |f(x) - G_{\mathbf{A},\mathbf{B}}(x)| < \epsilon.$$

Proof. The theorem with better bounds and without assumption of being LDR neural network is proved in [12] as Theorem 4. For each binary step unit or rectifier linear unit in the construction of the general neural network, attach $(n-1)$ dummy units, and expand the weights associated to this unit from a vector to an LDR matrix based on Theorem 3.1.1. By doing so we need to add a factor n to the original amount of units, and the asymptotic bounds are relaxed accordingly. □

3.3 Training LDR Neural Networks

In this section, we reformulate the gradient computation of LDR neural networks. The computation for propagating through a fully-connected layer can be written as

$$\mathbf{y} = \sigma(\mathbf{W}^T\mathbf{x} + \boldsymbol{\theta}), \tag{3.3.1}$$

where $\sigma(\cdot)$ is the activation function, $\mathbf{W} \in \mathbb{R}^{n\times kn}$ is the weight matrix, $\mathbf{x} \in \mathbb{R}^n$ is input vector and $\boldsymbol{\theta} \in \mathbb{R}^{kn}$ is bias vector. According to Equation (7), if $\mathbf{W}_i$ is an LDR matrix with operators $(\mathbf{A}_i, \mathbf{B}_i)$ satisfying conditions of Theorem 3.1.1, then it is essentially determined by two matrices $\mathbf{G}_i \in \mathbb{R}^{n\times r}, \mathbf{H}_i \in \mathbb{R}^{n\times r}$ as

$$\mathbf{W}_i = \Big[\sum_{k=0}^{q-1} \mathbf{A}_i^k \mathbf{G}_i \mathbf{H}_i^T \mathbf{B}_i^k\Big](\mathbf{I} - a\mathbf{B}_i^q)^{-1}. \tag{3.3.2}$$

To fit the back-propagation algorithm, our goal is to compute derivatives $\frac{\partial O}{\partial \mathbf{G}_i}$, $\frac{\partial O}{\partial \mathbf{H}_i}$ and $\frac{\partial O}{\partial \mathbf{x}}$ for any objective function $O = O(\mathbf{W}_1, \ldots, \mathbf{W}_k)$.

In general, given that $\mathbf{a} := \mathbf{W}^{\mathbf{T}}\mathbf{x} + \boldsymbol{\theta}$, we have:

$$\frac{\partial O}{\partial \mathbf{W}} = \mathbf{x}(\frac{\partial O}{\partial \mathbf{a}})^T, \frac{\partial O}{\partial \mathbf{x}} = \mathbf{W}\frac{\partial O}{\partial \mathbf{a}}, \frac{\partial O}{\partial \boldsymbol{\theta}} = \frac{\partial O}{\partial \mathbf{a}}\mathbf{1}. \tag{3.3.3}$$

Here $\mathbf{1}$ is a column vector full of ones. Let $\hat{\mathbf{G}}_{ik} := \mathbf{A}_i^k\mathbf{G}_i$, $\hat{\mathbf{H}}_{ik} := \mathbf{H}_i^T\mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}$, and $\mathbf{W}_{ik} := \hat{\mathbf{G}}_{ik}\hat{\mathbf{H}}_{ik}$. The derivatives of $\frac{\partial O}{\partial \mathbf{W}_{ik}}$ can be computed as following:

$$\frac{\partial O}{\partial \mathbf{W}_{ik}} = \frac{\partial O}{\partial \mathbf{W}_i}. \tag{3.3.4}$$

According to Equation (3.3.3), if we let $\mathbf{a} = \mathbf{W}_{ik}$, $\mathbf{W} = \hat{\mathbf{G}}_{ik}^T$ and $\mathbf{x} = \hat{\mathbf{H}}_{ik}$, then $\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}}$ and $\frac{\partial O}{\partial \hat{\mathbf{H}}_{ik}}$ can be derived as:

$$\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}} = \big[\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}^T}\big]^T = \big[\hat{\mathbf{H}}_{ik}\frac{\partial O}{\partial \mathbf{W}_{ik}}\big]^T = (\frac{\partial O}{\partial \mathbf{W}_{ik}})^T\hat{\mathbf{H}}_{ik}^T, \tag{3.3.5}$$

$$\frac{\partial O}{\partial \hat{\mathbf{H}}_{ik}} = \hat{\mathbf{G}}_{ik}^T\frac{\partial O}{\partial \mathbf{W}_{ik}}. \tag{3.3.6}$$

Similarly, let $\mathbf{a} = \hat{\mathbf{G}}_{ik}$, $\mathbf{W} = (\mathbf{A}_i^k)^T$ and $\mathbf{x} = \mathbf{G}_i$, then $\frac{\partial O}{\partial \mathbf{G}_i}$ can be derived as:

$$\begin{aligned}\frac{\partial O}{\partial \mathbf{G}_i} &= \sum_{k=0}^{q-1}(\mathbf{A}_i^k)^T(\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}}) \\ &= \sum_{k=0}^{q-1}(\mathbf{A}_i^k)^T(\frac{\partial O}{\partial \mathbf{W}_{ik}})^T\hat{\mathbf{H}}_{ik}^T.\end{aligned} \tag{3.3.7}$$

Substituting with $\mathbf{a} = \hat{\mathbf{H}}_{ik}$, $\mathbf{W} = \mathbf{H}_i^T$ and $\mathbf{x} = \mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}$, we have $\frac{\partial O}{\partial \mathbf{H}_i}$ derived as:

$$\begin{aligned}\frac{\partial O}{\partial \mathbf{H}_i} &= \sum_{k=0}^{q-1} \mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}(\frac{\partial O}{\partial \hat{\mathbf{H}}_{ik}})^T \\ &= \sum_{k=0}^{q-1} \mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}(\frac{\partial O}{\partial \mathbf{W}_{ik}})^T \hat{\mathbf{G}}_{ik}.\end{aligned} \tag{3.3.8}$$

In this way, derivatives $\frac{\partial O}{\partial \mathbf{G}_i}$ and $\frac{\partial O}{\partial \mathbf{H}_i}$ can be computed given $\frac{\partial O}{\partial \mathbf{W}_{ik}}$, equal to $\frac{\partial O}{\partial \mathbf{W}_i}$. The essence of back-propagation algorithm is to propagate gradients backward from the layer with objective function to the input layer. $\frac{\partial O}{\partial \mathbf{W}_i}$ can be calculated from previous layer and $\frac{\partial O}{\partial \mathbf{x}}$ will be propagated to the next layer if necessary.

In practice one may choose matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ that can be multiplied by a vector fast, for example, diagonal matrices, permutation matrices, banded matrices, etc. Then the space complexity (the number of parameters for storage) of $\mathbf{W}_i$ can be $O(2n+2nr)$ rather than $O(n^2)$ in the case of usual dense matrix. The term $2n$ is associated with $\mathbf{A}_i$ and $\mathbf{B}_i$ and the term $2nr$ with $\mathbf{G}_i$ and $\mathbf{H}_i$. The time complexity of $\mathbf{W}_i^T\mathbf{x}$ is $O(q(3n+2nr))$ compared with $O(n^2)$ of dense matrix. Particularly, when $\mathbf{W}_i$ is a structured, e.g., Toeplitz matrix, the space complexity will be $O(2n)$. This is because the Toeplitz matrix is defined by $2n$ parameters. Moreover, its matrix-by-vector multiplication can be accelerated by using Fast Fourier Transform (for Toeplitz and circulant

matrices), resulting in time complexity $O(n \log n)$. In this way the back-propagation computation for the layer can be done in nearly linear time.

3.4 Conclusion

In this chapter, we have proven the universal approximation property of LDR neural networks. In addition, we also theoretically show that the error bounds of LDR neural networks are at least as efficient as in the case of general unstructured neural network. Besides, we also develop the back-propagation based training algorithm for universal LDR neural networks. Our study provides some theoretical foundation for the empirically observed success of the application of the LDR neural networks.

Bibliography

[1] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[2] D. Bini, V. Pan, and W. Eberly. Polynomial and matrix computations volume 1: Fundamental algorithms. *SIAM Review*, 38(1):161–164, 1996.

[3] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[7] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[8] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[9] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[10] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[12] S. Liang and R. Srikant. Why deep neural networks? *arXiv preprint arXiv:1610.04161*, 2016.

[13] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[14] V. Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

[15] V. Sindhwani, T. Sainath, and S. Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015.

[16] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[17] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

Chapter 4

Gaussian Elimination without Pivoting

4.1 BGE and GENP

For a nonsingular 2×2 block matrix $A = \begin{pmatrix} B & C \\ D & E \end{pmatrix}$ of size $n \times n$ with nonsingular $k \times k$ *pivot block* $B = A_{k,k}$, define $S = S(A_{k,k}, A) = E - DB^{-1}C$, the *Schur complement* of $A_{k,k}$ in A, and the block factorizations,

$$A = \begin{pmatrix} I_k & O_{k,r} \\ DB^{-1} & I_r \end{pmatrix} \begin{pmatrix} B & O_{k,r} \\ O_{r,k} & S \end{pmatrix} \begin{pmatrix} I_k & B^{-1}C \\ O_{k,r} & I_r \end{pmatrix} \tag{4.1.1}$$

and

$$A^{-1} = \begin{pmatrix} I_k & -B^{-1}C \\ O_{k,r} & I_r \end{pmatrix} \begin{pmatrix} B^{-1} & O_{k,r} \\ O_{r,k} & S^{-1} \end{pmatrix} \begin{pmatrix} I_k & O_{k,r} \\ -DB^{-1} & I_r \end{pmatrix}. \tag{4.1.2}$$

We verify readily that S^{-1} is the $(n-k) \times (n-k)$ trailing (that is, southeastern) block of the inverse matrix A^{-1}, and so the Schur complement S is nonsingular since the matrix A is nonsingular.

Factorization (4.1.2) reduces the inversion of the matrix A to the inversion of the leading block B and its Schur complement S, and we can recursively reduce the inversion task to the case of the leading blocks and Schur complements of decreasing sizes as long as the leading blocks are nonsingular. After sufficiently many recursive steps of this process of BGE, we only need to invert matrices of small sizes, and then we can stop the process and apply a selected black box inversion algorithm, e.g., based on orthogonalization.

In $\lceil \log_2(n) \rceil$ recursive steps all pivot blocks and all other matrices involved into the resulting factorization turn into scalars, all matrix multiplications and inversions turn into scalar multiplications and divisions, and we arrive at a *complete recursive factorization* of the matrix A. If $k = 1$ at all recursive steps, then the complete recursive factorization (4.1.2) defines GENP.

Moreover, any complete recursive factorizations turns into GENP up to the order in which we consider its steps. This follows because at most $n-1$ distinct Schur complements $S = S(A_{k,k}, A)$, for $k = 1, \dots, n-1$, are involved in all recursive block factorization processes for $n \times n$ matrices A, and so we arrive at the same Schur complement in a fixed position via GENP and via any other recursive block factorization (4.1.1). Hence we can interpret factorization step (4.1.1) as the block elimination of the first k columns of the matrix A, which produces the matrix $S = S(A_{k,k}, A)$. If the dimensions

$d_1, \dots, d_r$ and $\bar{d}_1, \dots, \bar{d}_{\bar{r}}$ of the pivot blocks in two block elimination processes sum to the same integer k, that is, if $k = d_1 + \cdots + d_r = \bar{d}_1 + \cdots + \bar{d}_{\bar{r}}$, then both processes produce the same Schur complement $S = S(A_{k,k}, A)$. The following results extend this observation.

Theorem 4.1.1. *In the recursive block factorization process based on (4.1.1), the diagonal block and its Schur complement in every block diagonal factor is either a leading block of the input matrix A or the Schur complement $S(A_{h,h}, A_{k,k})$ for some integers h and k such that $0 < h < k \le n$ and $S(A_{h,h}, A_{k,k}) = (S(A_{h,h}, A))_{h,h}$.*

Corollary 4.1.1. *The recursive block factorization process based on equation (4.1.1) can be completed by involving no singular pivot blocks (and, in particular, no pivot elements vanish) if and only if the input matrix A is strongly nonsingular.*

Proof. Combine Theorem 4.1.1 with the equation $\det A = (\det B) \det S$, implied by (4.1.1). □

Remark 4.1.1. *Applying BGE (unike GENP) we can use the benefits of block matrix algorithms.*

Remark 4.1.2. *One can benefit from application of BGE to computations with structured matrices. E.g., the MBA superfast algorithm, by Morf [M74],*

[M80] and by Bitmead and Anderson [BA80], runs in nearly linear arithmetic time in the case of Toeplitz and Toeplitz-like inputs. This algorithm is precisely the recursive BGE, accelerated by means of exploiting the Toeplitz-like structure of the input matrix extended throughout the recursive process of BGE (cf. Remark 4.3.1). The algorithm has been further extended to computations with structured matrices of other classes [P01, Chapter 5].

4.2 Preprocessing for GENP and BGE

In this section, A denotes a nonsingular $n \times n$ matrix.

Suppose that the vector $\mathbf{y} = A\mathbf{b}$ satisfies pre-processed linear systems $AH\mathbf{y} = \mathbf{b}$ and $FAH\mathbf{y} = F\mathbf{b}$. Then the vector $\mathbf{x} = H\mathbf{y}$ satisfies the linear system $A\mathbf{x} = \mathbf{b}$ as well as the pre-processed linear system $FA\mathbf{x} = F\mathbf{b}$. Next we estimate the efficiency of preprocessing for GENP and BGE with random and fixed post-multipliers H. Our analysis is immediately extended to preprocessings $A \rightarrow FA$, $A \rightarrow FAH$, and $A \rightarrow FAF^H$.

One can verify *by action* whether a fixed preprocessing works, by testing whether pre-processed GENP or BGE solves a linear system of equations $A\mathbf{x} = \mathbf{b}$, but for probabilistic analysis we proceed in circuited ways, based on Theorem 4.1.1 and Corollary 4.1.1.

4.2.1 Basic definitions and auxiliary results

We call GENP and BGE *safe* if they proceed to the end with no divisions by 0.

Corollary 4.1.1 implies the following result for computations in any field.

Theorem 4.2.1. *GENP is safe if and only if the input matrix is strongly nonsingular.*

Next assume that GENP and BGE are performed numerically, with rounding to a fixed precision, e.g., the IEEE standard double precision. Then extend the concept of safe GENP and BGE to *numerically safe GENP and BGE* by requiring that the input matrix be strongly nonsingular and *strongly well-conditioned*, that is, that the matrix itself and all its square leading blocks be nonsingular and well-conditioned (cf. item 1.4.13).

Any inversion algorithm for a nonsingular matrix is highly sensitive to both input and rounding errors if and only if the matrix is ill-conditioned [GL13], and likewise GENP is highly sensitive to the input and rounding errors if and only if some of the square leading blocks are ill-conditioned (see [PQZ13, Theorem 5.1] for quantitative version of these statements).

Remark 4.2.1. *BGE is safe if so does GENP. Likewise BGE is safe numerically if so does GENP. Thus our proofs of safety and numerical safety*

of GENP apply to BGE. The converse is not true, however. GENP fails (resp. fails numerically) if any leading square block of the input matrix is singular (resp. ill-conditioned), but BGE may by-pass this block and be safe (resp. numerically safe).

4.2.2 GENP with Gaussian preprocessing is likely to be numerically safe

Suppose that a nonsingular and well-conditioned matrix A has been pre-processed with a Gaussian multiplier H. Let us prove that the products FA and AH are likely to be numerically safe for GENP and BGE, in good accordance with the results of our tests reported in Section 4.5.

Theorem 4.2.2. *(Cf. [PQY15, Corollary 5.2].)*

Assume that we are given a nonsingular and well-conditioned $n \times n$ matrix A and a pair of $n \times n$ Gaussian matrices F and H. Then

(i) the matrices FA and AH are strongly nonsingular with a probability 1,

(ii) $\max_{k=1}^{n}\{||((AH)_{k,k})^+||, ((FA)_{k,k})^+||\} \le \nu_{n,k}^+/\sigma_n(A) = \nu_{n,k}^+||A^+||$ *for* $\nu_{n,k}^+$ *of Appendix A (cf. items 1.4.2, 1.4.4, and 1.4.9).*

Proof. The proof is similar for both products AH and FA; we only cover the case of the former one.

Part (i) follows from part (ii) of Theorem A.1.1 applied for $H = G$.

To prove part (ii), note that $(AH)_{k,k} = A_{k,n}H_{n,k}$, substitute SVD $A_{k,n} = S_{k,n}\Sigma T^T$ (cf. items 1.4.2 and 1.4.9), and obtain $(AH)_{k,k} = S_{k,n}\Sigma T^T H_{n,k} = S_{k,n}\Sigma G_{n,k}$ where $G_{n,k} = T^T H_{n,k}$ is an $n \times k$ Gaussian matrix by virtue of Lemma A.2.1 because $H_{n,k}$ is a Gaussian matrix and the matrix T is orthogonal. It follows that $((AH)_{k,k})^+ = G^+_{n,k}\Sigma^{-1}S^T_{k,n}$ (cf. item 1.4.5). Hence $||((AH)_{k,k})^+|| = ||G^+_{n,k}\Sigma^{-1}|| \le ||G^+_{n,k}||\ ||\Sigma^{-1}||$ because the matrix $S_{k,n}$ is orthogonal. Substitute the equations $||G^+_{n,k}|| = \nu^+_{n,k}$ and $||\Sigma^{-1}|| = 1/\sigma_n(A) = ||A^+||$ (cf. items 1.4.7 and 1.4.11). □

Theorems 4.2.2, A.2.1, and A.2.2 together imply the following primal and dual results (i) and (ii).

Corollary 4.2.1. *(i) Suppose GENP and BGE have been applied to a matrix A pre-processed with a Gaussian multiplier F or H or with a pair of Gaussian multipliers F and H. Then these algorithms are safe with probability 1 if and only if the matrix A is nonsingular. Moreover they are likely to be also numerically safe if this matrix is nonsingular and well-conditioned.*

(ii) GENP and BGE are safe and numerically safe when they are applied to average input matrix defined under the Gaussian probability distribution and pre-processed with any fixed nonsingular and well-conditioned multiplier.

Remark 4.2.2. *The upper estimates of Theorem A.2.2 for $\nu_{n,k}$ are strengthened dramatically as the integer $n-k$ increases. Hence we can strengthen numerical safety of recursive BGE by stopping the recursive process when it reduces the factorization task to the inversion of matrices of size $h \times h$ for a fixed small positive h, say, $h = 4$. Then we can invert these matrices of small size with no numerical problems and at a low computational cost, e.g., by applying orthogonalization. We call this technique* incomplete BGE. *Likewise, we can stop GENP when we reduce the factorization task to the case of $h \times h$ matrix, then apply orthogonalization, and arrive at* incomplete GENP.

4.2.3 Recursive block preprocessing for GENP

We can pre-process an $n \times n$ input matrix with Gaussian multipliers by using fewer random parameters and arithmetic operations if we proceed recursively. At first pre-process the $k \times k$ leading block of the input matrix for a proper integer $k < n$ by using $n \times k$ Gaussian multipliers. Having factored this block, we decrease the input size from n to $n-k$, and then we can re-apply Gaussian preprocessing. Already by using such a two-step block preprocessing for $k = n/2$, we save $1/4$ of all random parameters and $3/8$ of arithmetic operations involved, but this also yields an additional benefit. Namely, recall the bound $||((AG)_{k,k})^+|| \le \nu^+_{n,k}/\sigma_n(A)$ of part (ii) of Theorem 4.2.2 for the $k \times k$ leading

block of an input matrix A. The factor $1/\sigma_n(A)$ on the right-hand side is fixed for all k, but the factor $\nu^+_{n,k}$ is expected to decrease fast as k decreases from n, implying smaller expected residual norm of the output approximation.

One can apply this recipe to the case where $k = n-4$, say, and then solve the remaining linear system of four equations by applying some numerically stable methods, e.g., based on orthogonalization (cf. Remark 4.2.2).

4.2.4 Choice of multipliers. Universal sets of multipliers. Random versus fixed multipliers. Random universal sets

If we fix a nonsingular and well-conditioned multiplier H, then part (ii) of Corollary 4.2.1 implies that GENP and BGE are safe and numerically safe for most of nonsingular and well-conditioned input matrices A. This applies even to $H = I_n$, that is, to GENP and BGE with no preprocessing.

For any fixed multiplier H, however, there exist bad nonsingular and well-conditioned input matrices A, such that GENP and BGE are unsafe or numerically unsafe for the input AH. Such a bad matrix A can be a typical input of some actual computations, but by virtue of Corollary 4.2.1, a nonsingular and well-conditioned input can be bad only for a small fraction of all multipliers.

This suggests the following policy of *testing a family of multipliers by*

action for a fixed input: *successively or concurrently* apply multipliers from this family (and possibly also vary the policy of pre-, post- and two-sided preprocessing) and stop as soon as you succeed in any of these applications, that is, when you observe a small relative residual norm $||A\mathbf{x}-\mathbf{b}||/||(A \mid \mathbf{x})||$ or a small value of the growth factor in the LU factorization of the matrix FA, AH, FAH, or FAF^H.

Can we choose a small *universal set* of multipliers, with which such a test would succeed for any input? Clearly, we are motivated to choose structured or sparse multipliers such that we can generate them and multiply them by an input matrix at a low cost. Can we choose a small universal set made up of sparse and structured multipliers? So far even the problem of constructing any small universal set of multipliers is open.

By virtue of part (i) of Corollary 4.2.1, we would succeed with a probability close to 1 for any input if we choose a random multiplier from a family of all matrices under the Gaussian probability distribution, and so such a family is a *random universal set* of multipliers. This set is not small, however, its multipliers are not sparse or structured, and the above problems remain open even if, instead of fixing a multiplier, we allow to choose it at random from a fixed small family of matrices.

In Sections 4.3.5, 4.4, and **??**, we present our results about random univer-

sal sets made up of structured matrices, but right away we note the following *dilemma of random versus fixed preprocessing* for such sets: which highly unlikely failure should the user try to avoid more – with random (e.g., Gaussian) or fixed preprocessing? Indeed, for actual computations, the user can be satisfied with non-universal preprocessing as long as it covers the input sets of interest.

4.3 Some classes of structured multipliers

4.3.1 What kind of multipliers do we seek?

We seek multipliers F and H that support both safety and numerical safety of GENP, and according to Remark 4.2.1, their support for GENP can always be readily extended to BGE.

1. Multipliers F and H must be nonsingular and well-conditioned.

2. The relative residual norm of the output should be small when GENP is applied to a pre-processed matrix.

3. The cost of the computation of the product FA, AH, FAH or FAF^H should be small.

4. Random multipliers should be generated by using fewer random parameters, and properties 1 and 2 are only required for them with a

probability 1 or close to 1.

5. In the case of structured input matrices, the multipliers should have consistent structure.

4.3.2 Structured multipliers for safe GENP

If we only require safety of GENP, but not necessarily its numerical safety, then our requirements 1 and 2 to the multipliers can be softhen accordingly.

Randomized multipliers that support safe GENP and BGE with probability 1, for any nonsingular input, have been found already in 1991 and covered in [BP94, Section 2.13]. Among them one-sided preprocessing with random Toeplitz multipliers of [KP91] is most efficient, but slightly inferior two-sided preprocessing with random triangular Toeplitz multipliers of [KS91] has become most popular.

In Section 4.4, we prove that even random circulant multipliers, involving fewer flops and random parameters (see Appendix C and [P01] for definitions) ensure safe GENP and BGE with probability 1 under Gaussian choice of these parameters or with a probability close to 1 under their uniform choice from a large set. Using circulant multipliers saves 50% of random parameters and enables a 4-fold (resp. 2-fold) acceleration of the preprocessing of [KS91] (resp. [KP91]).

4.3.3 Circulant multipliers

A circulant multiplier H is defined by the n entries of its first column, and then the computation of the product AH is reduced essentially to performing $3n$ FFTs at the nth roots of 1. This involves about $4.5n^2 \log(n)$ flops, for a general matrix A. In modern computational practice such a cost bound can still be dominated by the cost of pivoting. Furthermore parallel implementation of FFT is known to be highly efficient, in particular, when it is based on Application Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs), incorporating Butterfly Circuits [DE]. Moreover, in the important case of Toeplitz or Toeplitz-like matrix A and a circulant multiplier H, we can compute a standard displacement representation of the product AH by applying just $O(n \log(n))$ flops (cf. [P01]).

Thus the random circulant multipliers satisfy requirement 1 above with a probability close to 1 (according to Remarks C.0.2 and C.0.3). They also satisfy requirements 3 and 4 (see Corollary C.0.1) as well as requirement 5 for the matrix structure of Toeplitz type. Empirically they also satisfy requirement 2. This suggests that random circulant multipliers are good candidates also for supporting numerical GENP and BGE.

Remark 4.3.1. *(Cf. Remark 4.1.2.) In the presence of rounding errors,*

the MBA algorithm fails except for the inputs of small size because of severe numerical problems [B85], while pivoting is not an option for solving Toeplitz or Toeplitz-like linear systems of equations because it destroys the matrix structure. So preprocessing is badly needed in this case. Fortunately, preprocessing with appropriate randomized structured multipliers is likely to fix these problems, and similarly in the extensions of the MBA algorithm to computations with other structured matrices (cf. [P01, Sections 5.6 and 5.7]). In particular, random circulant multiplication keeps Toeplitz structure intact (cf. [P01, Chapters 4 and 5]), and the computation of a standard displacement representation of the product of a Toeplitz or Toeplitz-like $n \times n$ matrix by a circulant matrix only requires $O(n \log(n))$ arithmetic operations (cf. [P01]) or just $O(n)$ multiplications if the circulant matrices involved are represented by using the factorization of Theorem C.0.1.

4.3.4 Structured multipliers from the study of low-rank approximation

As we are going to see in Section 2.4.3, multipliers for GENP and BGE work similarly to the ones for low-rank approximation of a matrix. Next we accommodate for GENP some efficient structured multipliers developed for the latter task in the last decade.

This includes *subsample random Hadamard transforms* (hereafter *SRHT*),

subsample random Fourier transforms (hereafter *SRFT*), and the chains of random Givens rotations (see [HMT11, Sections 4.6 and 11] and [M11]). All of them are unitary up to scaling. In applications to low-rank approximation, they are $n \times l$ rectangular matrices where $l \leq n$, and typically $l \ll n$, but in application to GENP and BGE they are square matrices. These multipliers are represented as the products of basic matrices P, C, and D defined separately for each family of multipliers. Here P is a (random or fixed) permutation matrix, D is a (random or fixed) diagonal matrix (and so each matrix P or D has n nonzero entries), and C is a core matrix.

SRHT and SRFT multipliers are represented as the products DCP. Multipliers based on the chains of random Givens rotations are represented as the products

$$C = D''P''C'D'P'CD\Omega_n P. \tag{4.3.1}$$

Here the prime symbol $'$ indicates independent realization of a matrix, and here and hereafter $\Omega_n = (\omega^{ij})_{i,j=0}^{n-1}$ is the $n \times n$ matrix of discrete Fourier transform, DFT(n), where $n = 2^k$, for an integer $k \geq 0$, and $\omega = \omega_n = \exp(2\pi\sqrt{-1}/n)$ is a primitive nth root of 1 (cf. Definition B.0.1). Scaling by $1/\sqrt{n}$ preserves the condition number of the matrix Ω_n and turns it into a unitary matrix (cf. items 1.4.5 and 1.4.11).

Remark 4.3.2. *Instead of the complex matrix DFT(n), one can consider the real matrices of the sine, cosine or Hartley transforms (cf. [P01, Section 3.11]).*

Next we specify the core matrices for multipliers based on SRHT, SRFT, and equation (4.3.1).

In the case of an SRHT multiplier, $C = H_k$ is the matrix of *Hadamard transform*, also called *Walsh–Hadamard transform*. Next we define it recursively, for any pair of integers $d \geq 1$ and $k \geq d$. This generalizes the customary recursive definition, in which $d = k$:

$$H_j = \begin{pmatrix} H_{j-1} & H_{j-1} \\ H_{j-1} & -H_{j-1} \end{pmatrix},\ j = k, k-1, \dots, k-d+1;\ H_{k-d} = \begin{pmatrix} I_{2^{k-d}} & I_{2^{k-d}} \\ I_{2^{k-d}} & -I_{2^{k-d}} \end{pmatrix}. \tag{4.3.2}$$

In the case of an SRFT multiplier, C is the $n \times n$ matrix Ω_n of DFT(n) where $n = 2^k$, for an integer $k \geq 0$, but next we define that matrix recursively, similarly to equation (4.3.2), for any pair of integers $d \geq 1$ and $k \geq d$. This generalizes the customary recursive definition in which $d = k$ (cf. [P01, equation 2.3.1]):

$$\Omega_{2^j} = P_j\ \mathrm{diag}(\Omega_{2^{j-1}}, \Omega_{2^{j-1}})\ \mathrm{diag}(I_{2^{j-1}}, D_j) \begin{pmatrix} I_{2^{j-1}} & I_{2^{j-1}} \\ I_{2^{j-1}} & -I_{2^{j-1}} \end{pmatrix}. \tag{4.3.3}$$

Here P_j is the $2^j \times 2^j$ odd/even permutation matrix, such that $P_{2s}(\mathbf{u}) = \mathbf{v}$, $\mathbf{u} = (u_i)_{i=0}^{2s-1}$, $\mathbf{v} = (v_i)_{i=0}^{2s-1}$, $v_i = u_{2i}$, $v_{i+s} = u_{2i+1}$, $i = 0, 1, \dots, s-1$, and

$D_{2s} = \mathrm{diag}(\omega_{2s}^i)_{i=0}^{s-1}$.

A core matrix of any of the two classes SRHT and SRFT can be multiplied by a vector by using $O(n\log(n))$ flops.

In the case of the chains of random Givens rotations of (4.3.1),

$$C = G(1,2,\theta_1)G(2,3,\theta_2)\cdots G(n-1,n,\theta_{n-1}), \tag{4.3.4}$$

and $G(i,j,\theta)$ is the matrix of Givens rotation on $\mathbb{C}^n$ by the angle θ in the (i,j) coordinate plane (cf. [GL13]). This core matrix can be multiplied by a vector by using $O(n)$ flops, but multiplication by a vector of the factor Ω in equation (4.3.1) involves order of $n\log(n)$ flops.

4.3.5 Numerical GENP with structured multipliers: bad inputs

Randomized versions of the multipliers of the previuos section are random universal for the task of low-rank approximation, but not for supporting GENP. We have proved randomized universality of Gaussian multipliers for the latter task, but cannot extend the proof to any family of sparse and structured multipliers. Moreover, next we specify some inputs for which GENP fails numerically if it is not pre-processed at all or if it is pre-processed with any circulant or SRFT multiplier.

At first recall from [Pa] that GENP is numerically unsafe for the $n \times n$

unitary matrix $A = \frac{1}{\sqrt{n}}F$ and consequently for the inverse matrix $A^{-1} = \frac{1}{\sqrt{n}}\Omega_n^H$ as well, provided that n is a large integer and Ω_n denotes the matrix DFT(n) of discrete Fourier transform (which is unitary up to scaling by $1/\sqrt{n}$). Of course, one does not need to apply GENP in order to invert these matrices, but by extending this result of [Pa], we specify some hard inputs for numerical application of GENP pre-processed with any fixed circulant multiplier and consequently with a Gaussian circulant multiplier.

The proof in [Pa] can be readily extended to the matrices U_f, $\Omega_n R$, $R^H\Omega_n^H$, $U_f R$, and $R^H U_f^H$, for a complex f such that $|f| = 1$, the matrix U_f of Theorem C.0.1, and a unitary (e.g., permutation) matrix R. Next, based on these results, we prove that GENP is numerically unsafe also if it is pre-processed with some structured matrices of large sizes.

Theorem 4.3.1. *Assume that we are given a large integer n, a complex f such that $|f| = 1$, the $n \times n$ DFT matrix $\Omega = \Omega_n$, an $n \times n$ unitary (e.g., permutation) matrix R, and a circulant $n \times n$ matrix $Z_1(\mathbf{v}) = \Omega^{-1}D\Omega$ with diagonal matrix $D = \mathrm{diag}(g_j)_{j=1}^n$. Write $U_f = \Omega D(\mathbf{f})$ for $D(\mathbf{f})$ of Theorem C.0.1. Then application of GENP to the matrices $\Omega Z_f(\mathbf{v})$, $\Omega Z_f(\mathbf{v})R$, $Z_f(\mathbf{v})\Omega^H$, $R^H Z_f(\mathbf{v})\Omega^H$, $U_f Z_f(\mathbf{v})$, $U_f Z_f(\mathbf{v})R$, $Z_f(\mathbf{v})U_f^H$, and $R^H Z_f(\mathbf{v})U_f^H$ is numerically unsafe.*

Proof. At first recall that $\Omega Z_1(\mathbf{v}) = D\Omega$, by virtue of Theorem C.0.1, and that $(D\Omega)_{k,k} = D_{k,k}\Omega_{k,k}$. Then recall from [Pa] that GENP applied to the matrix Ω fails numerically, which occurs because there is a singular or ill-conditioned leading block $\Omega_{k,k}$ of the matrix Ω.

If the diagonal matrix D is nonsingular and well-conditioned, then so are its leading block $D_{k,k}$, and hence the block $(D\Omega)_{k,k} = D_{k,k}\Omega_{k,k}$ as well, because so is the matrix $\Omega_{k,k}$.

If the matrix $D_{k,k}$ is singular or ill-conditioned, then so are the matrices D and $D\Omega$ as well because Ω is a unitary matrix (up to scaling by $1/\sqrt{n}$). □

Up to scaling by a constant, the matrix $D\,\Omega\,R$, for a random permutation matrix R, is an $n \times n$ SRFT matrix from [HMT11, Sections 4.6 and 11.1], whose $n \times l$ and $l \times n$ blocks are extensively used in various randomized matrix computations, for $l < n$ and usually for $l \ll n$. Theorem 4.3.1 shows that GENP with an $n \times n$ SRFT multiplier is likely to fail numerically already for the identity input matrix.

4.3.6 Some simplified multipliers

By virtue of part (ii) of Corollary 4.2.1, even non-universal multipliers satisfying requirements 1–5 of Section 4.3 can be valuable. Next we simplify the generation of the multipliers of the previous subsection and their multiplica-

tion by input matrices.

Actually there can be some tradeoff between low cost of preprocessing and output accuracy, but for all of our previous and next multipliers, a single refinement iteration was always sufficient in our tests in order to match or to exceed the output accuracy of GEPP (see also some empirical data in [PQZ13], [BDHT13], [DDF14], and [PQY15] and see [H02, Chapter 12], [GL13, Section 3.5.3], and the references therein for detailed coverage of iterative refinement). A refinement iteration involves $O(n^2)$ flops versus the dominant cubic cost of $\frac{2}{3}n^3$ flops, involved in Gaussian elimination, but for small n quadratic cost of refinement can make up a large share of the overall cost. In our tests, however, preprocessing of GENP with most of our multipliers (unlike the PRBT multipliers of [BDHT13], [DDF14], and [BBBDD14]) achieved high output accuracy even without iterative refinement.

Next we describe some refined multipliers.

1. We can fill the first column $\mathbf{v}$ of a circulant multiplier with 0s, except for a small number $\mathrm{nz}(\mathbf{v})$ of its entries. Then we can compute the product AH by using at most $(2\mathrm{nz}(\mathbf{v})-1)n$ flops. We have neither formal nor empirical support for numerical safety of GENP and BGE with this preprocessing alone, but empirically we do observe such support when

preprocessing combines these and some other multipliers (cf. Table 4.5.6).

2. *Abridged SRHT multipliers* defined by equation (4.3.2) for a small positive integer d (possibly even for $d = 1$) are simple to generate and can be multiply by a vector and by an input matrix by using $O(n)$ and $O(n^2)$ flops, respectively.

3. The same comments apply to *abridged SRFT multipliers* defined by equation (4.3.3) for a small positive integer d.

4. The multipliers defined by the chains of random Givens rotations of (4.3.1) are simplified similarly and in particular can be multiplied by a vector and an input matrix by using $O(n)$ and $O(n^2)$ flops, respectively, if in the definition by equation (4.3.1) we remove the factor Ω_n or replace it by its abridged version defined by equation (4.3.3) for $n = 2^k$ and a small positive integer d.

5. An interesting class of multipliers for recursive two-sided preprocessing, with *Partial Random Butterfly Transforms* (hereafter we use the acronym *PRBTs*), was proposed ad hoc in Technical Reports of 1995 by Parker and Parker and Pierce, which still remain unpublished (cf. [PP95]). These multipliers have been improved, carefully implemented,

and then extensively tested in the paper [BDHT13]. For an $n \times n$ input matrix and for even $n = 2k$, that paper defines PRBT as follows,

$$B^{(n)} = \frac{1}{\sqrt{2}} \begin{pmatrix} R & S \\ R & -S \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} I_k & I_k \\ I_k & -I_k \end{pmatrix} \mathrm{diag}(R, S) \tag{4.3.5}$$

where R and S are random diagonal nonsingular matrices. The paper [BDHT13] defines multipliers F and H recursively by using PRBT blocks. According to [BDHT13], the two-sided recursive processes of depth $d = 2$ with PRBT blocks are "sufficient in most cases". In such processes $F = \mathrm{diag}(B_1^{(n/2)}, B_2^{(n/2)})B^{(n)}$ and the multiplier H is defined similarly. In the case of depth-d recursion, $d \geq 2$, each of the multipliers F and H is defined as the product of d factors made up of 2^j diagonal blocks of size $n/2^j \times n/2^j$, for $j = 0, \ldots, d-1$, each block of the same type as above.

The PRBT recursive preprocessing of depth d involves $2dn$ random values, and the computation of the product FAH involves $5dn^2$ flops. One needs just dn random values if $F = H^T$. Saving flops is not of the highest importance because in computational practice pivoting typically costs even more than using $n \times n$ circulant multipliers.

The PRBT-based preprocessing satisfies requirements 1, 3 and 4, listed in the beginning of this subsection, and empirically satisfies require-

ment 2 as well, but not requirement 5.

6. The left multipliers $(I_n + Z_{\pm}^T)^{-1}$ and the right ones $(I_n + Z_{\pm})^{-1}$ where $Z_{\pm}$ denote $n \times n$ matrices filled with 0s, except for the $n-1$ entries of the first subdiagonal filled with the values ± 1. These multipliers satisfy requirements 1, 3, and 4 as well as requirement 2 (empirically) and requirement 5 for rank structured matrices A of [VVM07], [EGH13]. The computation of the product FAH involves less than $(n-1)\mathrm{nz}(A)$ additions and subtractions where $\mathrm{nz}(M)$ denotes the number of nonzero entries of the matrix M, $\mathrm{nz}(M) \le n^2$ for an $n \times n$ matrix M. Furthermore we estimate readily that $\kappa(M) \le 2\sqrt{n}$ for $M = I + Z_{\pm}^T$ and $M = I + Z_{\pm}$, and so the multipliers are well-conditioned.

   We call the preprocessing $A \to (I_n + Z_{\pm}^T)^{-1}A$ *arithmetic partial pivoting* and the two-sided preprocessing $A \to (I_n + Z_{\pm}^T)^{-1}A(I_n + Z_{\pm})^{-1}$ *arithmetic complete pivoting* because these two policies emulate partial and complete pivoting, respectively. Namely, we choose a linear combination of row entries or both of row and column entries with coefficients $\pm$ instead of choosing their absolute maximum in row entries or both of row and column entries, respectively.

7. The inverses of tridiagonal Hermitian or real symmetric matrices T

are somewhat similar to the inverses of bidiagonal matrices. In particular one can solve both bidiagonal and tridiagonal linear systems of equations in linear arithmetic time ($n-1$ and $9n-8$, respectively) and in logarithmic parallel time, e.g., by applying the *cyclic reduction* algorithm. By choosing a column-diagonally dominant matrix T with diagonal filled with 1s and with off-diagonal entries having absolute values at most $h < 1/2$, we obtain that $||T||_1 \leq 1+2h$, $||T^{-1}||_1 \leq 1/(1-2h)$ (cf. [GL13, Theorem 4.1.2]), and so $\kappa_1(T^{-1}) = ||T||_1\ ||T^{-1}||_1 \leq (1+2h)/(1-2h) = 1+4h/(1-2h))$, which is at most 3 for $h = 1/4$.

8. A diagonal (or bidiagonal) plus rank-1 matrix or its inverse. Then again such a matrix can be multiplied by a vector by using $O(n)$ flops, and by choosing a matrix with dominant diagonal, we can bound its condition number as we desire.

We hope that our work will motivate searching for other efficient multipliers and amelioration of the multipliers listed above. The cost of generating random multipliers of the listed classes and their multiplication by the input matrix is significantly smaller than the cost of pivoting, and so we can consider these multipliers *basic building blocks* for multipliers F and H, which

we can compute as the products and/or sums of small numbers of these blocks. For example, we can choose H being the sum of two blocks of the form $(I + Z_{\pm}^T)^{-1}$ or $(I + Z_{\pm})^{-1}$ plus, possibly, a matrix cI_n. Adding the latter term for $c \geq 4n$, say, would ensure that the sum H is diagonally dominant and thus nonsingular and well-conditioned. To increase the power of our preprocessing, we can include the matrices of *diagonal scaling* and *permutation matrices* as our basic building blocks. Our tests in Section 4.5 show that such combinations can greatly increase the efficiency of preprocessing.

For a simple example of potential power of such combinations, one can explore variations of PRBT-based multipliers when one allows to use also non-costly permutation multipliers. With this modification, is it possible to stay with one-sided preprocessing and/or decrease the depth d of the recursion without losing the output accuracy?

Another challenge, linked to the dilemma of random versus fixed preprocessing, is in *limiting randomization*, by using fewer or no random parameters. For example, we can limit randomization of preprocessing $A \rightarrow Z_{\pm}^T A$, $A \rightarrow AZ_{\pm}$, or $A \rightarrow Z_{\pm}^T A Z_{\pm}$ by choosing at random just the signs $\pm$ of the $n-1$ nonzero subdiagonal entries of the matrix $Z_{\pm}$, and we can even fix the signs $\pm$ deterministically for all or most of these $n-1$ entries.

Table 4.3.1: Some basic classes of structured multipliers

multiplier	**parameters**	**multiplication cost**
circulant	n	$4.5n^2\log(n)+O(n^2)$ flops
sparse circulant	$\mathrm{nz}(\mathbf{v})$	$(2\mathrm{nz}(\mathbf{v})-1)n$ flops
PRBT-based	$2dn$	$5dn^2$ flops
inverse of bidiagonal	$n-1$	$(n-1)\mathrm{nz}(A)$ flops
core SRHT	0	$O(n\log(n))$ flops
abridged core SRHT	0	$O(n)$ flops
core SRFT	0	$O(n\log(n))$ flops
abridged core SRHT	0	$O(n)$ flops
core chain of Givens rotations	$6n$	$O(n)$ flops

4.4 Randomized circulant preprocessing for symbolic GENP and BGE

Application of GENP in symbolic computations faces no numerical problems and is safe (that is, encounters no divisions by 0) if and only if the input matrix is strongly nonsingular (cf. Theorem 4.2.1). Theorem A.1.1 implies that a Gaussian matrix is strongly nonsingular with probability 1 (over infinite fields) and that a uniform random matrix is strongly nonsingular with a probability close to 1 over finite fields of large cardinality. Therefore, for average input matrix A defined under these probability distributions, GENP and BGE are safe.

Our Theorem 4.4.1 and Corollary 4.4.1 imply that, with Gaussian circulant as well as uniform random circulant preprocessing, GENP and BGE

are likely to be safe universally, that is, for any nonsingular input matrix. We need more than two pages, not counting definitions, in order to prove Theorem 4.4.1, but this enables us to accelerate by a factor of four the preprocessing of [KS91], highly popular among the researchers in symbolic computations, and we also remove one half of random variables involved. Namely, preprocessing of [KS91] requires pre- and post-multiplication of an $n \times n$ input matrix A by an upper and a lower triangular Toeplitz matrices, respectively (cf. (**??**)), at the overall cost dominated by the cost of performing twelve DFT(n) per row of an input matrix A (see Remark C.0.1), and in addition one must generate $2n-1$ random values. We only need to post-multiply a matrix A by a single circulant matrix, at the cost dominated by the cost of performing three DFT(n) per row of an input matrix A, and we only generate n random parameters.

Theorem 4.4.1. *Suppose* $A = (a_{i,j})_{i,j=1}^{n}$ *is a nonsingular matrix,* $T = (t_{i-j+1})_{i,j=1}^{n}$ *is a Gaussian* f*-circulant matrix,* $B = AT = (b_{i,j})_{i,j=1}^{n}$*,* f *is a fixed complex number,* $t_1, \dots, t_n$ *are variables, and* $t_k = f t_{n+k}$ *for* $k = 0, -1, \dots, 1-n$*. Let* $B_{l,l}$ *denotes the* l*-th leading blocks of the matrix* B *for* $l = 1, \dots, n$*, and so* $\det(B_{l,l})$ *are polynomial in* $t_1, \dots, t_n$*, for all* l*,* $l = 1, \dots, n$*. Then neither of these polynomials vanishes identically in* $t_1, \dots, t_n$*.*

Proof. Fix a positive integer $l \leq n$. With the convention $\alpha_{k\pm n} = f\alpha_k$, for $k = 1, \ldots, n$, we can write

$$B_{l,l} = \Big(\sum_{k_1=1}^{n} \alpha_{k_1} t_{k_1}, \sum_{k_2=1}^{n} \alpha_{k_2+1} t_{k_2}, \ldots, \sum_{k_l=1}^{n} \alpha_{k_l+l-1} t_{k_l} \Big), \tag{4.4.1}$$

where α_j is the jth column of $A_{l,n}$. Let $a_{i,j+n} = f a_{i,j}$, for $k = 1, \ldots, n$, and readily verify that

$$b_{i,j} = \sum_{k=1}^{n} a_{i,j+k-1} t_k,$$

and so $\det(B_l)$ is a homogeneous polynomial in $t_1, \ldots, t_n$.

Now Theorem 4.4.1 is implied by the following lemma.

Lemma 4.4.1. *If* $\det(B_{l,l}) = 0$ *identically in all the variables* $t_1, \ldots, t_n$*, then*

$$\det(\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_l}) = 0 \tag{4.4.2}$$

for all l*-tuples of subscripts* $(i_1, \ldots, i_l)$ *such that* $1 \leq i_1 < i_2 < \cdots < i_l \leq n$.

Indeed let $A_{l,n}$ denote the block submatrix made up of the first l rows of A. Note that if (4.4.2) holds for all l-tuples of the subscripts $(i_1, \ldots, i_l)$ above, then the rows of the block submatrix $A_{l,n}$ are linearly dependent, but they are the rows of the matrix A, and their linearly dependence contradicts the assumption that the matrix A is nonsingular.

In the rest of this section we prove Lemma 4.4.1. At first we order the

l-tuples $I = (i_1, \ldots, i_l)$, each made up of l positive integers written in non-decreasing order, and then we apply induction.

We order all l-tuples of integers by ordering at first their largest integers, in the case of ties by ordering their second largest integers, and so on.

We can define the classes of these l-tuples up to permutation of their integers and congruence modulo n, and then represent every class by the l-tuple of nondecreasing integers between 1 and n. Then our ordering of l-tuples of ordered integers takes the following form, $(i_1, \ldots, i_l) < (i'_1, \ldots, i'_l)$ if and only if there exist a subscript j such that $i_j < i'_j$ and $i_k = i'_k$ for $k = j + 1, \ldots, l$.

We begin our proof of Lemma 4.4.1 with the following basic result.

Lemma 4.4.2. *It holds that*

$$\det(B_{l,l}) = \sum_{1 \le i_1 \le i_2 \le \cdots \le i_l \le n} a_{\prod_{j=1}^{l} t_{i_j}} \prod_{j=1}^{l} t_{i_j}$$

where a tuple $(i_1, \ldots, i_l)$ *may contain repeated elements,*

$$a_{\prod_{j=1}^{l} t_{i_j}} = \sum_{(i'_1, \ldots, i'_l)} \det(\alpha_{i'_1}, \alpha_{i'_2+1}, \ldots, \alpha_{i'_l+l-1}), \tag{4.4.3}$$

and $(i'_1, \ldots, i'_l)$ *ranges over all permutations of* $(i_1, \ldots, i_l)$.

Proof. By using (4.4.1) we can expand $\det(B_{l,l})$ as follows,

$$\begin{aligned}
\det(B_{l,l}) &= \det\Big(\sum_{k_1=1}^{n}\alpha_{k_1}t_{k_1}, \sum_{k_2=1}^{n}\alpha_{k_2+1}t_{k_2}, \ldots, \sum_{k_l=1}^{n}\alpha_{k_l+l-1}t_{k_l}\Big) \\
&= \sum_{i_1=1}^{n} t_{i_1} \det\Big(\alpha_{i_1}, \sum_{k=1}^{n}\alpha_{k+1}t_k, \ldots, \sum_{k_l=1}^{n}\alpha_{k_l+l-1}t_{k_l}\Big) \\
&= \sum_{i_1=1}^{n} t_{i_1} \sum_{i_2=1}^{n} t_{i_2} \det\Big(\alpha_{i_1}, \alpha_{i_2+1}, \sum_{k_2=1}^{n}\alpha_{k_2+2}t_{k_2}, \ldots, \sum_{k_l=1}^{n}\alpha_{k_l+l-1}t_{k_l}\Big) \\
&= \ldots \\
&= \sum_{i_1=1}^{n} t_{i_1} \sum_{i_2=1}^{n} t_{i_2} \cdots \sum_{i_l=1}^{n} t_{i_l} \det\left(\alpha_{i_1}, \alpha_{i_2+1}, \ldots, \alpha_{i_l+l-1}\right). \qquad (4.4.4)
\end{aligned}$$

Consequently the coefficient $a_{\prod_{j=1}^{l} t_{i_j}}$ of any term $\prod_{j=1}^{l} t_{i_j}$ is the sum of all determinants

$$\det\left(\alpha_{i'_1}, \alpha_{i'_2+1}, \ldots, \alpha_{i'_l+l-1}\right)$$

where $(i'_1, \ldots, i'_l)$ ranges over all permutations of $(i_1, \ldots, i_l)$, and we arrive at (4.4.3).

□

In particular, the coefficient of the term t_1^l is

$$a_{t_1 \cdot t_1 \cdot \cdots \cdot t_1} = \det(\alpha_1, \alpha_2, \ldots, \alpha_l)$$

. This coefficient equals zero because $B_{l,l}$ is identically zero, by assumption of lemma 4.4.1, and we obtain

$$\det(\alpha_1, \alpha_2, \ldots, \alpha_l) = 0. \qquad (4.4.5)$$

This is the basis of our inductive proof of Lemma 4.4.1. In order to complete the induction step, it remains to prove the following lemma.

Lemma 4.4.3. *Let $J = (i_1, \ldots, i_l)$ be a tuple such that $1 \leq i_1 < i_2 < \cdots < i_l \leq n$.*

Then J is a subscript tuple of the coefficient of the term $\prod_{j=1}^{l} t_{i_j - j + 1}$ in equation (4.4.3).

Moreover, J is the single largest tuple among all subscript tuples.

Proof. Hereafter $\det(\alpha_{i'_1}, \alpha_{i'_2+1}, \ldots, \alpha_{i'_l+l-1})$ is said to be *the determinant associated with the permutation* $(i'_1, \ldots, i'_l)$ of $(i_1, \ldots, i_l)$ in (4.4.3). Observe that $\det(\alpha_{i_1}, \ldots, \alpha_{i_l})$ is the determinant associated with $\mathcal{I} = (i_1, i_2 - 1, \ldots, i_l - l + 1)$ in the coefficient $a_{\prod_{j=1}^{l} t_{i_j - j + 1}}$.

Let $\mathcal{I}'$ be a permutation of $\mathcal{I}$. Then $\mathcal{I}'$ can be written as $\mathcal{I}' = (i_{s_1} - s_1 + 1, i_{s_2} - s_2 + 1, \ldots, i_{s_l} - s_l + 1)$, where $(s_1, \ldots, s_l)$ is a permutation of $(1, \ldots, l)$. The determinant associated with $\mathcal{I}'$ has the subscript tuple $\mathcal{J}' = (i_{s_1} - s_1 + 1, i_{s_2} - s_2 + 2, \ldots, i_{s_l} - s_l + l)$. j satisfies the inequality $j \leq i_j \leq n - l + j$ because by assumption $1 \leq i_1 < i_2 < \cdots < i_l \leq n$, for any $j = 1, 2, \ldots, l$. Thus, $i_{s_j} - s_j + j$ satisfies the inequality $j \leq i_{s_j} - s_j + j \leq n - l + j \leq n$, for any s_j. This fact implies that no subscript of $\mathcal{I}'$ is negative or greater than n.

Let $\mathcal{J}'' = (i_{s_{r_1}} - s_{r_1} + r_1, i_{s_{r_2}} - s_{r_2} + r_2, \ldots, i_{s_{r_l}} - s_{r_l} + r_l)$ be a permutation of $\mathcal{J}$ such that its elements are arranged in the nondecreasing order. Now suppose $\mathcal{J}'' \geq J$. Then we must have $i_{s_{r_l}} - s_{r_l} + r_l \geq i_l$. This implies that

$$i_l - i_{s_{r_l}} \leq r_l - s_{r_l}. \tag{4.4.6}$$

Observe that

$$l - s_{r_l} \leq i_l - i_{s_{r_l}} \tag{4.4.7}$$

because $i_1 < i_2 < \cdots < i_l$ by assumption. Combine bounds (4.4.6) and (4.4.7) and obtain that $l - s_{r_l} \leq i_l - i_{s_{r_l}} \leq r_l - s_{r_l}$ and hence $r_l = l$.

Apply this argument recursively for $l-1, \ldots, 1$ and obtain that $r_j = j$ for any $j = 1, \ldots, l$. Therefore $\mathcal{J} = \mathcal{J}'$ and $\mathcal{I}' = \mathcal{I}$. It follows that J is indeed the single largest subscript tuple. □

By combining Lemmas 4.4.2 and 4.4.3, we support the induction step of the proof of Lemma 4.4.1, which we summarize as follows:

Lemma 4.4.4. *Assume the class of l-tuples of l positive integers written in the increasing order in each l-tuple and write* $\det(I) = \det(\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_l})$ *if* $I = (\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_l})$.

Then $\det(I) = 0$ *provided that* $\det(J) = 0$ *for all* $J < I$.

Finally we readily deduce Lemma 4.4.2 by combining this result with equation (4.4.5). This completes the proof of Theorem 4.4.1. □

Corollary 4.4.1. *Assume any nonsingular $n \times n$ matrix A and a finite set $\mathcal{S}$ of cardinality $|\mathcal{S}|$. Sample the values of the n coordinates $v_1, \dots, v_n$ of a vector $\mathbf{v}$ at random from this set. Fix a complex f and define the matrix $H = Z_f(\mathbf{v})$ of size $n \times n$, with the first column vector $\mathbf{v} = (v_i)_{i=1}^n$. Then GENP and BGE are safe for the matrix AH*

(i) with a probability of at least $1 - 0.5(n-1)n/|\mathcal{S}|$ if the values of the n coordinates $v_1, \dots, v_n$ of a vector $\mathbf{v}$ have been sampled uniformly at random from a finite set $\mathcal{S}$ of cardinality $|\mathcal{S}|$ or

(ii) with probability 1 if these coordinates are i.i.d. Gaussian variables.

(iii) The same claims hold for the matrices FA and $F = H^T$.

Proof. Theorems 4.2.1, 4.4.1, and A.1.1 together imply parts (i) and (ii) of the corollary. By applying transposition, extend them to part (iii). □

4.5 Numerical Experiments

Numerical experiments have been designed by the first author and have been performed by Xiaodong Yan for Tables 4.5.1–4.5.4 and 4.6.1–4.6.3 and by the second author for the other tables. The test have been run by using

MATLAB in the Graduate Center of the City University of New York on a Dell computer with the Intel Core 2 2.50 GHz processor and 4G memory running Windows 7. Gaussian matrices have been generated by applying the standard normal distribution function randn of MATLAB. We refer the reader to [PQZ13], [PQY14], [PQY15], and [PQZa] for other extensive tests of GENP with randomized preprocessing and of Algorithm 2.3.1.

4.5.1 Tests for GENP

Tables 4.5.1–4.5.4 show the maximum, minimum and average relative residual norms $||A\mathbf{y}-\mathbf{b}||/||\mathbf{b}||$ as well as the standard deviation for the solution of 1000 linear system $A\mathbf{x}=\mathbf{b}$ with Gaussian vector $\mathbf{b}$ and $n\times n$ input matrix A for each n, $n=256,512,1024$. The linear systems have been solved by using GEPP, GENP, or GENP pre-processed with real Gaussian, real Gaussian circulant, and random circulant multipliers, each followed by a single loop of iterative refinement.

The tests have been applied to the matrices

$$A=\begin{pmatrix} A_k & B \\ C & D \end{pmatrix}, \tag{4.5.1}$$

with $k\times k$ blocks A_k, B, C and D, for $k=n/2$, scaled so that $||B||\approx||C||\approx||D||\approx 1$, the $k-4$ singular values of the matrix A_k were equal 1 and the

other ones were set to 0 (cf. [H02, Section 28.3]), and with Gaussian Toeplitz matrices B, C, and D, that is, with Toeplitz matrices of (**??**), each defined by the i.i.d. Gaussian entries of its first row and first column. (The norm $||A^{-1}||$ ranged from 2.2×10^1 to 3.8×10^6 in these tests.) In the tests covered in Table 4.5.4, the matrix A was set to equal Ω, the matrix of DFT(n). For preprocessing, either Gaussian or the unitary Gaussian circulant matrices $C = \Omega^{-1} D(\Omega \mathbf{v})\Omega$ have been used as multipliers, with $\mathbf{v} = (v_i)_{i=0}^{n-1}$, $v_i = \exp(2\pi\phi_i\sqrt{-1}/n)$ and n i.i.d. real Gaussian variables ϕ_i, $i = 0, \dots, n-1$ (cf. Theorem C.0.1 and Remark C.0.3).

As should be expected, GEPP has always produced accurate solutions, with average relative residual norms ranging from 10^{-12} to 7×10^{-13}, but GENP with no preprocessing has consistently produced corrupted output with relative residual norms ranging from 10^{-3} to 10^2 for the input matrices A of equation (4.5.1). Even much worse was the output accuracy when GENP with no preprocessing or with Gaussian circulant preprocessing was applied to the matrix $A = \Omega$. In all other cases, however, GENP with random circulant preprocessing and with a single loop of iterative refinement has produced solution with desired accuracy, matching the output accuracy of GEPP. Furthermore GENP has performed similarly when it was applied to a nonsingular and well-conditioned input pre-processed with a Gaussian

multiplier.

Table 4.5.1: Relative residual norms: GENP with Gaussian multipliers

dim.	**iter.**	**mean**	**max**	**min**	**std**
256	0	6.13×10^{-9}	3.39×10^{-6}	2.47×10^{-12}	1.15×10^{-7}
256	1	3.64×10^{-14}	4.32×10^{-12}	1.91×10^{-15}	2.17×10^{-13}
512	0	5.57×10^{-8}	1.44×10^{-5}	1.29×10^{-11}	7.59×10^{-7}
512	1	7.36×10^{-13}	1.92×10^{-10}	3.32×10^{-15}	1.07×10^{-11}
1024	0	2.58×10^{-7}	2.17×10^{-4}	4.66×10^{-11}	6.86×10^{-6}
1024	1	7.53×10^{-12}	7.31×10^{-9}	6.75×10^{-15}	2.31×10^{-10}

Table 4.5.2: Relative residual norms: GENP with Gaussian circulant multipliers

dim.	**iter.**	**mean**	**max**	**min**	**std**
256	0	8.97×10^{-11}	1.19×10^{-8}	6.23×10^{-13}	4.85×10^{-10}
256	1	2.88×10^{-14}	2.89×10^{-12}	1.89×10^{-15}	1.32×10^{-13}
512	0	4.12×10^{-10}	3.85×10^{-8}	2.37×10^{-12}	2.27×10^{-9}
512	1	5.24×10^{-14}	5.12×10^{-12}	2.95×10^{-15}	2.32×10^{-13}
1024	0	1.03×10^{-8}	5.80×10^{-6}	1.09×10^{-11}	1.93×10^{-7}
1024	1	1.46×10^{-13}	4.80×10^{-11}	6.94×10^{-15}	1.60×10^{-12}

We also tested GENP with additive preprocessing applied to the same $n \times n$ test matrices A of (4.5.1), but for $n = 32, 64, 128, 256$. In this case we applied GENP to the matrix $C = A - UV^T$ where U and V were $n \times h$ random Gaussian subcirculant matrices each defined by the n i.i.d. Gaussian entries of its first column and scaled so that $||A|| = 2||UV^T||$. Then we computed the solution $\mathbf{x}$ to the linear system $A\mathbf{x} = \mathbf{b}$ for a Gaussian vector

Table 4.5.3: Relative residual norms: GENP with circulant multipliers filled with ± 1

dim.	**iter.**	**mean**	**max**	**min**	**std**
256	0	2.37×10^{-12}	2.47×10^{-10}	9.41×10^{-14}	1.06×10^{-11}
256	1	2.88×10^{-14}	3.18×10^{-12}	1.83×10^{-15}	1.36×10^{-13}
512	0	7.42×10^{-12}	6.77×10^{-10}	3.35×10^{-13}	3.04×10^{-11}
512	1	5.22×10^{-14}	4.97×10^{-12}	3.19×10^{-15}	2.29×10^{-13}
1024	0	4.43×10^{-11}	1.31×10^{-8}	1.28×10^{-12}	4.36×10^{-10}
1024	1	1.37×10^{-13}	4.33×10^{-11}	6.67×10^{-15}	1.41×10^{-12}

Table 4.5.4: Relative residual norms: GENP for DFT(n) with Gaussian multipliers

dim.	**iter.**	**mean**	**max**	**min**	**std**
256	0	2.26×10^{-12}	4.23×10^{-11}	2.83×10^{-13}	4.92×10^{-12}
256	1	1.05×10^{-15}	1.26×10^{-15}	9.14×10^{-16}	6.76×10^{-17}
512	0	1.11×10^{-11}	6.23×10^{-10}	6.72×10^{-13}	6.22×10^{-11}
512	1	1.50×10^{-15}	1.69×10^{-15}	1.33×10^{-15}	6.82×10^{-17}
1024	0	7.57×10^{-10}	7.25×10^{-8}	1.89×10^{-12}	7.25×10^{-9}
1024	1	2.13×10^{-15}	2.29×10^{-15}	1.96×10^{-15}	7.15×10^{-17}

$\mathbf{b}$ by substituting the Sherman–Morrison–Woodbury formula (**??**) into the equation $\mathbf{x} = A^{-1}\mathbf{b}$.

We present the test results in Table 4.5.5. The results changed little when we scaled the matrices U and V to increase the ratio $||A||/||UV^T||$ to 10 and 100.

Finally we tested GENP with preprocessing by means of some core multipliers of Section 4.3 and some of their numerous possible combinations. We

Table 4.5.5: Relative residual norms of GENP with Gaussian subcirculant additive preprocessing

n	h	Refinement	**mean**	**std**
16	4	0	1.67e-11	7.87e-11
16	4	1	6.15e-15	2.92e-14
32	4	0	8.42e-11	4.86e-10
32	4	1	1.49e-14	9.09e-14
64	4	0	9.23e-11	3.99e-10
64	4	1	1.63e-14	4.24e-14
128	4	0	6.55e-10	2.42e-09
128	4	1	7.50e-13	2.99e-13
256	4	0	1.13e-08	3.38e-08
256	4	1	1.10e-12	6.23e-13

hope that our study will prompt further research in this direction. The test results are represented in Tables 4.5.6 and 4.5.7.

In this series of our tests we applied GENP to the above matrices of (4.5.1) and six families of benchmark matrices from [BDHT13], pre-processed with multipliers combining the ones of following three basic families.

Family 1: Three recursions of the matrices of DFT(n) with a single random permutation.

Family 2: Sparse circulant matrices $C = \Omega^{-1}D(\Omega\mathbf{v})\Omega$, where the vector $\mathbf{v}$ has been filled with 0s, except for its ten coordinates filled with ± 1. Here and hereafter each sign + or − has been assigned with probability 1/2.

Family 3: Sum of two inverse bidiagonal matrices. At first their main

diagonals have been filled with the integer 101, and their first subdiagonals have been filled with ± 1. Then each matrix have been multiplied by a diagonal matrix $\mathrm{diag}(\pm 2^{b_i})$, where b_i were random integers uniformly chosen from 0 to 3.

We tested GENP on ten combinations of these three basic families of multipliers, listed below. The size of the linear system was 128. For each combination we have performed 100 tests and have recorded the average relative error $||A\mathbf{x} - \mathbf{b}||/||\mathbf{b}||$ with matrices A from the seven benchmark families and vectors $\mathbf{b}$ being standard Gaussian vectors. Here are these ten combinations.

1. $F = I$, H is a matrix of Family 1.
2. $F = I$, H is a matrix of Family 3.
3. $F = H$ is a matrix of Family 1.
4. $F = H$ is a matrix of Family 3.
5. F is a matrix of Family 1, H is a matrix of Family 3.
6. $F = I$, H is the product of two matrices of Family 1.
7. $F = I$, H is the product of two matrices of Family 2.
8. $F = I$, H is the product of two matrices of Family 3.
9. $F = I$, H is the sum of two matrices of Families 1 and 3.
10. $F = I$, H is the sum of two matrices of Families 2 and 3.

We tested these multipliers for the same linear systems as in our previous tests in this section and for six classes generated from Matlab, by following their complete description in Matlab and [BDHT13]. Here is the list of these seven test classes.

1. The matrices A of (4.5.1).

2. 'circul': circulant matrices whose first row is a standard Gaussian random vector.

3. 'condex': counter-examples to matrix condition number estimators.

4. 'fiedler': symmetric matrices generated with (i, j) and (j, i) elements equal to $c_i - c_j$ where $c_1, \ldots, c_n$ are i.i.d. standard Gaussian variables.

5. 'orthog': orthogonal matrices with (i, j) elements $\sqrt{\frac{2}{n+1}} \sin \frac{ij\pi}{n+1}$.

6. 'randcorr': random $n \times n$ correlation matrices with random eigenvalues from a uniform distribution. (A correlation matrix is a symmetric positive semidefinite matrix with 1's on the diagonal.)

7. 'toeppd': $n \times n$ symmetric, positive semi-definite (SPSD) Toeplitz matrices T equal to the sums of m rank-2 SPSD Toeplitz matrices. Specifically,

$$T = w(1) * T(\theta(1)) + ... + w(m) * T(\theta(m))$$

where $\theta(k)$ are i.i.d. Gaussian variables and

$$T(\theta(k)) = (cos(2\pi(i-j)\theta(k)))_{i,j=1}^{n}$$

.

In our tests, for some pairs of inputs and multipliers, GENP has produced no meaningful output. In such cases we filled the respective entries of Tables 4.5.6 and 4.5.7 with ∞.

GENP pre-processed with our multipliers of the 9th combination of three basic families, has produced accurate outputs without iterative refinement for all seven benchmark classes of input matrices. With the other combinations of the three basic families of our multipliers, this was achieved from 4 to 6 (out of 7) benchmark input classes. For comparison, the 2-sided preprocessing with PRBT-based multipliers of [BDHT13] and [BBBDD14] always required iterative refinement.

4.5.2 Tests for sampling algorithm with Gaussian and random structured multipliers

Tables 4.6.1–4.6.4 show the results of testing Algorithm 2.3.1 for rank-r approximation of $n \times n$ matrices M for $n = 256, 512, 1024$ and $l = r = 8, 32$.

The input matrices M, having numerical rank r, have been defined by their SVDs, $M = S_M \Sigma_M T_M^T$, for S_M and T_M generated as $n \times n$ ran-

Table 4.5.6: Relative residual norms output by pre-processed GENP with no refinement iterations

class	1	2	3	4	5
1	2.61e-13	6.09e-15	∞	2.62e+02	7.35e-15
2	2.02e+02	4.34e-14	5.34e-16	∞	7.35e+02
3	4.34e-13	8.36e-15	∞	3.03e+02	1.94e-14
4	1.48e+01	1.36e-12	2.39e-16	1.01e-11	4.71e+01
5	3.71e-11	2.21e-14	∞	2.85e+01	5.83e-10
6	3.33e-13	9.36e-15	∞	3.66e-05	7.04e-15
7	7.76e-12	3.55e-14	9.91e+01	7.90e+00	7.75e+00
8	7.95e+00	9.55e-14	7.56e-16	∞	5.74e+03
9	5.36e-13	1.51e-14	4.26e-16	2.24e-11	3.68e-13
10	3.50e-12	8.43e-14	3.43e-13	2.90e-10	1.36e+01

dom orthogonal matrices by means of orthogonal factorization of $n \times n$ Gaussian matrices and for $\Sigma_M = \mathrm{diag}(\sigma_j)_{j=1}^{n}$, for $\sigma_j = 1/j,\ j = 1, \dots, r$, $\sigma_j = 10^{-10},\ j = r+1, \dots, n$ (cf. [H02, Section 28.3]). Hence $||M|| = 1$ and $\kappa(M) = 10^{10}$.

Tables 4.6.1–4.6.3 display the resulting data for the residual norms $rn = ||M - QQ^T M||$ in 1000 runs of the tests of Algorithm 2.3.1, applied to the above input matrices M for every pair of n and r provided that the random multipliers B have been generated as $n \times r$ real Gaussian matrices, real Gaussian subcirculant matrices (cf. Remark C.0.3), and subcirculant matrices filled with the values ± 1 whose signs $\pm$ have been chosen at random.

We have also run 1000 similar tests for multipliers of each of eight classes

Table 4.5.7: Relative residual norms output by pre-processed GENP followed by a single refinement iteration

class	1	2	3	4	5
1	1.13e-15	6.90e-17	∞	1.12e+00	5.23e-17
2	5.07e-04	7.71e-17	1.03e-16	∞	4.40e+02
3	1.14e-15	7.34e-17	∞	5.43e-13	5.15e-17
4	1.55e-03	6.19e-17	1.31e-16	5.69e-13	2.69e+02
5	9.80e-16	6.96e-17	∞	6.75e+01	5.35e-17
6	1.08e-15	6.13e-17	∞	6.35e-13	5.08e-17
7	3.47e+01	6.17e-17	2.61e+06	5.21e+00	5.31e-17
8	2.56e-04	6.67e-17	1.15e-16	∞	7.96e+02
9	9.81e-16	7.44e-17	3.99e-17	6.40e-13	5.09e-17
10	9.79e-16	8.32e-17	1.14e-16	7.34e-13	4.07e+01

below, generated from Families 1,2 and 3 of the previous subsection. Namely, we performed Algorithm 2.3.1 for low-rank approximation by using multipliers B obtained as the $n \times r$ western (that is, leftmost) blocks B of the following classes of $n \times n$ matrices.

1. B is a matrix from Family 1.
2. B is a matrix from Family 2.
3. B is a matrix from Family 3.
4. B is the product of two matrices of Family 1.
5. B is the product of two matrices of Family 2.
6. B is the product of two matrices of Family 3.
7. B is the sum of two matrices of Family 1 and 3.

8. B is the sum of two matrices of Family 2 and 3.

The average residual norms are displayed in Table 4.6.4. This shows that the algorithm with our preprocessing has consistently output close approximations to low-rank input matrices. Table 4.6.4 displays the results of the same tests with the multipliers of AH and ASPH transforms of Sections 4.3.6 and **??**. We show them separately from the tests of Table 4.6.4 because with these multipliers our preprocessing is performed at particularly low arithmetic cost (see Section **??**). Nevertheless the tests produced the results similar to the ones of Table 4.6.4.

4.6 More Test Results

Table 4.6.1: Residual norms rn in the case of using Gaussian multipliers

r	n	**mean**	**max**
8	256	7.54×10^{-8}	1.75×10^{-5}
8	512	4.57×10^{-8}	5.88×10^{-6}
8	1024	1.03×10^{-7}	3.93×10^{-5}
32	256	5.41×10^{-8}	3.52×10^{-6}
32	512	1.75×10^{-7}	5.57×10^{-5}
32	1024	1.79×10^{-7}	3.36×10^{-5}

Table 4.6.2: Residual norms rn in the case of using Gaussian subcirculant multipliers

r	n	**mean**	**max**
8	256	3.24×10^{-8}	2.66×10^{-6}
8	512	5.58×10^{-8}	1.14×10^{-5}
8	1024	1.03×10^{-7}	1.22×10^{-5}
32	256	1.12×10^{-7}	3.42×10^{-5}
32	512	1.38×10^{-7}	3.87×10^{-5}
32	1024	1.18×10^{-7}	1.84×10^{-5}

Table 4.6.3: Residual norms rn in the case of using subcirculant random multipliers filled with ± 1

r	n	**mean**	**max**
8	256	7.70×10^{-9}	2.21×10^{-7}
8	512	1.10×10^{-8}	2.21×10^{-7}
8	1024	1.69×10^{-8}	4.15×10^{-7}
32	256	1.51×10^{-8}	3.05×10^{-7}
32	512	2.11×10^{-8}	3.60×10^{-7}
32	1024	3.21×10^{-8}	5.61×10^{-7}

Table 4.6.4: Extended Low-rank Approximation Tests

n	r	class 1	class 2	class 3	class 4	class 5	class 6
256	8	5.94e-09	4.35e-08	2.64e-08	2.20e-08	7.73e-07	5.15e-09
256	32	2.40e-08	2.55e-09	8.23e-08	1.58e-08	4.58e-09	1.36e-08
512	8	1.11e-08	8.01e-09	2.36e-09	7.48e-09	1.53e-08	8.15e-09
512	32	1.61e-08	4.81e-09	1.61e-08	2.83e-09	2.35e-08	3.48e-08
1024	8	5.40e-09	3.44e-09	6.82e-08	4.39e-08	1.20e-08	4.44e-09
1024	32	2.18e-08	2.03e-08	8.72e-08	2.77e-08	3.15e-08	7.99e-09

Table 4.6.5: Extended Low-rank Approximation Tests Using AH and ASPH Transforms

n	r	Average Relative Norm	
		AH	ASPH
256	8	8.43e-09	4.89e-08
256	32	3.53e-09	5.47e-08
512	8	7.96e-09	3.16e-09
512	32	1.75e-08	7.39e-09
1024	8	6.60e-09	3.92e-09
1024	32	7.50e-09	5.54e-09

Bibliography

[B85] J. R. Bunch, Stability of Methods for Solving Toeplitz Systems of Equations, *SIAM Journal on Scientific and Statistical Computing*, **6, 2**, 349–364, 1985.

[BA80] R. R. Bitmead, B. D. O. Anderson, Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations, *Linear Algebra and Its Applications*, **34**, 103–116, 1980.

[BBBDD14] M. Baboulin, D. Becker, G. Bosilca, A. Danalis, J. Dongarra, An Efficient Distributed Randomized Algorithm for Solving Large Dense Symmetric Indefinite Linear Systems, *Parallel Computing (7th Workshop on Parallel Matrix Algorithms and Applications),* **40, 7**, 213–223, 2014.

[BBD12] D. Becker, M. Baboulin, J. Dongarra, Reducing the Amount of Pivoting in Symmetric Indefinite Systems, *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics, PPAM 2011, Lecture Notes in Computer Science*, **7203**, 133–142, Springer, 2012. Also *INRIA*, Research Report 7621 (05/2011) and *University of Tennessee*, Technical Report ICL-UT-11-06.

[BCD14] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, O. Schwartz, Communication Lower Bounds and Optimal Algorithms for Numerical Linear Algebra, *Acta Numerica* (an invited paper), **23**, 1–155, 2014.

[BDHT13] M. Baboulin, J. Dongarra, J. Herrmann, S. Tomov, Accelerating Linear System Solutions Using Randomization Techniques, *ACM Transactions on Mathematical Software (TOMS)*, **39, 2**, 2013.

[BLR15] M. Baboulin, X. S. Li, F-H. Rouet, Using Random Butterfly Transformations to Avoid Pivoting in Sparse Direct Methods, *Proceedings of VECPAR 2014, Lecture Notes in Computer Science*, **8969**, 135–144, Springer, 2015.

[BP94] D. Bini, V. Y. Pan, *Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms*, Birkhäuser, Boston, 1994.

[BV88] W. Bruns, U. Vetter, *Determinantal Rings, Lecture Notes in Math.,* **1327**, Springer, Heidelberg, 1988.

[CD05] Z. Chen, J. J. Dongarra, Condition Numbers of Gaussian Random Matrices, *SIAM. J. on Matrix Analysis and Applications*, **27**, 603–620, 2005.

[CPW74] R. E. Cline, R. J. Plemmons, G. Worm, Generalized Inverses of Certain Toeplitz Matrices, *Linear Algebra and Its Applications,* **8**, 25–33, 1974.

[D88] J. Demmel, The Probability That a Numerical Analysis Problem Is Difficult, *Math. of Computation*, **50**, 449–480, 1988.

[DDF14] S. Donfack, J. Dongarra, M. Faverge, M. Gates, J. Kurzak, P. Luszczek, I. Yamazaki,
A Survey of Recent Developments in Parallel Implementations of Gaussian Elimination, CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE, *Concurrency Computat.: Pract. Exper.* (2014) Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3306

[DE] *Dillon Engineering*, http://www.dilloneng.com/fft_ip/parallel-fft.

[DKM06] P. Drineas, R. Kannan, M.W. Mahoney, Fast Monte Carlo Algorithms for Matrices I–III, *SIAM J. on Computing*, **36, 1**, 132–206, 2006.

[DL78] R. A. Demillo, R. J. Lipton, A Probabilistic Remark on Algebraic Program Testing, *Information Processing Letters*, **7**, **4**, 193–195, 1978.

[DS01] K. R. Davidson, S. J. Szarek, Local Operator Theory, Random Matrices, and Banach Spaces, in *Handbook on the Geometry of Banach Spaces*

(W. B. Johnson and J. Lindenstrauss editors), pages 317–368, North Holland, Amsterdam, 2001.

[E88] A. Edelman, Eigenvalues and Condition Numbers of Random Matrices, *SIAM J. on Matrix Analysis and Applications*, **9**, **4**, 543–560, 1988.

[EGH13] Y. Eidelman, I. Gohberg, I. Haimovici, *Separable Type Representations of Matrices and Fast Algorithms. Volume 1. Basics. Completion Problems. Multiplication and Inversion Algorithms. Volume 2. Eigenvalue method*, Birkhauser, 2013.

[ES05] A. Edelman, B. D. Sutton, Tails of Condition Number Distributions, *SIAM J. on Matrix Analysis and Applications*, **27**, **2**, 547–560, 2005.

[FKV98/04] A. Frieze, R. Kannan, S. Vempala, Fast Monte-Carlo Algorithms for Finding Low-rank Approximations, *J. of ACM*, **51**, 1025–1041, 2004. Proc. version in *39th FOCS*, pages 370–378, IEEE Computer Society Press, 1998.

[G11] J.F. Grcar, Mathematicians of Gaussian Elimination, *Notices of the Amer. Math. Society*, **58, 6**, 782–792, June/July 2011.

[GL13] G. H. Golub, C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 2013 (fourth addition).

[GS72] I. Gohberg, A. Sementsul, On the Inversion of Finite Toeplitz Matrices and Their Continuous Analogs, *Matematicheskiie Issledovaniia* (in Russian), **7**, **2**, 187–224, 1972.

[GTZ97] S. A. Goreinov, E. E. Tyrtyshnikov, N. L. Zamarashkin, A Theory of Pseudo-skeleton Approximations, *Linear Algebra and Its Applications*, **261**, 1–21, 1997.

[GZT97] S. A. Goreinov, N. L. Zamarashkin, E. E. Tyrtyshnikov, Pseudo-skeleton Approximations by Matrices of Maximal Volume, *Mathematical Notes*, **62, 4**, 515–519, 1997.

[H02] N. J. Higham, *Accuracy and Stability in Numerical Analysis*, SIAM, Philadelphia, 2002 (second edition).

[HMT11] N. Halko, P. G. Martinsson, J. A. Tropp, Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions, *SIAM Review*, **53, 2**, 217–288, 2011.

[KP91] E. Kaltofen, V. Y. Pan, Processor Efficient Parallel Solution of Linear Systems over an Abstract Field, *Proc. 3rd Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'91)*, 180–191, ACM Press, New York, 1991.

[KS91] E. Kaltofen, B. D. Saunders, On Wiedemann's Method for Solving Sparse Linear Systems, *Proceedings of AAECC–5, Lecture Notes in Computer Science*, **536**, 29–38, Springer, Berlin, 1991.

[M74] M. Morf, Fast Algorithms for Multivariable Systems, Ph.D. Thesis, *Department of Electrical Engineering, Stanford University*, Stanford, CA, 1974.

[M80] M. Morf, Doubling Algorithms for Toeplitz and Related Equations, *Proc. IEEE International Conference on ASSP*, 954–959, IEEE Press, Piscataway, New Jersey, 1980.

[M11] M. W. Mahoney, Randomized Algorithms for Matrices and Data, *Foundations and Trends in Machine Learning*, NOW Publishers, **3, 2**, 2011. (Abridged version in: *Advances in Machine Learning and Data Mining for Astronomy*, edited by M. J. Way, et al., pp. 647-672, 2012.)

[P01] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser/Springer, Boston/New York, 2001.

[P15] V. Y. Pan, Transformations of Matrix Structures Work Again, *Linear Algebra and Its Applications*, 465, 1–32, 2015.

[Pa] V. Y. Pan, How Bad Are Vandermonde Matrices? Available at arxiv: 1504.02118, submitted on 8 April 2015, revised on 10 July 2015.

[PGMQ] V. Y. Pan, D. Grady, B. Murphy, G. Qian, R. E. Rosholt, A. Ruslanov, Schur Aggregation for Linear Systems and Determinants, *Theoretical Computer Science*, *Special Issue on Symbolic–Numerical Algorithms* (D. A. Bini, V. Y. Pan, and J. Verschelde editors), **409**, **2**, 255–268, 2008.

[PIMR10] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning for Matrix Computations, *Linear Algebra and Its Applications*, **432**, 1070–1089, 2010.

[PP95] D. S. Parker, B. Pierce, The Randomizing FFT: An Alternative to Pivoting in Gaussian Elimination, Tech. Report CSD 950037, *Computer Science Dept., Univ. California at Los Angeles*, 1995.

[PQ10] V. Y. Pan, G. Qian, Solving Homogeneous Linear Systems with Randomized Preprocessing, *Linear Algebra Appls.*, **432**, 3272–3318, 2010.

[PQ12] V. Y. Pan, G. Qian, Solving Linear Systems of Equations with Randomization, Augmentation and Aggregation, *Linear Algebra and Its Applications*, **437**, 2851–1876, 2012.

[PQY14] V. Y. Pan, G. Qian, X. Yan, Random Multipliers Numerically Stabilize Gaussian and Block Gaussian Elimination: Proofs and an Extension to Low-rank Approximation, arxiv 1406.5802 math.NA (28 pages), June 23, 2014.

[PQY15] V. Y. Pan, G. Qian, X. Yan, Random Multipliers Numerically Stabilize Gaussian and Block Gaussian Elimination: Proofs and an Extension to Low-rank Approximation, *Linear Algebra and Its Applications*, **481**, 202–234, 2015.

[PQZ13] V. Y. Pan, G. Qian, A. Zheng, Randomized Preprocessing versus Pivoting, *Linear Algebra and Its Applications*, **438, 4**, 1883–1899, 2013.

[PQZC] V. Y. Pan, G. Qian, A. Zheng, Z. Chen, Matrix Computations and Polynomial Root-finding with Preprocessing, *Linear Algebra and Its Applications*, **434**, 854–879, 2011.

[PQZa] V. Y. Pan, G. Qian, L. Zhao, New Studies of Randomized Augmentation and Additive Preprocessing, *Linear Algebra and Its Applications*, in print, and arxiv 1412.5864 math.NA (38 pages), submitted on December 18, 2014, revised on July 10, 2015.

[PSZ15] V. Y. Pan, J. Svadlenka, L. Zhao, Estimating the Norms of Circulant and Toeplitz Random Matrices and Their Inverses, *Linear Algebra and Its Applications*, **468**, 197–210, 2015.

[PW08] V. Y. Pan, X. Wang, Degeneration of Integer Matrices Modulo an Integer, *Linear Algebra and Its Applications*, **429**, 2113–2130, 2008.

[PZ15] V. Y. Pan, L. Zhao, Randomized Circulant and Gaussian Preprocessing, in *Proceedings of the 17th International Workshop on Computer Algebra in Scientific Computing (CASC'2015)*, (V. P. Gerdt, V. Koepf, and E. V. Vorozhtsov, editors), *Lecture Notes in Computer Science*, **9301**, 359–373, Springer International Publishing Switzerland, 2015.

[PZa] V. Y. Pan, L. Zhao, Can We Combine the Powers of CG, FMM, and Low-rank Approximation Algorithms?, Available in arxiv, October 2015.

[S80] J. T. Schwartz, Fast Probabilistic Algorithms for Verification of Polynomial Identities, *Journal of ACM*, **27**, **4**, 701–717, 1980.

[SST06] A. Sankar, D. Spielman, S.-H. Teng, Smoothed Analysis of the Condition Numbers and Growth Factors of Matrices, *SIAM J. on Matrix Analysis and Applics.*, **28**, **2**, 446–476, 2006.

[T00] E. E. Tyrtyshnikov, Incomplete Cross Approximation in the Mosaic-skeleton Method, *Computing*, **64, 4**, 367–380, 2000.

[TS90] L. Trefethen, R. Schreiber, Average Case Analysis of Gaussian Elimination, *SIAM Journal on Matrix Analysis and Applications*, **11, 3**, 335–360, 1990.

[VVM07] R. Vandebril, M. Van Barel, N. Mastronardi, *Matrix Computations and Semiseparable Matrices: Linear Systems* (Volume 1), The Johns Hopkins University Press, Baltimore, Maryland, 2007.

[W61] J.H. Wilkinson, Error Analysis of Direct Methods of Matrix Inversion, *Journal of the ACM*; **8, 3**, 281–330, 1961.

[XXG12] J. Xia, Y. Xi, M. Gu, A Superfast Structured Solver for Toeplitz Linear Systems via Randomized Sampling, *SIAM Journal on Matrix Analysis and Applications*, **33**, 837–858, 2012.

[YC97] M.C. Yeung, T.F. Chan, Probabilistic Analysis of Gaussian Elimination without Pivoting, *SIAM Journal on Matrix Analysis and Applications*, **18, 2**, 499–517, 1997.

[Z79] R. E. Zippel, Probabilistic Algorithms for Sparse Polynomials, *Proceedings of EUROSAM'79, Lecture Notes in Computer Science*, **72**, 216–226, Springer, Berlin, 1979.

Appendices

Appendix A

Random matrices

A.1 Definitions and expected strong nonsingularity

We use the acronym " *i.i.d.*" for "independent identically distributed", keep referring to standard Gaussian random variables just as *Gaussian*, and call random variables *uniform* over a fixed finite set if their values are sampled from this set under the uniform probability distribution on it.

Defining a random matrix H, we assume that its entries are linear combinations of finitely many *i.i.d.* random variables, under the Gaussian or uniform probability distribution. The matrix is *Gaussian* if all its entries are i.i.d. Gaussian variables.

Theorem A.1.1. *Assume a nonsingular $n \times n$ matrix A and an $n \times n$ matrix H whose entries are linear combinations of finitely many i.i.d. random*

variables.

Let $\det((AH)_{l,l})$ *vanish identically in them for neither of the integers* l, $l = 1, \dots, n$.

(i) If the variables are uniform over a set $\mathcal{S}$ *of cardinality* $|\mathcal{S}|$, *then the matrix* $(AH)_{l,l}$ *is singular with a probability at most* $l/|\mathcal{S}|$, *for any* l, *and the matrix* AH *is strongly nonsingular with a probability at least* $1 - 0.5(n-1)n/|\mathcal{S}|$.

(ii) If these i.i.d. variables are Gaussian, then the matrix AH *is strongly nonsingular with probability 1.*

Proof. Part (i) of the theorem follows from a celebrated lemma of [DL78], also known from [Z79] and [S80]. Derivation is specified, e.g., in [PW08]. Part (ii) follows because the equation $\det((AH)_{l,l})$ for any integer l in the range from 1 to n defines an algebraic variety of a lower dimension in the linear space of the input variables (cf. [BV88, Proposition 1]). □

A.2 Rotational invariance and the condition number of a Gaussian matrix

Lemma A.2.1. *(*Rotational invariance of a Gaussian matrix.*) Suppose that* k, m, *and* n *are three positive integers,* G *is an* $m \times n$ *Gaussian matrix, and* S *and* T *are* $k \times m$ *and* $n \times k$ *orthogonal matrices, respectively. Then* SG

and GT are Gaussian matrices.

Next we recall some estimates for the norm and the condition number of a Gaussian matrix. For simplicity we assume that we deal with real matrices, but similar estimates in the case of complex matrices can be found in [D88], [E88], [CD05], and [ES05].

Hereafter we write $\nu_{m,n} = ||G||$, $\nu^+_{m,n} = ||G^+||$, and $\nu^+_{m,n,F} = ||G^+||_F$, for a Gaussian $m \times n$ matrix G, and write $\mathbb{E}(v)$ for the expected value of a random variable v.

Theorem A.2.1. *(Cf. [DS01, Theorem II.7].) Suppose that m and n are positive integers, $h = \max\{m, n\}$, $t \geq 0$, and $z \geq 2\sqrt{h}$. Then*

Probability$\{\nu_{m,n} > z\} \leq \exp(-(z - 2\sqrt{h})^2/2)$ *and*

Probability$\{\nu_{m,n} > t + \sqrt{m} + \sqrt{n}\} \leq \exp(-t^2/2)$.

Theorem A.2.2. *Let $\Gamma(x) = \int_0^\infty \exp(-t)t^{x-1}dt$ denote the Gamma function and let $x > 0$. Then*

(i) Probability $\{\nu^+_{m,n} \geq m/x^2\} < \frac{x^{m-n+1}}{\Gamma(m-n+2)}$ *for $m \geq n \geq 2$,*

(ii) Probability $\{\nu^+_{n,n} \geq x\} \leq 2.35\sqrt{n}/x$ *for $n \geq 2$,*

(iii) $\mathbb{E}(\nu^2_{F,m,n}) = m/|m - n - 1|$, *provided that $|m - n| > 1$, and*

(iv) $\mathbb{E}(\nu^+_{m,n}) \leq e\sqrt{m}/|m - n|$, *provided that $m \neq n$.*

Proof. See [CD05, Proof of Lemma 4.1] for part (i), [SST06, Theorem 3.3] for part (ii), and [HMT11, Proposition 10.2] for parts (iii) and (iv). □

Theorem A.2.2 provides probabilistic upper bounds on $\nu^{+}_{m,n}$. They are reasonable already for square matrices, for which $m = n$, but become much stronger as the difference $|m - n|$ grows large.

Theorems A.2.1 and A.2.2 combined imply that an $m \times n$ Gaussian matrix is very well-conditioned if the integer $m - n$ is large or even moderately large, and still can be considered well-conditioned if the integer $|m - n|$ is small or even vanishes (possibly with some grain of salt in the later case). These properties are immediately extended to all submatrices because they are also Gaussian.

Appendix B

Matrices of discrete Fourier transform

Definition B.0.1. *Write* $\omega = \exp(\frac{2\pi\sqrt{-1}}{n})$, $\Omega = \Omega_n = (\omega^{ij})_{i,j=0}^{n-1}$, $\Omega^{-1} = \frac{1}{n}\Omega^H$, *$\omega$ denotes a primitive n-th root of unity, Ω and Ω^{-1} denote the matrices of the discrete Fourier transform at n points and its inverse, to which we refer as DFT(n) and IDFT(n), respectively,*

$$\Omega = \Omega_n = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \omega^{n-1} & \omega^{2n-2} & \omega^{3n-3} & \dots & \omega^{(n-1)^2} \end{pmatrix}.$$

Remark B.0.1. *If $n = 2^k$ is a power of 2, we can apply the FFT algorithm and perform DFT(n) and IDFT(n) by using only $1.5n\log_2(n)$ and $1.5n\log_2(n) + n$ arithmetic operations, respectively. For an $n \times n$ input and any n, we can perform DFT(n) and IDFT(n) by using $cn\log(n)$ arithmetic*

operations, but for a larger constant c (see [P01, Section 2.3]).

Appendix C

f-circulant and subcirculant matrices

For a positive integer n and a complex scalar f, define the $n \times n$ unit f-circulant matrix $Z_f = \begin{pmatrix} 0 & f \\ I_{n-1} & 0 \end{pmatrix}$ and the $n \times n$ general *f-circulant matrix* $Z_f(\mathbf{v}) = \sum_{i=0}^{n} v_i Z_f^i$,

$$Z_f = \begin{pmatrix} 0 & \dots & \dots & 0 & f \\ 1 & \ddots & & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & & \ddots & 0 & 0 \\ 0 & \dots & \dots & 1 & 0 \end{pmatrix}$$

and

$$Z_f(\mathbf{v}) = \begin{pmatrix} v_0 & fv_{n-1} & \cdots & fv_1 \\ v_1 & v_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & fv_{n-1} \\ v_{n-1} & \cdots & v_1 & v_0 \end{pmatrix}.$$

$Z_f(\mathbf{v})$ is a lower triangular Toeplitz matrix for $f = 0$, *circulant* for $f = 1$, and skew-circulant for $f = -1$. $Z^n = fI$ for I denoting the identity matrix,

and the matrix $Z_f(\mathbf{v})$ is defined by its first column $\mathbf{v} = (v_i)_{i=0}^{n-1}$.

We call an f-circulant matrix a *Gaussian f-circulant* (or just *Gaussian circulant* if $f = 1$) if its first column is filled with independent Gaussian variables. For every fixed f, the f-circulant matrices form an algebra in the linear space of $n \times n$ Toeplitz matrices $T = (t_{i-j})_{i,j=0}^{n-1}$ (cf. equation (**??**)).

Hereafter, for a vector $\mathbf{u} = (u_i)_{i,j=0}^{n-1}$, write $D(\mathbf{u}) = \mathrm{diag}(u_0, \dots, u_{n-1})$, that is, $D(\mathbf{u})$ is the diagonal matrix with the diagonal entries $u_0, \dots, u_{n-1}$.

Theorem C.0.1. *(Cf. [CPW74].) If $f \neq 0$, then f^n-circulant matrix $Z_{f^n}(\mathbf{v})$ of size $n \times n$ can be factored as follows, $Z_{f^n}(\mathbf{v}) = U_f^{-1} D(U_f \mathbf{v}) U_f$ for $U_f = \Omega D(\mathbf{f})$, $\mathbf{f} = (f^i)_{i=0}^{n-1}$, and $f \neq 0$. In particular, for circulant matrices, $D(\mathbf{f}) = I$, $U_f = \Omega$, and $Z_1(\mathbf{v}) = \Omega^{-1} D(\Omega \mathbf{v}) \Omega$.*

Corollary C.0.1. *For $f \neq 0$ one can multiply an $n \times n$ f-circulant matrix by a vector by applying two DFT(n), an IDFT(n), and additionally $n + 2\delta_f n$ multiplications and divisions where $\delta_f = 0$ if $f = 1$ and $\delta_f = 1$ otherwise.*

Remark C.0.1. *We cannot apply this theorem directly to a triangular Toeplitz (0-circulant) matrix, but we can represent such a matrix (as well as any square Toeplitz matrix) as the sum of a circulant matrix and a skew-circulant one and then multiply this sum by a vector at roughly the double computational cost, compared to the case of $f \neq 0$.*

Corollary C.0.2. *Let $D(\mathbf{u}) = \mathrm{diag}(u_0, \dots, u_{n-1})$ (cf. part (ii) of Definition B.0.1).*

(i) If we are given a diagonal matrix $D(\mathbf{u})$ for $\mathbf{u} = \Omega\mathbf{v}$, then we can recover the vector $\mathbf{v} = \frac{1}{n}\Omega^H\mathbf{u}$, which defines the entries of the circulant matrix $Z_1(\mathbf{v})$.

(ii) If the vector $\mathbf{v}$ is Gaussian, then so is also the vector $\mathbf{u} = (u_i)_{i=1}^n = \frac{1}{\sqrt{n}}\Omega\mathbf{v}$ (by virtue of Lemma A.2.1) and vice versa. Each of the two vectors defines a Gaussian circulant matrix $Z_1(\mathbf{v})$.

(iii) By choosing $u_i = \exp(\frac{\phi_i}{2\pi}\sqrt{-1})$ and real Gaussian variable ϕ_i for all i, we arrive at a random unitary $n \times n$ circulant matrix $Z_1(\mathbf{v})$ defined by n real Gaussian parameters ϕ_i, $i = 0, \dots, n-1$. Alternatively we can set $\phi_i = \pm 1$ for all i and choose the signs $\pm$ at random, with i.i.d. probability 1/2 for each sign.

(iv) By adding another Gaussian parameter ϕ, we can define a random unitary f-circulant matrix $Z_f(\mathbf{v})$ for $f = \exp(\frac{\phi}{2\pi}\sqrt{-1})$.

The following results of [PSZ15] imply that a Gaussian circulant matrix is well-conditioned with a probability close to 1.

Theorem C.0.2. *Suppose that $Z_1(\mathbf{v}) = \Omega^H D\Omega$ is a nonsingular circulant $n \times n$ matrix, and let $D(\mathbf{g}) = \mathrm{diag}(g_i)_{i=1}^n$, for $\mathbf{g} = (g_i)_{i=1}^n$. Then $||Z_1(\mathbf{v})|| =$*

$\max_{i=1}^{n} |g_i|$, $||Z_1(\mathbf{v})^{-1}|| = \min_{j=1}^{n} |g_j|$, *and* $\kappa(Z_1(\mathbf{v})) = \max_{i,j=1}^{n} |g_i/g_j|$, *for* $\mathbf{v} = \Omega^{-1}\mathbf{g}$.

Remark C.0.2. *Suppose that a circulant matrix* $Z_1(\mathbf{v})$ *has been defined by its first column vector* $\mathbf{v}$ *filled with the integers* ± 1 *for a random choice of the i.i.d. signs* $\pm$*, each* $+$ *and* $-$ *chosen with probability* $1/2$*. Then, clearly, the entries* g_i *of the vector* $\mathbf{g} = \Omega\mathbf{v} = (g_i)_{i=1}^{n}$ *satisfy* $|g_i| \le n$ *for all* i *an* n*, and furthermore, with a probability close to 1,* $\max_{i=1}^{n} \log(1/|g_i|) = O(\log(n))$ *as* $n \to \infty$.

Remark C.0.3. *In the case of a Gaussian circulant matrix* $Z_1(\mathbf{v})$*, all the entries* g_i *are i.i.d. Gaussian variables, and the condition number* $\kappa(Z_1(\mathbf{v})) = \max_{i,j=1}^{n} |g_i/g_j|$ *is not likely to be large. The blocks of the matrix* $Z_1(\mathbf{v})$ *are not circulant matrices, and so the latter property is not extended to them, unlike the case of a Gaussian matrix. We can, however, extend the above bound to the condition numbers* $\kappa(B)$ *of* $n \times k$ *and* $k \times n$ *blocks* B *of a non-singular* $n \times n$ *circulant matrix* $Z_1(\mathbf{v})$*, for any positive integer* $k < n$*, because* $\kappa(B) \le \kappa(Z_1(\mathbf{v}))$*, for such blocks (cf. [GL13, Corollary 8.6.3]). We call them* subcirculant *matrices. They are well-conditioned with probability close to 1, and we can even force them to be unitary, while still been defined by by* n *i.i.d. Gaussian variables, like the matrices in part (iii) of Corollary C.0.2.*

Random subcirculant multipliers have advantage of preserving Toeplitz-like matrix structure (cf. [P01, Chapter 5]) and thus can serve as natural pre-processors for Toeplitz and Toeplitz-like matrices. By using such a multiplier, one can improve a little the recipe in [XXG12, Section 3.2], where a random general rectangular Toeplitz multiplier is applied to a Toeplitz-like input.