Defect Detection from 3D Ultrasonic Measurements Using Matrix-free Sparse Recovery Algorithms

Sebastian Semper*, Jan Kirchhof*†, Christoph Wagner*, Fabian Krieg*†,
Florian Römer†, Ahmad Osman† and Giovanni Del Galdo*‡
*Technische Universität Ilmenau, Institute for Information Technology, Germany
†Fraunhofer Institute for Non-Destructive Testing IZFP, Saarbrücken, Germany
‡Fraunhofer Institute for Integrated Circuits IIS, Ilmenau, Germany

Abstract—**In this paper, we propose an efficient matrix-free algorithm to reconstruct locations and size of flaws in a specimen from volumetric ultrasound data by means of a native 3D Sparse Signal Recovery scheme using Orthogonal Matching Pursuit (OMP). The efficiency of the proposed approach is achieved in two ways. First, we formulate the dictionary matrix as a block multilevel Toeplitz matrix to minimize redundancy and thus memory consumption. Second, we exploit this specific structure in the dictionary to speed up the correlation step in OMP, which is implemented matrix-free. We compare our method to state-of-the-art, namely 3D Synthetic Aperture Focusing Technique, and show that it delivers a visually comparable performance, while it gains the additional freedom to use further methods such as Compressed Sensing.**

I. Introduction & State of the Art

In ultrasonic Non-Destructive Testing (NDT), the goal is to localize and characterize flaws inside of a specimen from ultrasonic measurements. To achieve this, various post-processing methods have been proposed, e.g. the Total Focusing Method [1], [2] or the Synthetic Aperture Focusing Technique [3]. It has been shown that Sparse Signal Recovery (SSR) based approaches [4], [5], [6], [7] lead to improved reconstruction results compared to the classical focusing techniques. So far, SSR has only been applied to 2D ultrasonic B-scan data due to its computational demands. Interestingly, the authors in [8] illustrate how exploiting multilevel Toeplitz structures massively lowers the computational demands for the solution of 3D acoustic scattering problems, where a large system of linear equations has to be solved, which is done by a matrix-free algorithm. The notion matrix-free means to solve the linear system of equations, no full system matrix is built up in memory, instead only the matrix-vector multiplication is implemented in a specific algorithm, which then is used by the solver for the linear system. We apply ideas, which are to some degree inspired by the approach in [8], to compress the dictionary occurring in an SSR problem to make the computations feasible on a modern HPC node.

This new matrix-free approach allows to solve SSR problems as discussed in Section II at much higher resolutions than before. Therefore, one can reconstruct flaws in the complete 3D volumetric data instead of sliced reconstructions on 2D B-scans. This yields the advantage that one can make use of forward models which take the continuity of the data across multiple B-scans into account. Further, a sophisticated SSR approach is the foundation to detect defects from low-rate measurements using Compressed Sensing (CS) [9].

In this paper, the following notation is used. For $n \in N$, a given ordered set $S \subset \{1, \dots, n\}$ and a vector $\boldsymbol{x} \in \mathbb{C}^n$ we set $\boldsymbol{x}_S \in \mathbb{C}^{|S|}$ to be the restriction of $\boldsymbol{x}$ to the elements indexed by S. For given $n \in \mathbb{N}$ we denote with $\boldsymbol{I}_n$ the identity matrix on $\mathbb{C}^n$ and 1-dimensional Fourier matrix acting on vectors in $\mathbb{C}^n$ as $\boldsymbol{\mathcal{F}}_n$. Also, $\boldsymbol{A} \otimes \boldsymbol{B}$ denotes the Kronecker product. For a matrix $\boldsymbol{M} \in \mathbb{C}^{n \times n}$, the matrix $\boldsymbol{M}_S$ consists of the columns and rows indexed by the set S.

II. Ultrasonic Non-destructive testing

In ultrasonic NDT, the widespread pulse-echo method inserts an ultrasound pulse into the specimen and measuring the echo signal. Following [5], the discrete measurement samples of this process can be modeled as delayed echoes of the inserted pulse given by

$$\begin{aligned} b_{x,y}(mt_s) = \sum_{i=1}^{I} & a_i \cdot g(\tau_{x,y}(x_i, y_i, z_i)) \\ & \cdot h(mt_s - \tau_{x,y}(x_i, y_i, z_i)) + n(mt_s), \end{aligned} \tag{1}$$

where t_s is the sampling period, $h : \mathbb{R} \to \mathbb{R}$ is the inserted pulse, a_i the reflectivity coefficient of the i-th reflector, $g : \mathbb{R} \to \mathbb{R}$ a window function to account for attenuation and $n(\cdot)$ the measurement noise. To take several pulse-echo measurements from different positions of the specimen surface, we define a 2-dimensional equispaced grid on the surface defined by

$$\begin{aligned} G_{2D} = \{(x, y) | x = n_x \cdot \Delta x, n_x \in \{0, \dots, N_x - 1\}, \\ y = n_y \cdot \Delta y, n_y \in \{0, \dots, N_y - 1\}\}, \end{aligned}$$

where $\Delta x = \Delta y$ is the grid spacing and $(x, y) \in G_{2D}$ in Eq. (1). The observed echoes of a particular reflector produce a predefined trace $\tau_k(x_i, y_i, z_i)$ in these measurements that depends on the distance change between transducer and reflector based on the specimen geometry. By assuming a flat surface, we get for the trace

$$\tau_{x,y}(x_i, y_i, z_i) = \frac{1}{c}\sqrt{(x - x_i)^2 + (y - y_i)^2 + z_i^2},$$

The authors would like to thank Henning Schwanbeck for the support with the HPC. Christoph Wagner is supported by the Carl-Zeiss Foundation under the project "PRIME". Sebastian Semper is funded by DFG under the project "CoSMoS". Florian Römer is is funded by the Fraunhofer Internal Programs under Grant No. Attract 025-601128.

where c is the speed of sound within the medium. The $M \cdot N_x \cdot N_y$ samples are then collected in

$\boldsymbol{b}_{m,x,y} = [b_{x,y}(m \cdot t_s)]_{m \in \{0,\dots,M-1\},(x,y)\in G_{2D}} \in \mathbb{R}^{M \times N_x \times N_y}$.

The reflections arise from possible defects inside the specimen, which we want to locate. To recover the positions of the I reflectors, we need to fit the parameter triplets (z_i, x_i, y_i) to the observed data. To this, we discretize z and define the 3D grid $G_{3D} = \{(z,x,y)|(x,y) \in G_{2D}, z = n_z \cdot \Delta z, n_z = 0,\dots,M-1\}$ with $\Delta z = t_s \cdot c$, which aligns the reconstruction grid with the sampling grid of the measurement, and a total number of $N = N_x N_y M$ voxels. This gridding process introduces a model error, which is negligible as long as we chose G_{3D} fine enough. Then $\operatorname{vec} \boldsymbol{b}$ can be written as a linear combination of atoms $\boldsymbol{H}_{z_n,x_n,y_n} \in \mathbb{R}^N$, $(z_n, x_n, y_n) \in G_{3D}$, where the elements of $\boldsymbol{H}_{z_n,x_n,y_n}$ are given by $g(\tau_{x,y}(x_n, y_n, z_n)) \cdot h(mt_s - \tau_{x,y}(x_n, y_n, z_n))$. In other words, the vector $\boldsymbol{H}_{z_n,x_n,y_n}$ contains the volumetric data set of a single reflector at position (z_n, x_n, y_n). This is expressed concisely via

$$\operatorname{vec} \boldsymbol{b} = \boldsymbol{H}\boldsymbol{a} + \boldsymbol{n}. \tag{2}$$

Here, $\boldsymbol{H} \in \mathbb{R}^{N \times N}$ is the so-called dictionary matrix containing all vectors $\boldsymbol{H}_{z_n,x_n,y_n}$ as its columns, $\boldsymbol{a} \in \mathbb{R}^N$ contains the a_i from Eq. (1), where the corresponding index i for a reflector at position (z, x, y) is determined by $i = z \cdot N_x N_y + y \cdot N_x + x$. Finally $\boldsymbol{n} \in \mathbb{R}^N$ is a vector containing the noise as well as the model error, e.g., from the gridding. Since the number of defects can be assumed small compared to the total number of possible reflector positions, finding $\boldsymbol{a}$ in Eq. (2) can be achieved by solving a SSR problem [5], which in our case reads as

$$\min \|\boldsymbol{a}\|_0 \quad \text{s.t.} \quad \operatorname{vec} \boldsymbol{b} = \boldsymbol{H} \cdot \boldsymbol{a} \tag{3}$$

For realistic problem sizes in ultrasonic NDT, computing the full matrix $\boldsymbol{H}$ becomes infeasible with current computer technology. For instance, for the measurement we consider in this paper, $\boldsymbol{b}$ is of size $181 \cdot 381 \cdot 182 = 12550902$. The resulting $\boldsymbol{H}$ contains approximately $1.57 \cdot 10^{14}$ elements that would require 1.3 PB of storage. One approach is to store $\boldsymbol{H}$ in a sparse data structure. In the worst case, it contains approximately $N_x^2 N_y^2 M N_p$ nonzeros, with N_p being the number of points needed to sample $h(t)$. So although $N_p \ll M$, a sparse representation of $\boldsymbol{H}$ does not allow to completely store $\boldsymbol{H}$ in memory.

To overcome these current computational limits, the structure in $\boldsymbol{H}$ must be exploited, effectively removing redundancy. SSR algorithms such as Fast Iterative Shrinkage Thresholding (FISTA) [10] or Orthogonal Matching Pursuit (OMP) [11] utilize only the forward and backward projections, $\boldsymbol{H}^{\mathrm{H}}\boldsymbol{x}$ and $\boldsymbol{H}\boldsymbol{x}$ for some given $\boldsymbol{x} \in \mathbb{R}^N$, since following [12, A.3] the solution to the least squares problem (needed in OMP) can be updated iteratively from the solution in the previous step. Therefore, we will follow this approach throughout the remainder of this paper. Section III introduces efficient projection algorithms for matrices having a block multilevel Toeplitz structure [8], [13]. In Lemma IV.1 we derive that $\boldsymbol{H}$ indeed is block multilevel Toeplitz and present reconstruction results from volumetric ultrasound measurement data, further analyzing computation time and memory consumption.

III. Algorithms

This section is dedicated to efficient forward and backward projections of multilevel circulant and Toeplitz matrices, where the first two sections III-A and III-B revise the procedure to derive efficient implementations for "standard" circulant and Toeplitz matrices.

A. Circulant Matrices

Define the mapping $\boldsymbol{\Gamma} : \mathbb{C}^n \to \mathbb{C}^{n \times n}$ for given $n \in \mathbb{N}$ and $\boldsymbol{c} \in \mathbb{C}^n$

$$\boldsymbol{c} \mapsto \boldsymbol{\Gamma}(\boldsymbol{c}) = \begin{bmatrix} c_1 & c_n & \dots c_2 \\ c_2 & c_1 & \ddots c_3 \\ \vdots & & \ddots c_n \\ c_n & c_{n-1} & \dots c_1 \end{bmatrix}.$$

It is a well known fact, that for any $\boldsymbol{c} \in \mathbb{C}^n$ it holds that

$$\boldsymbol{\Gamma}(\boldsymbol{c}) = \frac{1}{n} \boldsymbol{\mathcal{F}}_n^{\mathrm{H}} \operatorname{diag}(\boldsymbol{\mathcal{F}}_n \boldsymbol{c}) \boldsymbol{\mathcal{F}}_n, \tag{4}$$

which allows to implement an efficient algorithm to calculate the action, or forward transform, of $\boldsymbol{\Gamma}(\boldsymbol{c})$ or the action of $\boldsymbol{\Gamma}(\boldsymbol{c})^{\mathrm{H}}$, the backward transform, to a vector $\boldsymbol{x} \in \mathbb{C}^n$, i.e.,

$$\boldsymbol{y}_1 = \boldsymbol{\Gamma}(\boldsymbol{c})\boldsymbol{x} \quad \text{and} \quad \boldsymbol{y}_2 = \boldsymbol{\Gamma}(\boldsymbol{c})^{\mathrm{H}}\boldsymbol{x}$$

by exploiting the Fast Fourier Transform (FFT) [14]. Moreover, it allows storing $\boldsymbol{\Gamma}(\boldsymbol{c})$ memory-efficiently as $\boldsymbol{\mathcal{F}}_n \boldsymbol{c}$, reducing the memory complexity to $\mathcal{O}(n)$, compared to $\mathcal{O}(n^2)$ as in the case of unstructured matrices.

However, the key idea behind the FFT algorithm is also its greatest caveat, because it only achieves reasonable performance if n factors into many small prime factors. In the extreme case where n is prime on the other hand, the FFT is not faster than a standard matrix-vector multiplication. To alleviate this drawback we combine zero-padding with the FFT algorithm to increase the dimension n to a more feasible dimension in terms of its prime factors. To this end we define the mapping $\varphi : \mathbb{N} \to \mathbb{N}$, where $\varphi(n)$ is the number of multiply-and-accumulate (MAC) operations necessary to calculate $\boldsymbol{\mathcal{F}}_n \boldsymbol{x}$ for $\boldsymbol{x} \in \mathbb{C}^n$. Moreover, we define $\Phi : \mathbb{N} \to \mathbb{N}$ as $\Phi(n) = \min \operatorname{argmin}_{m \geqslant 2n-1} \varphi(m)$ and $\boldsymbol{\gamma}_k : \mathbb{C}^n \to \mathbb{C}^{2n-1+k}$ as

$$\boldsymbol{\gamma}_k(\boldsymbol{c}) = [c_1, \dots, c_n, \boldsymbol{0}_k^{\mathrm{T}}, c_2, \dots, c_n]^{\mathrm{T}}.$$

The key idea is now to embed $\boldsymbol{\Gamma}(\boldsymbol{c})$ into a larger circulant matrix $\boldsymbol{\mathcal{C}}_c(\boldsymbol{c}) \in \mathbb{C}^{N \times N}$, such that the possible bottleneck when calculating $\boldsymbol{\mathcal{F}}_n$ via an FFT is replaced by the shorter execution time of the FFT for $\boldsymbol{\mathcal{F}}_k$ for some $k \geqslant 2n - 1$.

Using the Cooley-Tukey algorithm for computing the FFT as found in [15], an algebraic matrix representation can be derived, which describes $\boldsymbol{\mathcal{F}}_n$ as composition of $\boldsymbol{\mathcal{F}}_\nu$, with $\nu \in \mathbb{N}$ being a prime factor of n, and $\boldsymbol{\mathcal{F}}_p$ with $p \in \mathbb{N} = n/\nu$ being the order of the square remainder matrix

$$\boldsymbol{\mathcal{F}}_n = \boldsymbol{P}_n^{(\nu)} \cdot (\boldsymbol{I}_\nu \otimes \boldsymbol{\mathcal{F}}_p) \cdot \boldsymbol{D}_n^{(\nu)} \cdot (\boldsymbol{\mathcal{F}}_\nu \otimes \boldsymbol{I}_p). \tag{5}$$

where $\boldsymbol{D}_n^{(\nu)}$ is a diagonal matrix of size n holding the FFT twiddle factors corresponding to ν and $\boldsymbol{P}_n^{(\nu)}$ being the stride permutation matrix which can be defined according to its effect on a vector via

$$\boldsymbol{P}_n^{(\nu)} \cdot [x_0, x_1, \dots, x_{n-1}]^{\mathrm{T}} = [y_0, y_1, \dots, y_{\nu-1}]^{\mathrm{T}},$$

where $y_i = [x_i, x_{i+p}, x_{i+2p}, \dots, x_{i+(\nu-1)p}]$. Analyzing (5) we find a product of four square matrices of size $n \times n$ each. $\boldsymbol{P}_n^{(\nu)}$

can be implemented through clever memory access, requiring no additional ops, whereas $\boldsymbol{D}_n^{(\nu)}$ requires n MAC operations. The anterior Kronecker product resembles a block diagonal matrix with ν copies of the remainder matrix requiring $\nu \cdot \varphi(p)$ MAC ops and, given $\nu = n$ no operations at all, as it then equals to the identity matrix. The posterior Kronecker product is a structured sparse matrix, requiring $p \cdot \nu^2 = \nu \cdot n$ MAC ops. In total, (5) requires $\nu \cdot \varphi(p) + (\nu + 1)n$ MAC ops. Defining $\boldsymbol{\psi}_n \in \mathbb{N}^w$ as the prime factor decomposition of n with w being the prime factor count, we recursively apply (5) over all $\boldsymbol{\psi}_{n,i}$ resulting in a total complexity of

$$\varphi(n) = n \cdot (1 + w + \sum_{i=1}^{w} \boldsymbol{\psi}_{n,i}). \tag{6}$$

The special case $\nu = 4$ needs to be considered carefully during the implementation of the aforementioned optimization problem. As all elements in $\boldsymbol{\mathcal{F}}_4$ are drawn from the set $(1, -1, i, -i)$ only simple additions are required in order to compute its action on a vector. Many current computing platforms are capable of exploiting this, resulting in an additional gain over the estimate $\varphi(4)$. The implementation used in this work employs a variant of Dijkstra's algorithm [16] for the search and rewards prime factors of four on grounds of the preceding consideration. To incorporate above finding about ϕ in our procedure, we define

$$\boldsymbol{\mathcal{C}}_n(\boldsymbol{c}) = \begin{cases} \boldsymbol{\Gamma}(\boldsymbol{c}), & \text{for} \quad \varphi(\Phi(n)) \geqslant \varphi(n) \\ \boldsymbol{\Gamma}(\boldsymbol{\gamma}_{\Phi(n)-2n-1}(\boldsymbol{c})) & \text{otherwise.} \end{cases}$$

The case $\varphi(\Phi(n)) \geqslant \varphi(n)$ represents the fact that we cannot improve the execution time needed to apply $\boldsymbol{\mathcal{F}}_n$ by expanding the circulant matrix. To finalize the procedure note that for $k \in \mathbb{N}$ and $\boldsymbol{x} \in \mathbb{C}^n$ it holds that

$$\begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{z} \end{bmatrix} = \boldsymbol{\Gamma}(\boldsymbol{\gamma}_k(\boldsymbol{c})) \cdot \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{0}_{n-1+k} \end{bmatrix}, \quad \text{where} \quad \boldsymbol{y}_1 = \boldsymbol{\Gamma}(\boldsymbol{c}) \cdot \boldsymbol{x}$$

with $\boldsymbol{y}_1 \in \mathbb{C}^n$ and $\boldsymbol{z} \in \mathbb{C}^{n-1+k}$. As such, the application of $\boldsymbol{\Gamma}(\boldsymbol{\gamma}_k(\boldsymbol{c}))$ to a vector can be carried out efficiently according to (4).

B. Toeplitz Matrices

The concepts of Section III-A can be extended to the more general case, where the matrix is not circulant, but Toeplitz instead. For $n \in \mathbb{N}$ and $\boldsymbol{t} \in \mathbb{C}^{2n-1}$, we define a mapping $\boldsymbol{\Theta} : \mathbb{C}^{2n-1} \to \mathbb{C}^{n \times n}$ via

$$\boldsymbol{t} \mapsto \boldsymbol{\Theta}(\boldsymbol{t}) = \begin{bmatrix} t_1 & t_{2n-1} & \dots & t_{n+1} \\ t_2 & t_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{2n-1} \\ t_n & t_{n-1} & \dots & t_1 \end{bmatrix}.$$

Similar to $\boldsymbol{\gamma}_k$ before, we define a mapping $\boldsymbol{\vartheta}_k : \mathbb{C}^{2n-1} \to \mathbb{C}^{2n-1+k}$, as

$$\boldsymbol{\vartheta}_k(\boldsymbol{t}) = [t_1, t_2, \dots, t_n, \boldsymbol{0}_k, t_{n+1}, \dots, t_{2n-1}]^{\mathrm{T}}.$$

With this definition at hand it is clear that for $\boldsymbol{t} \in \mathbb{C}^{2n-1}$, $\boldsymbol{x} \in \mathbb{C}^n$ and $\boldsymbol{z} \in \mathbb{C}^k$ the relation

$$\begin{bmatrix} \boldsymbol{\Theta}(\boldsymbol{t}) \cdot \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix} = \boldsymbol{\Gamma}(\boldsymbol{\vartheta}_k(\boldsymbol{t})) \cdot \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{0}_k \end{bmatrix} \tag{7}$$

holds for any $k \geqslant 0$. This implies that Toeplitz matrices also have an efficient forward and backward transform by means of the algorithm provided for circulant matrices. Again, we need to find a suitable $k \geqslant 0$, such that the calculations in (7) are most efficient. To this end, we simply set

$$k = \min \operatorname*{argmin}_{\ell \geqslant 0} \varphi(2n - 1 + \ell).$$

C. Multilevel Circulant Matrices

As a next generalization, we define so called multilevel circulant matrices, which are not circulant by themselves, but consist of multiple nested levels of circulant structures. Let $d \geqslant 2$, $\boldsymbol{n} = [n_1, \dots, n_d]$, $\boldsymbol{n}_{1-} = [n_1, \dots, n_{d-1}]$ and $\boldsymbol{n}_{-1} = [n_2, \dots, n_d]$. Then, given a d-dimensional complex sequence $\boldsymbol{c} = [c_{\boldsymbol{k}}]$ for the multi index $\boldsymbol{k} \in \mathbb{N}^d$ a d-level circulant matrix $\boldsymbol{C}_{\boldsymbol{n},d}$ is recursively defined as

$$\boldsymbol{C}_{\boldsymbol{n},d} = \begin{bmatrix} \boldsymbol{C}_{[1,\boldsymbol{n}_{-1}],\ell} & \boldsymbol{C}_{[n_1,\boldsymbol{n}_{-1}],\ell} & \dots & \boldsymbol{C}_{[2,\boldsymbol{n}_{-1}],\ell} \\ \boldsymbol{C}_{[2,\boldsymbol{n}_{-1}],\ell} & \boldsymbol{C}_{[1,\boldsymbol{n}_{-1}],\ell} & \dots & \boldsymbol{C}_{[3,\boldsymbol{n}_{-1}],\ell} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{C}_{[n_1,\boldsymbol{n}_{-1}],\ell} & \boldsymbol{C}_{[n_1-1,\boldsymbol{n}_{-1}],\ell} & \dots & \boldsymbol{C}_{[1,\boldsymbol{n}_{-1}],\ell} \end{bmatrix},$$

where $\ell = d - 1$. So for $\boldsymbol{n} = [2, 2]$ and $\boldsymbol{c} \in \mathbb{C}^{2 \times 2}$ we get

$$\boldsymbol{C}_{[2,2],2} = \begin{bmatrix} \boldsymbol{C}_{[1,2],1} & \boldsymbol{C}_{[2,2],1} \\ \boldsymbol{C}_{[2,2],1} & \boldsymbol{C}_{[1,2],1} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{2,1} & c_{2,2} \\ c_{1,2} & c_{1,1} & c_{2,2} & c_{2,1} \\ c_{2,1} & c_{2,2} & c_{1,1} & c_{1,2} \\ c_{2,2} & c_{2,1} & c_{1,2} & c_{1,1} \end{bmatrix}.$$

It is worth noting that we stick with $\boldsymbol{c}$ containing all defining elements as the representation of a circulant matrix even in the d-level case, where $\boldsymbol{c}$ then becomes a tensor of order d as a natural extension to the circulant case for $d = 1$. To clarify how the elements in $\boldsymbol{c}$ are placed into $\boldsymbol{C}_{\boldsymbol{n},d}$, we note that

$$\boldsymbol{C}_{[n_1,\dots,n_{d-1},k],1} = \boldsymbol{\Gamma}\left(c_{[n_1,\dots,n_{d-1},k]}\right)$$

for all $k = 1, \dots, n_d$. In spirit of the sections before, we aim at providing efficient means of storing $\boldsymbol{C}_{\boldsymbol{n},d}$ and applying its forward and backward transform to a vector. To exploit the multilevel structure, we exploit the diagonalization of a multilevel circulant matrix [13], being

$$\boldsymbol{C}_{\boldsymbol{n},d} = \bigotimes_{i=1}^{d} \frac{1}{n_i} \boldsymbol{\mathcal{F}}_{n_i}^{\mathrm{H}} \operatorname{diag}\left(\bigotimes_{i=1}^{d} \boldsymbol{\mathcal{F}}_{n_i} \cdot \operatorname{vec} \boldsymbol{c} \right) \bigotimes_{i=1}^{d} \boldsymbol{\mathcal{F}}_{n_i}. \tag{8}$$

This expression directly yields an algorithm to efficiently multiply $\boldsymbol{C}_{\boldsymbol{n},d}$ with a vector, because Fourier matrices have a fast algorithm, and [17] describes how to efficiently compute the forward and backward transform of a Kronecker product, which also only makes use of the forward and backward transform of the factors involved.

D. Multilevel Toeplitz Matrices

Similar to the previous section, we can also efficiently handle the more general multilevel Toeplitz case. Let $d \geqslant 2$. Then, given a d-dimensional complex sequence $\boldsymbol{t} = [t_{\boldsymbol{k}}]$ for the multi index $\boldsymbol{k} \in \mathbb{N}^d$, a d-level Toeplitz matrix $\boldsymbol{T}_{\boldsymbol{n},d}$ is recursively defined as

$$\boldsymbol{T}_{(\boldsymbol{n},d)}(\boldsymbol{t}) = \begin{bmatrix} \boldsymbol{T}_{(1,\boldsymbol{m}),\ell} & \boldsymbol{T}_{(2n_1-1,\boldsymbol{m}),\ell} & \dots & \boldsymbol{T}_{(n_1+1,\boldsymbol{m}),\ell} \\ \boldsymbol{T}_{(2,\boldsymbol{m}),\ell} & \boldsymbol{T}_{(1,\boldsymbol{m}),\ell} & \dots & \boldsymbol{T}_{(n_1+2,\boldsymbol{m}),\ell} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{T}_{(n_1,\boldsymbol{m}),\ell} & \boldsymbol{T}_{(n_1-1,\boldsymbol{m}),\ell} & \dots & \boldsymbol{T}_{(1,\boldsymbol{m}),\ell} \end{bmatrix},$$

where $\boldsymbol{m} = \boldsymbol{n}_{-1}$ and $\ell = d - 1$. For example, $\boldsymbol{n} = [2, 2]$ and $\boldsymbol{t} \in \mathbb{C}^{3\times 3}$ yields

$$\boldsymbol{T}_{[2,2],2} = \begin{bmatrix} \boldsymbol{T}_{[1,2],1} \, \boldsymbol{T}_{[3,2],1} \\ \boldsymbol{T}_{[2,2],1} \, \boldsymbol{T}_{[1,2],1} \end{bmatrix} = \begin{bmatrix} t_{1,1} \, t_{1,3} \, t_{3,1} \, t_{3,3} \\ t_{1,2} \, t_{1,1} \, t_{3,2} \, t_{3,1} \\ t_{2,1} \, t_{2,3} \, t_{1,1} \, t_{1,3} \\ t_{2,2} \, t_{2,1} \, t_{1,2} \, t_{1,1} \end{bmatrix}.$$

We also retain the notation $\boldsymbol{t}$ for the defining elements of $\boldsymbol{T}_{(\boldsymbol{n},d)}$ and it naturally becomes a tensor of order d. As in Section III-B we describe how a d-level Toeplitz matrix can be embedded into a larger d-level circulant matrix such that one can use the efficient methods available for those to apply $\boldsymbol{T}_{(\boldsymbol{n},d)}$ to a vector.

For a given block Toeplitz matrix $\boldsymbol{T}_{n,m} \in \mathbb{C}^{nm\times nm}$, which reads as

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T}_1 \, \boldsymbol{T}_{2n-1} \dots \boldsymbol{T}_{n+1} \\ \boldsymbol{T}_2 \quad \boldsymbol{T}_1 \quad \ddots \quad \vdots \\ \vdots \quad \ddots \quad \ddots \quad \vdots \\ \boldsymbol{T}_n \, \boldsymbol{T}_{n-1} \dots \boldsymbol{T}_1 \end{bmatrix},$$

which consists of $2n-1$ arbitrary matrices $\boldsymbol{T}_i \in \mathbb{C}^{m\times m}$, we define the mapping $\mathfrak{G}_{n,m} : \mathbb{C}^{nm\times nm} \to \mathbb{C}^{(2n-1)m\times(2n-1)m}$, which extends the block Toeplitz matrix to a block circulant matrix by

$$\boldsymbol{T} \mapsto \begin{bmatrix} \boldsymbol{T}_1 & \boldsymbol{T}_{2n-1} \dots \boldsymbol{T}_{n+1} \dots & \boldsymbol{T}_2 \\ \boldsymbol{T}_2 & \boldsymbol{T}_1 \quad \ddots \quad \vdots & \boldsymbol{T}_3 \\ \vdots & \ddots \quad \ddots \quad \vdots & \\ \boldsymbol{T}_n & \boldsymbol{T}_{n-1} \dots \boldsymbol{T}_1 & \vdots \\ \vdots & & \ddots \\ \boldsymbol{T}_{2n-1} & \boldsymbol{T}_{2n-2} \qquad \dots & \boldsymbol{T}_1 \end{bmatrix}.$$

It is worth noting that for each n it holds that $\mathfrak{G}_{n,1}(\boldsymbol{T}) = \boldsymbol{\Gamma}\left(\boldsymbol{\Theta}^{-1}(\boldsymbol{T})\right)$ for all Toeplitz matrices $\boldsymbol{T}$.

To finalize the embedding of the matrix $\boldsymbol{T}_{(\boldsymbol{n},d)}$ into a multilevel circulant matrix, we iteratively apply the appropriate $\mathfrak{G}_{n_i,m_j}$ via

$$\mathfrak{T}(\boldsymbol{T}_{(\boldsymbol{n},d)}) = \mathfrak{G}_{n_1,K} \left\{ \begin{bmatrix} \mathfrak{T}_\ell(\boldsymbol{T}_{(1,\boldsymbol{m}),\ell}) & \dots \mathfrak{T}_\ell(\boldsymbol{T}_{(n_1+1,\boldsymbol{m}),\ell}) \\ \mathfrak{T}_\ell(\boldsymbol{T}_{(2,\boldsymbol{m}),\ell}) & \dots \mathfrak{T}_\ell(\boldsymbol{T}_{(n_1+2,\boldsymbol{m}),\ell}) \\ \vdots \quad \ddots & \vdots \\ \mathfrak{T}_\ell(\boldsymbol{T}_{(n_1,\boldsymbol{m}),\ell}) \dots & \mathfrak{T}_\ell(\boldsymbol{T}_{(1,\boldsymbol{m}),\ell}) \end{bmatrix} \right\}$$

where $K = n_2 \cdot \dots \cdot n_d$. The matrix is by design d-level circulant and can be diagonalized as in (8). For an appropriately chosen index set S

$$\mathfrak{T}(\boldsymbol{T}_{(\boldsymbol{n},d)})_S = \boldsymbol{T}_{(\boldsymbol{n},d)} \tag{9}$$

holds. The set S can be constructed iteratively by keeping track of inserted spurious columns and rows into $\mathfrak{T}(\boldsymbol{T}_{(\boldsymbol{n},d)})$ during the above embedding procedure, compared to the original $\boldsymbol{T}_{(\boldsymbol{n},d)}$.

E. Block Multilevel Toeplitz Matrices

As outlined in Section II, the matrix $\boldsymbol{H}$ has a block structure, where each block $\boldsymbol{H}_{i,j}$ for $i, j = 0, \dots, M-1$ is a 2-level Toeplitz matrix, which is why we call $\boldsymbol{H}$ block multilevel Toeplitz. As such, we can collect the unique defining elements of $\boldsymbol{H}$ in $\boldsymbol{h} \in \mathbb{R}^{M\times M\times 2N_x-1\times 2N_y-1}$ and then set

$$\boldsymbol{H}_{i,j} = \boldsymbol{T}_{([N_x,N_y],2)}(\boldsymbol{h}_{i,j}).$$

With the methods of the previous section we can diagonalize each $\boldsymbol{H}_{i,j}$. To this end, we embed each $\boldsymbol{H}_{i,j}$ into a 2-level circulant matrix

$$\begin{bmatrix} \mathfrak{T}_2(\boldsymbol{H}_{0,0}) & \dots & \mathfrak{T}_2(\boldsymbol{H}_{0,M-1}) \\ \vdots & \ddots & \vdots \\ \mathfrak{T}_2(\boldsymbol{H}_{M-1,0}) & \dots & \mathfrak{T}_2(\boldsymbol{H}_{M-1,M-1}) \end{bmatrix} = \boldsymbol{K}^{\mathrm{H}} \cdot \boldsymbol{D} \cdot \boldsymbol{K} \tag{10}$$

for $\boldsymbol{F} = \boldsymbol{\mathcal{F}}_{2N_x-1} \otimes \boldsymbol{\mathcal{F}}_{2N_y-1}$, $\boldsymbol{K} = \boldsymbol{I}_M \otimes \boldsymbol{F}$ and

$$\boldsymbol{D} = \begin{bmatrix} \operatorname{diag}(\boldsymbol{F} \operatorname{vec} \tilde{\boldsymbol{h}}_{0,0}) & \dots & \operatorname{diag}(\boldsymbol{F} \operatorname{vec} \tilde{\boldsymbol{h}}_{0,M-1}) \\ \vdots & \ddots & \vdots \\ \operatorname{diag}(\boldsymbol{F} \operatorname{vec} \tilde{\boldsymbol{h}}_{M-1,0}) & \dots & \operatorname{diag}(\boldsymbol{F} \operatorname{vec} \tilde{\boldsymbol{h}}_{M-1,M-1}) \end{bmatrix},$$

where each $\tilde{\boldsymbol{h}}_{i,j}$ for $i, j = 0, \dots, M-1$ is chosen such that they contain the defining elements of the corresponding 2-level circulant matrix $\mathfrak{T}_2(\boldsymbol{H}_{i,j})$ in the 2D frequency domain. Because of the block diagonal structure of $\boldsymbol{H}$ in the basis $\boldsymbol{K}$, which posses an efficient transform in and from this basis, the whole matrix $\mathfrak{T}_2(\boldsymbol{H}_{i,j})$ posses an efficient matrix-vector multiplication.

F. Implementation

Recently, the authors' group released a package called `fastmat` [18] for the Python programming language, which provides means of working with structured matrices in a very convenient and efficient way, by hiding the implementational complexity given rise by the need for fast matrix-vector products from the user. Besides, `fastmat` also offers various SSR Algorithms like OMP and FISTA. The here described efficient algorithms in eqs. (4) and (7) to (10), including the discussed optimizations with respect to the FFT, are implemented there. The corresponding code [19] is freely available. All simulations and performance evaluations in this paper are carried out with this package.

IV. Ultrasound Reconstruction

In this section, we first derive the structure of the dictionary $\boldsymbol{H}$ and then exploit it by using its forward and backward transform in an SSR algorithm.

Lemma IV.1. *The matrix $\boldsymbol{H} \in \mathbb{R}^{MN_xN_y\times MN_xN_y}$ from* (2) *is block 2-level Toeplitz, where for the generating elements $\boldsymbol{h}$ it holds that $\boldsymbol{h} \in \mathbb{R}^{M\times M\times 2N_x-1\times 2N_y-1}$.*

Proof. We consider a column $\boldsymbol{H}_{z_1,x_1,y_1}$ of $\boldsymbol{H}$ with $(z_1, x_1, y_1) \in G_{3D}$. If we now pick an arbitrary $(z_2, x_2, y_2) \in G_{3D}$, we see that

$$[\boldsymbol{H}_{z_1,x_1,y_1}]_{z_2,x_2,y_2} = f(z_1, z_2, x_1 - x_2, y_1 - y_2),$$

for $f(u, v, x, y) = \frac{1}{c} t(x, y, u) \cdot h(v - t(x, y, u))$ with $t(x, y, z) = \sqrt{x^2 + y^2 + z^2}$. So the elements in a specific block of $\boldsymbol{H}$, which corresponds to the pair (u, v) and reads as

$$([\boldsymbol{H}_{u,i,j}]_{v,k,\ell})_{(i,j)\in G_{2D},(k,\ell)\in G_{2D}}$$

is 2-level Toeplitz because of the translational invariance with respect to $x_{1,2}$ and $y_{1,2}$. Then, the asserted structures of $\boldsymbol{H}$ and $\boldsymbol{h}$ follow easily. □

The computational advantage one obtains from exploiting this structure with respect to matrix-vector products is depicted in fig. 1, where we compare the performance of the `fastmat`

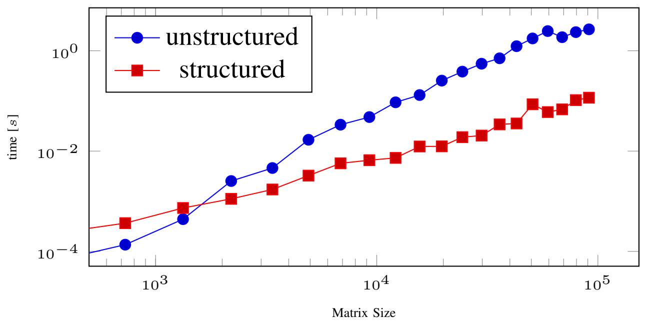

Figure 1. Comparison of matrix-vector multiplication with the block 2-level Toeplitz matrices in terms of computation time.

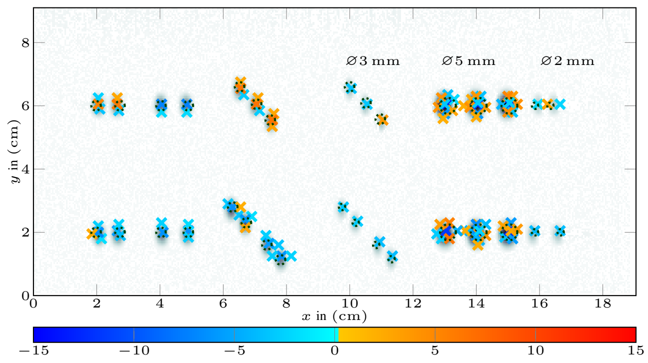

Figure 2. Reconstruction results: The OMP reconstruction (colored crosses) and the correct positions of the FBHs (dotted circles) are plotted on top of a 3D SAFT image. The color marks the amplitudes of the OMP support.

implementation of Equation (10) to the standard matrix-vector multiplication offered by `numpy` [20], which is very prevalent in scientific computing with Python. Starting at a matrix size > 1500 the `fastmat` version outperforms the standard numpy version significantly. The higher computational effort for lower matrix sizes stems from the fact that the efficient implementation introduces a constant overhead, which becomes negligible for larger problems. Next, we present SSR reconstructions of a volumetric ultrasound measurement of a steel specimen containing 32 Flat Bottom Holes (FBH) with various diameters depicted in fig. 2 as dotted circles. The measurements were acquired using a $4\,\mathrm{MHz}$ transducer moved on a grid with $\Delta x = \Delta y = 0.5\,\mathrm{mm}$ and sampled at $20\,\mathrm{MHz}$ leading to a data set of size $181 \times 381 \times 182$. Further, we assume a constant speed of sound $c = 5900\,\mathrm{m/s}$ and $g(\tau_{x,y}(x_n, y_n, z_n)) = \mathrm{e}^{-\gamma_{z_n}\tau_{x,y}(x_n,y_n,z_n)}$. The parameter γ_{z_n} depends on the beam width of the transducer.

The reconstruction is performed using OMP stopped after 110 iterations. We run more iterations than defects present, since due to the model mismatch, most defects require more than one support element in the solution to account for the energy at the defects position. The result is shown as colored cross marks in fig. 2. The reconstruction was performed on a High Performance Computing Cluster requiring an average memory of $333\,\mathrm{GB}$. Utilizing 16 CPU cores, a single OMP iteration takes about $3\,\mathrm{min}$. OMP reconstructs all FBHs. Note that the difference between the two groups of $5\,\mathrm{mm}$ holes is that the upper group has the same depth, while the lower group has three different depths. For comparison, a 3D SAFT reconstruction of the same measurement using Stolt's migration [21] is put underneath the OMP reconstruction in fig. 2.

V. Conclusion

In conclusion, as one can see, the matrix-free SSR approach yields similar results as 3D SAFT, but due to the freedom one can exploit when designing $\boldsymbol{H}$ in eq. (3) the reconstruction process can easily be adapted to more complex specimen geometries, physical propagation effects, flaw shapes, and measurement setups without changing the reconstruction method, whereas for SAFT the implementation has to be specifically tailored to a certain scenario. Last, but not least, one can easily extend the SSR approach to a CS [9] scheme allowing data reduction, which we leave to further publications.

References

[1] C. Holmes, B. Drinkwater, and P. Wilcox, "The post-processing of ultrasonic array data using the total focusing method," *Insight-NDT and Condition Monitoring*, vol. 46, no. 11, pp. 677–680, 2004.

[2] A. Tweedie, R. L. O'Leary, G. Harvey, A. Gachagan, C. Holmes, P. D. Wilcox, and B. W. Drinkwater, "Total focussing method for volumetric imaging in immersion non destructive evaluation," in *IEEE IUS*, Oct 2007.

[3] M. Spies, H. Rieder, A. Dillhöfer, V. Schmitz, and W. Müller, "Synthetic aperture focusing and time-of-flight diffraction ultrasonic imaging—past and present," *Journ. of NDE*, vol. 31, pp. 310–323, 2012.

[4] A. Tuysuzoglu, J. M. Kracht, R. O. Cleveland, M. C¸etin, and W. C. Karl, "Sparsity driven ultrasound imaging," *The Journal of the Acoustical Society of America*, vol. 131, no. 2, pp. 1271–1281, 2012.

[5] J. Kirchhof, F. Krieg, F. Römer, A. Ihlow, A. Osman, and G. Del Galdo, "Sparse Signal Recovery for Ultrasonic Detection and Reconstruction of Shadowed Flaws," in *IEEE ICASSP*, March 2017.

[6] H. Wu, J. Chen, S. Wu, H. Jin, and K. Yang, "A model-based regularized inverse method for ultrasonic B-scan image reconstruction," *Measurement Science and Technology*, vol. 26, no. 10, p. 105401, 2015.

[7] G. David, J.-l. Robert, B. Zhang, and A. F. Laine, "Time domain compressive beam forming of ultrasound signals," *Journ. of the Acoust. Soc. of America*, vol. 137, no. 5, pp. 2773–2784, 2015.

[8] M. Karimi, P. Croaker, and N. Kessissoglou, "Acoustic scattering for 3D multi-directional periodic structures using the boundary element method," *Journ. of the Acoust. Soc. of America*, vol. 141, no. 1, pp. 313–323, 2017.

[9] E. J. Candes and T. Tao, "Near-optimal signal recovery from random projections: Universal encoding strategies?" *IEEE Trans. Inf. Theor.*, vol. 52, no. 12, pp. 5406–5425, Dec. 2006.

[10] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm with application to wavelet-based image deblurring," in *IEEE ICASSP*, April 2009, pp. 693–696.

[11] Y. C. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal Matching Pursuit: recursive function approximation with applications to wavelet decomposition," in *27th Asil. Conf. Signals, Systems Comp.*, Nov 1993.

[12] S. Foucart and H. Rauhut, *A mathematical introduction to compressive sensing*, 1st ed. Birkhäuser Basel, 2013.

[13] P. J. Davis, *Circulant Matrices*, 1st ed., ser. Pure & Applied Mathematics. John Wiley & Sons Inc, 1979.

[14] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. of Comp.*, vol. 19, no. 90, 1965.

[15] A. Cortés, I. Vélez, and F. Sevillano, Juan, "Radik r^k FFTs: Matricial representation and SDC/SDF pipeline implementation," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2824–2839, 2009.

[16] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[17] P. Fernandes, B. Plateau, and W. J. Stewart, "Efficient descriptor-vector multiplications in stochastic automata networks," *J. ACM*, vol. 45, 1998.

[18] C. Wagner and S. Semper, "Fast linear transformations in python," arXiv:1710.09578, 2017.

[19] S. Semper and C. Wagner, "fastmat," https://github.com/EMS-TU-Ilmenau/fastmat, 2017.

[20] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[21] R. H. Stolt, "Migration by Fourier transform," *Geophysics*, vol. 43, Feb. 1978.