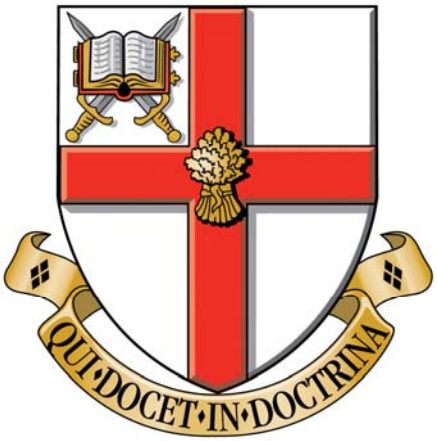

University of Chester

Author(s): Jason A Roberts ; Dmitry V Savostyanov ; Eugene E Tyrtyshnikov

Title: Superfast solution of linear convolutional Volterra equations using QTT approximation

Date: April 2014

Originally published in: Journal of Computational and Applied Mathematics

Example citation: Roberts, J.A., Savostyanov, D. V., & Tyrtyshnikov, E. E. (2014). Superfast solution of linear convolutional Volterra equations using QTT approximation. *Journal of Computational and Applied Mathematics, 260*, 434-448. http://dx.doi.org/10.1016/j.cam.2013.10.025

Version of item: Author's accepted manuscript

Available at: http://hdl.handle.net/10034/336402

Superfast solution of linear convolutional Volterra equations using QTT approximation

Jason A. Roberts[a], Dmitry V. Savostyanov[b,a,1], Eugene E. Tyrtyshnikov[b,1]

[a] *University of Chester, Parkgate Road, Chester, CH1 4BJ, UK*
[b] *Institute of Numerical Mathematics, Russian Academy of Sciences, Gubkina 8, Moscow, 119333, Russia*

Abstract

We address a linear fractional differential equation and develop effective solution methods using algorithms for inversion of triangular Toeplitz matrices and the recently proposed QTT format. The inverses of such matrices can be computed by the divide and conquer and modified Bini's algorithms, for which we present the versions with the QTT approximation. We also present an efficient formula for the shift of vectors given in QTT format, which is used in the divide and conquer algorithm. As the result, we reduce the complexity of inversion from the fast Fourier level $\mathcal{O}(n \log n)$ to the speed of superfast Fourier transform, i.e., $\mathcal{O}(\log^2 n)$. The results of the paper are illustrated by numerical examples.

Keywords: fractional calculus, Caputo derivative, triangular Toeplitz matrix, divide and conquer, tensor train format, QTT, fast convolution, superfast Fourier transform
2010 MSC: 15A69, 26A33, 45E10, 65F05

1. Introduction

Equations involving derivatives of fractional order are of great importance, due to their role in mathematical models applied in mechanics, biochemistry, electrical engineering, medicine, etc., see [1, 2, 3]. In this paper we present a superfast algorithm for the numerial solution of the linear equation

$$D_*^\alpha y(t) = F(t, y(t)) = my(t) + f(t), \quad 0 \le t \le T, \quad y(0) = y_0, \tag{1}$$

where $0 < \alpha < 1$ is the order of the fractional operator, $m \in \mathbb{R}$ is a constant referred to as *mass*, and $f(t)$ is a sufficiently well–behaved *forcing* term. For $\alpha = 1/2$ this equation is a scalar version of the Bagley-Torvik equation [4], which is used in the modelling of viscoelastic materials. The definitions of Caputo derivative D_*^α can be found in many sources, e.g. [5, 6], and are presented in appendix for the convenience.

Email addresses: j.roberts@chester.ac.uk (Jason A. Roberts), dmitry.savostyanov@gmail.com (Dmitry V. Savostyanov), eugene.tyrtyshnikov@gmail.com (Eugene E. Tyrtyshnikov)

[1] During this work D. V. Savostyanov and E. E. Tyrtyshnikov were supported by the Leverhulme Trust to visit, stay and work at the University of Chester, as the Visiting Research Fellow and the Visiting Professor, respectively. Their work was also supported in part by RFBR grants 11-01-00549, 12-01-91333-nnio-a, and Russian Federation Government Contracts 14.740.11.0345, 14.740.11.1067, 16.740.12.0727

The classical result of Diethelm [5, Lem. 6.2] allows us to rewrite (1) in the form

$$y(t) = y_0 + \frac{1}{\Gamma(\alpha)} \int_0^t (t-s)^{\alpha-1} \left(m y(s) + f(s) \right) ds, \tag{2}$$

where $\Gamma(\alpha) = \int_0^\infty e^{-t} t^{\alpha-1} dt$ is the gamma function. Eq. (2) is the weakly singular convolutional Volterra equation of the second kind with the Abel–type kernel. Volterra equations of second kind are well-studied and are proven to have a unique continuous solution for $0 \le t \le T$, see, e.g. [7, Thm. 3.2]. The solution is asymptotically stable if $m < 0$ (see [8]) which we will always assume in this paper.

For certain forcing terms, the solution of (2) can be found using series methods. In a general framework, we can discretize (2) using a collocation or Galerkin method and numerically solve the resulted linear system. This matrix approach to fractional calculus was brilliantly presented by I. Podlubny in [9]. In this paper we consider the collocation method and assume that $y(t)$ is approximated by a piecewise–linear function on a uniform grid $t_j = jh$, $j = 0, \dots, n$, where $h = T/n$. The stability of collocation methods for fractional equations was studied in [10, 11] and an error analysis can be found in [12]. The discretized equation is the following

$$y_j = y_0 + \frac{h^\alpha}{\Gamma(\alpha)} \sum_{k=0}^{j} w_{j,k} (m y_k + f_k), \qquad j = 1, \dots, n,$$

where $y_j = y(t_j)$, $f_k = f(t_k)$ and $w_{j,k}$ are quadrature weights, defined by integration of piecewise–linear basis functions with Abel-type kernel, i.e.,

$$w_{j,k} = \frac{1}{\alpha(\alpha+1)} \begin{cases} (j-1)^{\alpha+1} - (j-\alpha-1) j^\alpha, & k = 0, \\ (j-k-1)^{\alpha+1} - 2(j-k)^{\alpha+1} + (j-k+1)^{\alpha+1}, & 1 \le k < j, \\ 1, & k = j. \end{cases}$$

Finally, we obtain the linear system $Ay = b$ with triangular Toeplitz matrix and the right-hand side defined as follows,

$$\sum_{k=1}^{j} a_{j-k} y_k = b_j, \qquad j = 1, \dots, n, \tag{3}$$

$$a_p = \begin{cases} 1 - \gamma m, & p = 0, \\ -\gamma m \left((p-1)^{\alpha+1} - 2p^{\alpha+1} + (p+1)^{\alpha+1} \right), & p > 0, \end{cases}$$

$$b_j = y_0 + \gamma \left(\sum_{k=1}^{j} w_{j,k} f_k + w_{j,0} (m y_0 + f_0) \right),$$

where $\gamma = h^\alpha / \Gamma(\alpha+2)$.

The numerical scheme we use is analogous to the fractional Adams method proposed in [12] for a general (e.g. nonlinear) function $F(t, y(t))$. The method is developed as a generalization of the Adams–Bashforth–Moulton scheme from the classical numerical analysis of ordinary differential equations and a detailed error analysis is provided. The complexity of the fractional Adams method in the nonlinear case is $\mathcal{O}(n^2)$. To reduce this complexity, we can take into account the decay speed of the Abel kernel $k(s) = s^{\alpha-1}$ of the integral in (2). The so-called *fixed memory principle* [6, 13] and more accurate *nested mesh method* [14, 15] are based on truncation

and approximation of the tail of the integral (2), respectively, and have almost linear complexity w.r.t. n. We revise these methods in Sec. 2.

For linear $F(t, y(t))$, the problem writes as the linear system (3), which can be solved using well–developed algorithms for the inversion of triangular Toeplitz matrices, or triangular strip matrices, as they are referred in [9]. These methods are recalled in Sec. 3, and have the asymptotic complexity of the fast Fourier transform (FFT) algorithm, which is $\mathcal{O}(n \log n)$.

Recently, a superfast Fourier transform algorithm was proposed in [16], based on the approximation of vectors in the *quantized tensor train* (QTT) format [17, 18]. The method can be considered as a classical model of quantum superfast Fourier transform algorithm [19], and has a square-logarithmical complexity $\mathcal{O}(\log^2 n)$ for a certain class of vectors, for which such a model is efficient. This class of vectors is partially established in [20] and include, for example, vectors with sparse Fourier image. The numerical experiments provided in Sec. 4 show that the Abel kernel $t(s) = s^{1-\alpha}$ is efficiently approximated by the QTT format for all $0 < \alpha < 1$ with accuracy up to the machine threshold. Based on this observation, we propose the superfast inversion algorithm for the triangular Toeplitz matrix (3), using the QTT approximation.

The numerical experiments provided in Sec. 5 justify the accuracy and sublinear complexity of the method proposed.

2. Numerical method with logarithmic memory

In [6] and [13] the author describes an approach to the numerical integration involved in solving a fractional problem whereby the first part (or tail) of the integral is ignored (i.e. assuming the value of the integral over this region is negligible) and so the memory of the system is truncated at some point. The error introduced via this process is described in [6] for Riemann-Liouville fractional derivatives. In [14] the authors consider the error that is introduced when this approach is applied to problems expressed with respect to the Caputo fractional derivative. The authors show that by introducing a finite memory of fixed length T for the Caputo derivative we introduce an error of the form

$$E = \left| \frac{1}{\Gamma(1-\alpha)} \int_0^{t-T} \frac{y'(s)}{(t-s)^\alpha} ds \right|. \tag{4}$$

Letting $\sup_{s\in[0,t]} |y'(s)| = M$ then

$$E \leq \frac{M\left(t^{1-\alpha} - T^{1-\alpha}\right)}{\Gamma(2-\alpha)}. \tag{5}$$

So for a fixed memory $T < t$ we have a loss of order such that the error does not tend towards zero as the stepsize approaches zero. Indeed, the authors in [14] highlight that in order to preserve the order of the method we would need to choose T so that (for a fixed error bound E) we have

$$T^{1-\alpha} \geq t^{1-\alpha} - \left(\frac{E\Gamma(2-\alpha)}{M} \right), \tag{6}$$

which introduces a computational cost — precisely what the fixed memory principle is trying to avoid.

Figure 1: Example of the grids used at subsequent steps of the nested mesh method (from top to bottom). On each line the active points of the grid are shown by black and non-active by grey dots.

To overcome this it is proposed in [14], and described further in [15], that the fixed memory principle is amended so that the region of integration $[0,t]$ is decomposed into a sequence of finite-length intervals with differing stepsizes. So as we move 'backwards' along the interval from t to 0 the subintervals use coarser and coarser stepsizes, except possibly for some small sub-interval near zero due to the length of this subinterval not being an exact multiple of the current stepsize — in such circumstances the authors suggest a couple of alternative approaches for this subinterval, one such alternative being the use of the original stepsize. Such a *nested mesh* approach (see actual mesh on Fig. 1) is possible due to the scaling properties of the fractional integral, which are discussed in [15] and [14]. Thus the weights (of the Adams–type method described earlier) for calculating $\Omega_h^\alpha f(nh) \approx I^\alpha f(nh)$ with a stepsize h can be used to calculate $\Omega_{\omega^p h}^\alpha f\left(\omega^p nh\right) \approx I^\alpha f\left(n\omega^p h\right)$ using a stepsize of $\omega^p h$. The authors [14] define, for $h \in \mathbb{R}^+$, the mesh M_h by $M_h = \{hn, n \in \mathbb{N}\}$. If $\omega, r, p \in \mathbb{N}$, $\omega > 0$, $r > p$, then $M_{\omega^r h} \subset M_{\omega^p h}$. The authors then decompose the interval $[0,t]$, for fixed $T > 0$ in the following way:

$$[0,t] = [0, t-\omega^m T] \cup [t-\omega^m T, t-\omega^{m-1}T] \cup \cdots \cup [t-\omega T, t-T] \cup [t-T, t] \quad (7)$$

where $m \in \mathbb{N}$ is the smallest integer such that $t < \omega^{m+1}T$. A step length of h is used over the most recent time interval $[t-T,t]$ with successively larger step sizes over earlier intervals, as follows. Let $t, T, h \in \mathbb{R}$, $\omega^{m+1}T > t \geq \omega^m T$, $t > 1$, $h > 0$ with $t = nh$ for some $n \in \mathbb{N}$. The integral can be rewritten as

$$\begin{aligned} I_{[0,t]}^\alpha f(t) &= I_{[t-T,t]}^\alpha f(t) + \sum_{i=0}^{m-1} I_{[t-\omega^{i+1}T, t-\omega^i T]}^\alpha f(t) + I_{[0,t-\omega^m T]}^\alpha f(t) \quad (8) \\ &= I_{[t-T,t]}^\alpha f(t) + \sum_{i=0}^{m-1} \omega^{i\alpha} I_{[t-\omega T, t-T]}^\alpha f\left(\omega^i t\right) + \omega^{m\alpha} I_{[0,t-\omega^m T]}^\alpha f\left(\omega^m t\right), \quad (9) \end{aligned}$$

where $I_{[t-a,t-b]}^\alpha f(t) = \frac{1}{\Gamma(\alpha)} \int_{t-a}^{t-b} \frac{f(s)}{(t-s)^{1-\alpha}} ds$. The authors also show the following:

Theorem 2.1. *[14] The nested mesh scheme preserves the order of the underlying rule on which it is based.*

In addition, whilst the computational cost of the full-memory approach is of order $O(N^2)$, the nested-mesh approach has order $O(N \log N)$.

3. Inversion of triangular Toeplitz matrices

3.1. Basic properties of triangular Toeplitz matrices

Let $\mathcal{T}_n$ be a set of lower triangular Toeplitz $n \times n$ matrices[2], i.e.,

$$A \in \mathcal{T}_n \quad \Leftrightarrow \quad A = [a(j,k)]_{j,k=0}^{n-1}, \quad a(j,k) = a(j-k), \quad a(p) = 0, \quad p < 0.$$

It is easy to check the following properties of $\mathcal{T}_n$.

1. $A \in \mathcal{T}_n,\ B \in \mathcal{T}_n \ \Rightarrow\ AB \in \mathcal{T}_n$;
2. $A \in \mathcal{T}_n,\ B \in \mathcal{T}_n \ \Rightarrow\ AB = BA$;
3. $A \in \mathcal{T}_n,\ a_0 \neq 0 \ \Rightarrow\ A^{-1} \in \mathcal{T}_n$.

By the last property, the inverse matrix $B = A^{-1}$, as well as all matrices from $\mathcal{T}_n$, is defined by its first column. The standard solution method for triangular linear systems has complexity $\mathcal{O}(n^2)$ and yields the following trivial formula

$$b(0) = \frac{1}{a(0)}, \qquad b(j) = -\frac{1}{a(0)} \sum_{k=1}^{j} b(j-k)a(k), \quad j = 1, \ldots, n-1. \tag{10}$$

For $A, B \in \mathcal{T}_n$, the product $X = AB \in \mathcal{T}_n$ and is also defined by the first column $x = Ab$. Therefore, matrix-by-matrix multiplication in $\mathcal{T}_n$ is equivalent to the multiplication of a vector by the Toeplitz matrix, i.e., discrete *convolution* $x(j) = \sum_{k=0}^{j} a(j-k)b(k)$. A naive computation by this formula requires $\mathcal{O}(n^2)$ operations, but it is well-known that it can be computed in $\mathcal{O}(n \log n)$ operations using the fast Fourier transform (FFT) algorithm [21, 22]. To recall this, we note that each $n \times n$ Toeplitz matrix T is the leading submatrix of some $2n \times 2n$ circulant matrix

$$C = \begin{bmatrix} T & * \\ * & T \end{bmatrix}, \qquad C = [c(j,k)], \quad \text{where} \quad c(j,k) = c(j-k \mod 2n),$$

and all circulant matrices are diagonalized by unitary Fourier matrix as follows (see cf. [23])

$$C = F^* \Lambda F, \qquad \Lambda = \sqrt{2n}\, \mathrm{diag}(Fc).$$

Therefore, multiplication by C and hence by T can be performed by 3 FFTs of size $2n$ with complexity $\mathcal{O}(n \log n)$.

The inversion of triangular Toeplitz matrices has asymptotically the same complexity, i.e., $cM(n)$, where $M(n)$ denotes the complexity of matrix multiplication. The modern highly-improved inversion algorithms reduce the constant to $c = 1.4 + o(1)$ see, e.g. [24]. We now recall the classical algorithms, which have slightly larger constant c, but are much more simple and easy to follow. In Sec. 4 we will adjust the classical inversion algorithms to use the compressed format for the approximate representation of matrix, reducing the complexity to *sublinear* w.r.t. n.

[2]Here and further we write matrix and vector indices in round brackets instead of putting them as subscripts, in order to introduce the convenient notation for QTT representation later.

3.2. Divide and conquer method

To benefit from the Toeplitz structure and reach $\mathcal{O}(n \log n)$ complexity for the inversion algorithm, we can use the divide-and-conquer strategy. This was noted in [25] and developed in [26]. It is easy to check that if $2n \times 2n$ lower triangular Toeplitz matrix $A' \in \mathcal{T}_{2n}$ is partitioned to $n \times n$ matrices, the inverse matrix writes as follows

$$A' = \begin{bmatrix} A & \\ C & A \end{bmatrix}, \qquad (A')^{-1} = \begin{bmatrix} A^{-1} & \\ -A^{-1}CA^{-1} & A^{-1} \end{bmatrix} = \begin{bmatrix} A^{-1} & \\ & A^{-1} \end{bmatrix} \begin{bmatrix} I & \\ -CA^{-1} & I \end{bmatrix}, \tag{11}$$

where $A \in \mathcal{T}_n$, $A^{-1} \in \mathcal{T}_n$ and C is a Toeplitz matrix. If $n = 2^d$ and $A_d \in \mathcal{T}_{2^d}$, this formula yields the reccurent method to compute A_d^{-1}. We start from some small d_0 and use (10) to compute the inverse of $2^{d_0} \times 2^{d_0}$ leading submatrix $A_{d_0} \in \mathcal{T}_{2^{d_0}}$. Then we subsequently apply (11) and compute A_d^{-1} in $(d - d_0)$ steps. Each step requires to compute the first column of $A_t^{-1}C_tA_t^{-1}$ with $2^t \times 2^t$ Toeplitz matrices A_t^{-1} and C_t, where $t = d_0 + 1, \ldots, d$. Each multiplication is done in $\mathcal{O}(t2^t)$ operations, which summarizes to $\mathcal{O}(d2^d) = \mathcal{O}(n \log n)$ overall complexity. More accurately, the cost of the divide and conquer algorithm is smaller than 12 FFTs of size n.

3.3. Bini's and related approximate methods

In order to reduce the number of FFTs used in computations and obtain algorithm with better parallel performance, the approximate method to compute A^{-1} for $A \in \mathcal{T}_n$ was proposed in [27]. It is noted that $\mathcal{T}_n$ is the algebra generated by the matrix $H \in \mathcal{T}_n$ with unit elements on the subdiagonal and zeros elsewhere, i.e., transposed Jordan block with zero diagonal. Therefore, $A \in \mathcal{T}_n$ with first column $a = [a(j)]_{j=0}^{n-1}$ is written as $A = \sum_{j=0}^{n-1} a(j)H^j$. The idea is to add a small element ε^n at the top right corner of the matrix and substitute H by $H_\varepsilon = H + \varepsilon^n e_0 e_{n-1}^T$. It is easy to check that $D_\varepsilon H_\varepsilon D_\varepsilon^{-1} = \varepsilon C$, where $D_\varepsilon = \operatorname{diag}\{\varepsilon^j\}_{j=0}^{n-1}$, and $C = H_1 = H + e_0 e_{n-1}^T$ generates the algebra of circulant $n \times n$ matrices. Then A and A^{-1} are approximated as follows

$$\begin{gathered} A \approx \tilde{A}_\varepsilon = \sum_{j=0}^{n-1} a(j)H_\varepsilon^j = D_\varepsilon^{-1} \left(\sum_{j=0}^{n-1} a(j)\varepsilon^j C_j \right) D_\varepsilon = D_\varepsilon^{-1} C_\varepsilon D_\varepsilon = D_\varepsilon^{-1} F^* \Lambda_\varepsilon F D_\varepsilon, \\ A^{-1} \approx \tilde{A}_\varepsilon^{-1} = D_\varepsilon^{-1} F^* \Lambda_\varepsilon^{-1} F D_\varepsilon, \qquad \text{where} \quad \Lambda_\varepsilon = \sqrt{n}\operatorname{diag}(F a_\varepsilon), \quad a_\varepsilon(j) = a(j)\varepsilon^j. \end{gathered} \tag{12}$$

The first column of A_ε^{-1} is computed using two FFTs of size n.

This idea was revised in [28], where it was proposed to apply Bini's algorithm to the first column a of matrix A padded with n zero elements. The revised version of Bini's algorithm requires two FFTs of size $2n$ and has better accuracy properties.

3.4. Newton iteration

The classical Newton iteration

$$B_{k+1} = 2B_k - B_k A B_k \tag{13}$$

was proposed in [29] for the computing the inverse A^{-1} of a nonsingular matrix A. It converges quadratically if initial guess B_0 is s.t. $\|I - AB_0\| \leq 1$ in any operator norm of a matrix. In [30] it is shown that for $B_0 = \mu A^*$ with some small real μ

Newton iteration converges to the inverse A^{-1} of a nonsingular or pseudoinverse $A^{\dagger}$ of a singlular matrix A. In [31] further deep analysis is provided, for instance, it is shown that $\mu^{-1} = \|A\|_1 \|A\|_\infty$ is a good and reliable choice. In relation to Toeplitz and related structured matrices, the Newton iteration with approximation of the result on each step was developed using the concept of displacement ranks [32] and tensor product approximations [33, 34, 35].

For $A \in \mathcal{T}_n$ the choice of initial guess $B_0 = \mu A^*$ is not effective, since $A^* \notin \mathcal{T}_n$ and we can not perform iterations with $B_k \in \mathcal{T}_n$, which grants low storage and fast multiplication.

If $B_0 \in \mathcal{T}_n$, every Newton iteration costs two convolutions, i.e., 6 FFTs of size $2n$.

Remark 1. *A single Newton iteration for lower triangular Toeplitz matrices is slower than the divide and conquer method.*

It is not easy to provide a good initial guess $B_0 \in \mathcal{T}_n$ for which the Newton iteration with a given matrix $A \in \mathcal{T}_n$ converges in one or few steps. However, Newton iteration can be used to improve the accuracy of matrix $B \approx A^{-1}, B \in \mathcal{T}_n$ computed by other means, if $\|I - AB\| \leq 1$. For instance, we can note the following relation between the divide and conquer method and Newton iteration.

Remark 2. *For matrix $A' \in \mathcal{T}_{2n}$ defined in* (11)*, the Newton iteration* (13) *with initial guess*

$$B_0 = \begin{bmatrix} A^{-1} & 0 \\ 0 & 0 \end{bmatrix}, \qquad A \in \mathcal{T}_n,$$

gives $B_1 = (A')^{-1}$, i.e., converges in one step and is equivalent to the divide and conquer method (11)*.*

Therefore, for $A \in \mathcal{T}_n$, divide and conquer method is always better than the Newton iteration, which reduces to the divide and conquer method in the special case.

3.5. Decay of the elements of inverse matrix

It is instructive to look at the decay profiles of the elements of a triangular Toeplitz matrix (3) and its inverse, see Fig. 2. There is a jump in magnitude between diagonal and subdiagonal elements, i.e.,

$$\frac{a_0}{a_1} = \frac{1 - (\gamma m)^{-1}}{2^{\alpha+1} - 2}, \qquad \gamma = \frac{h^\alpha}{\Gamma(\alpha + 2)},$$

where the numerator increases when $n \to \infty$, $h \to 0$ and tends to one when $m \to -\infty$. After the jump, elements decay polynomially, i.e., $a_p \sim p^{\alpha-1}$ for $p \geq 1$. For the inverse matrix the behaviour is the same for a certain (possibly very long) set of elements. However, after certain point the rate of decay changes from $1 - \alpha$ to $1 + \alpha$, i.e., $b_p \sim p^{-\alpha-1}$ for $p \geq P$. The bend point P which is obtained from the experiment, is the monotonically decreasing function $P = P(\gamma m)$, i.e. the larger is the initial jump, the later the decay of element of the inverse matrix switches to faster rate. The observed behaviour of elements of inverse matrix allows us to predict the upper bound for the norm of the second half of vector, using the information about the first half. We will use this property in the next subsection, where the divide and conquer algorithm will be adapted for the vectors approximately given in the low–parametrical tensor–structured format.

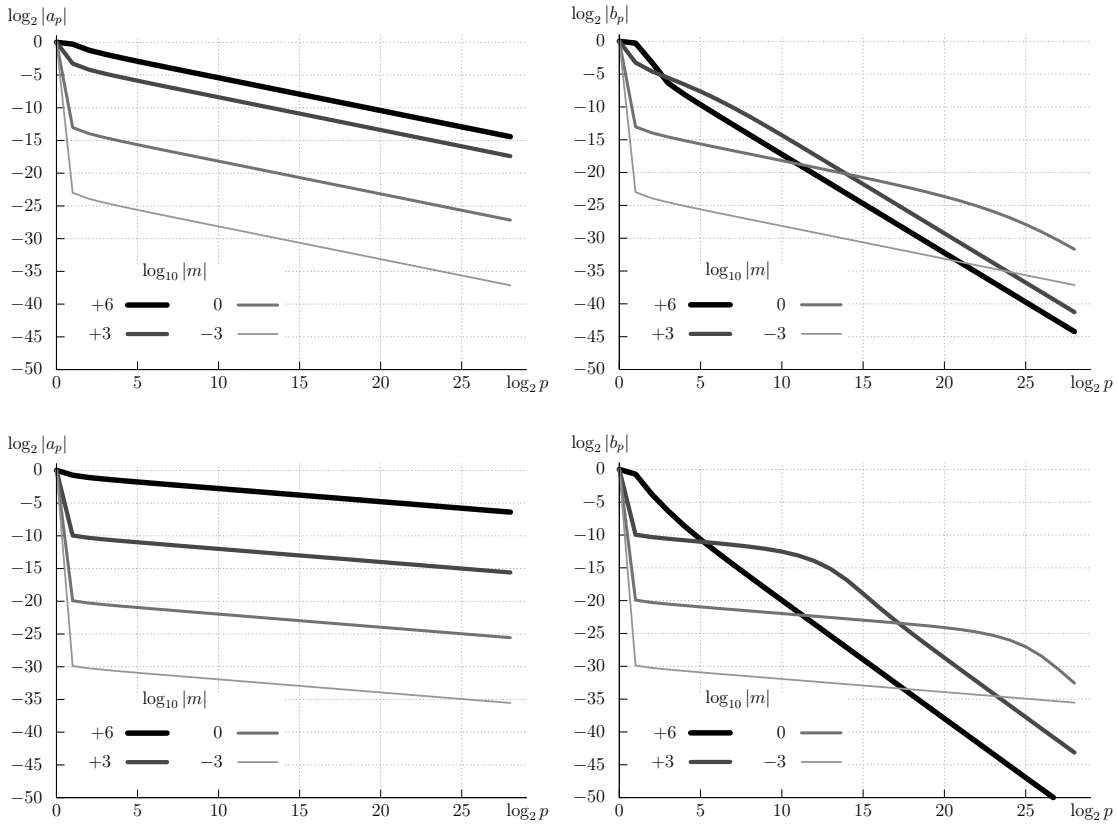

Figure 2: Decay profiles of triangular Toeplitz matrix (3) (left) and its inverse (right) for $n = 2^{28}$, $h = T/n$, $T = 10$ and $\alpha = 0.5$ (top) and $\alpha = 0.8$ (bottom) and for different mass m.

4. Inversion of triangular Toeplitz matrices using QTT approximation

4.1. Tensor train and quantized tensor train formats

A *tensor* is an array with d indices (or *modes*)

$$\mathbf{A} = [a(k_1, \ldots, k_d)], \qquad k_p = 0, \ldots, n_p - 1, \quad p = 1, \ldots, d.$$

The tensor train (TT) format [36, 17] for the tensor $\mathbf{A}$ reads[3]

$$a(k_1, k_2, \ldots, k_d) = A^{(1)}_{k_1} A^{(2)}_{k_2} \ldots A^{(d)}_{k_d}, \tag{14}$$

where each $A^{(p)}_{k_p}$ is an $r_{p-1} \times r_p$ matrix. Usually the *border conditions* $r_0 = r_d = 1$ are imposed to make every entry $a(k_1, \ldots, k_d)$ a scalar. However, larger r_0 and r_d can be considered and every entry of a tensor $\mathbf{A} = [a(k_1, \ldots, k_d)]$ becomes an $r_0 \times r_d$ matrix. Values $r_0, \ldots, r_{d-1}$ are referred to as *TT–ranks* and characterize the *separation properties* of the tensor $\mathbf{A}$. Three-dimensional arrays $A^{(p)} = [A^{(p)}_{k_p}]$ are referred to as *TT–cores*.

To apply the TT compression to low dimensional data, the idea of *quantization* was proposed [37, 18]. We will explain the idea for a one-dimensional vector $a =$

[3]We will often write the equations in elementwise form, which assumes that all indices run through all possible values.

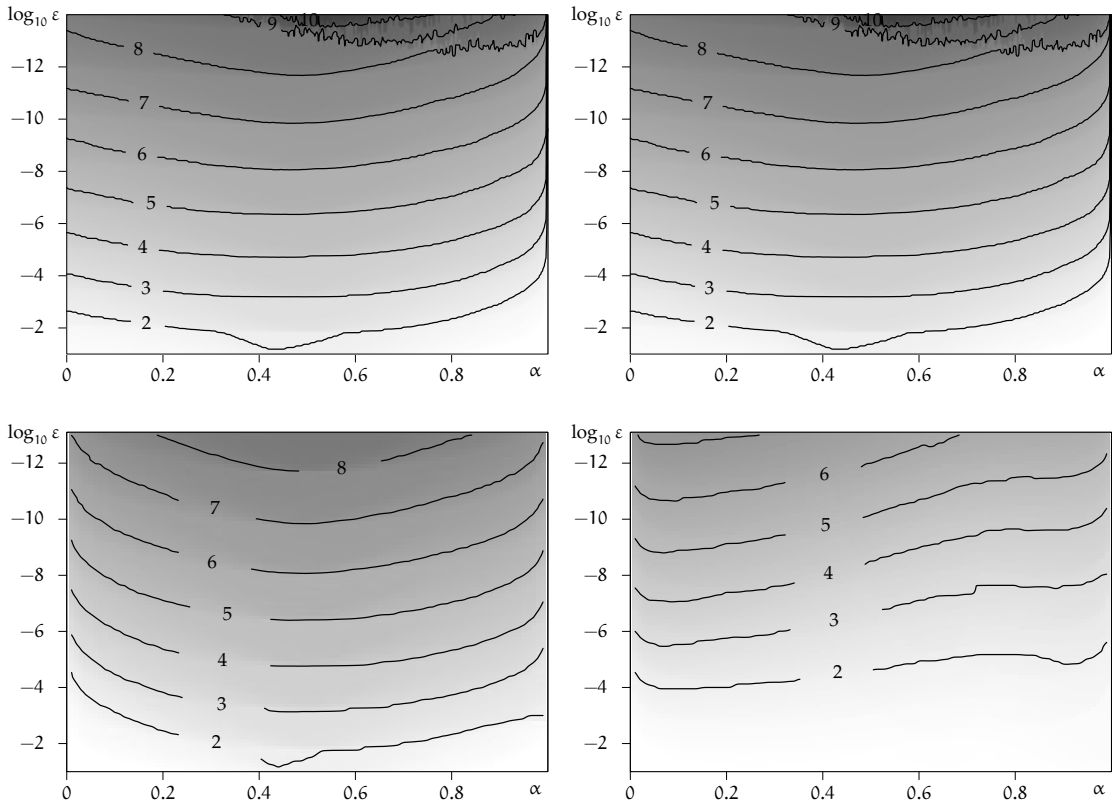

Figure 3: Effective QTT rank of vector $[k^{\alpha-1}]$ (top left), vector $[(k-1)^{\alpha+1} - 2k^{\alpha+1} + (k+1)^{\alpha+1}]$ (top right), first column of matrix (3) (bottom left) and its inverse (bottom right) w.r.t. parameter α and relative approximation accuracy ε in Frobenius norm. Problem size $n = 2^{28}$, maximum time $T = 10$, mass $m = -10^{+6}$

$[a(k)]_{k=0}^{n-1}$, restricting the discussion to $n = 2^d$. Define the binary notation of index k as follows

$$k = \overline{k_1 \dots k_d} \stackrel{\text{def}}{=} \sum_{p=1}^{d} k_p 2^{p-1}, \qquad k_p = 0, 1. \tag{15}$$

The isomorphic mapping $k \leftrightarrow (k_1, \dots k_d)$ allows us to *reshape* a vector $a = [a(k)]$ into the d–tensor $\dot{\mathbf{A}} = [\dot{a}(k_1, \dots, k_d)]$. The TT format (14) for the latter is called the *QTT format* and reads

$$a(k) = a(\overline{k_1 \dots k_d}) = \dot{a}(k_1, \dots, k_d) = A_{k_1}^{(1)} \dots A_{k_d}^{(d)}. \tag{16}$$

This idea appears in [37] in the context of matrix approximation. In [18] the TT format applied after the quantization of indices was called the *QTT format* and applied to a class of functions discretized on uniform grids, revealing the impressive approximation properties. In particular, it was proven that the QTT–ranks of $\exp x$, $\sin x$, $\cos x$, x^p are uniformly bounded w.r.t. the grid size. For the functions $e^{-\alpha x^2}$, x^α, $\frac{\sin x}{x}$, $\frac{1}{x}$, etc., similar properties were found experimentally.

The QTT separation function of the function $x^{\alpha-1}$ discretized on a uniform grid, is particularly important for us, because it motivates the use of the QTT approximation to develop the superfast algorithms for the solution of fractional differential equations. In the numerical experiment we found out that the QTT ranks are very moderate for all $0 < \alpha < 1$ and for accuracy up to the machine

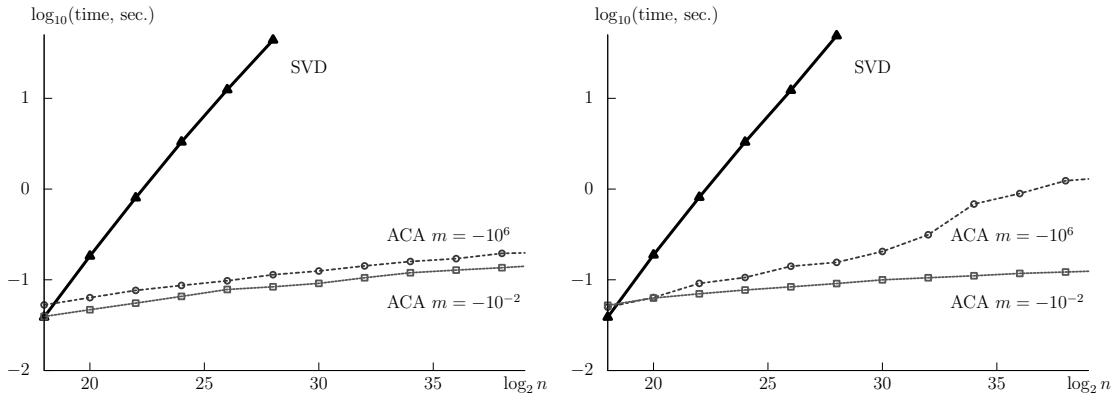

Figure 4: Runtimes of TT–SVD and TT–ACA algorithms for the approximation of matrix (3) in the QTT format w.r.t. size n. (left) $\alpha = 0.1$, (right) $\alpha = 0.9$.

threshold. The same holds for the first column of matrix (3) as well as for its inverse. On Fig. 3 we show the effective (average) QTT rank w.r.t. ε and α. We note that the effective rank does not overcome 10, even for very large grids up to $n = 2^{28}$.

To construct a superfast algorithms in the QTT format we first have to compress the data to this format using the algorithm with the sublinear complexity. The original TT–SVD algorithm proposed in [17] requires all elements of tensor and therefore does not suit for this purpose. To compress matrix (3) to QTT format, we apply cross interpolation algorithm TT–ACA proposed in [38]. This method computes the approximation using only a few elements of the original array, and does not require all elements to be computed. The comparison of runtimes of TT–SVD and TT–ACA algorthms is given on Fig. 4. The time that is required to choose the good subset of elements for the interpolation depends on the structure of data, which is defined by parameters α and m. It is easy to see that the behaviour of data is less regular for the large α and mass, which leads to larger runtimes of TT–ACA. Nevertheless, we clearly see that TT–ACA outperforms TT–SVD for all examples and has sublinear complexity w.r.t. problem size n.

4.2. Fourier transform and convolution in QTT format

Inversion algorithms for triangular Toeplitz matrices $A \in \mathcal{T}_n$ recalled in Sec. 3 are based on two main operations: Fourier transform and discrete convolution. The radix-2 reccurent relation which was known to Gauss [21] and lays behind the famous Cooley-Tuckey FFT algorithm [22] perfectly matches the multilevel structure of QTT format, resulting in the QTT–FFT algorithm [16]. For a vector of size $n = 2^d$ given approximately in QTT format (16), the QTT–FFT computes the Fourier transform with complexity $\mathcal{O}(d^2R^3)$, where R is the maximum QTT rank of the input vector, Fourier image, and all intermediate vectors of the algorithm.

The discrete convolution, i.e., multiplication by Toeplitz matrix, can be performed by three Fourier transforms with complexity $\mathcal{O}(d^2R^3)$, where R bounds the QTT ranks of both vectors to convolve as well as their Fourier images. As shown in [39], the convolution $c = a \star b$ of two vectors with QTT ranks bounded by r_a and r_b, can be written in QTT form with QTT ranks bounded by $2r_ar_b$. This representation has large QTT ranks, which can be reduced to the value bounded by $r_c \leq 2r_ar_b$

using some TT–truncation algorithm. We can use SVD–based algorithm proposed in [17] or iterative DMRG–type approach proposed in [40], resulting in convolution algorithms with $\mathcal{O}(dr_a^3 r_b^3)$ and $\mathcal{O}(d(r_a + r_b + r_c)r_a r_b r_c)$ complexity, respectively. If $r_a \approx r_b \approx r_c \approx R$, the QTT–FFT and DMRG–based convolution algorithms have complexity $\mathcal{O}(d^2 R^3)$ and $\mathcal{O}(dR^4)$, respectively. Therefore, we can not say in general which approach is better, even in the simplest case of almost equal QTT ranks. This will be established in numerical experiments.

4.3. Shifts of vectors in QTT format

The convolution algorithm proposed in [39] is based on the remarkable property of shift matrices $L \in \mathcal{T}_{2^d}$ and $U = L^T$, where the first column of L is $l = (0, 1, 0, 0, \ldots, 0)^T$. It is shown in [39] that all matrices L^p, $p = 1, \ldots, 2^d - 1$ have all QTT ranks two. Hence, if a vector a has QTT ranks $r_1, \ldots, r_{d-1}$, then the right shifted vector $b = L^p a$ for all p has QTT ranks not larger than $2r_1, \ldots, 2r_{d-1}$. The same holds for left shifts $c = U^p a$.

In the following theorem we improve this result for vectors, shifted by one element.

Theorem 4.1. *Let $a = [a(k)]_{k=0}^{2^d-1}$ has the QTT representation* (16)*, then the vector*

$$b = \begin{bmatrix} x & a(0) & \ldots & a(2^d - 2) \end{bmatrix}^T$$

has the QTT representation $b(k) = b(\overline{k_1 \ldots k_d}) = B_{k_1}^{(1)} \ldots B_{k_d}^{(d)}$ with the following cores

$$\begin{aligned} B_0^{(1)} &= \begin{bmatrix} & 1 \end{bmatrix}, \quad B_0^{(p)} = \begin{bmatrix} A_0^{(p)} & \\ & 1 \end{bmatrix}, \quad B_0^{(d)} = \begin{bmatrix} A_0^{(d)} \\ x \end{bmatrix}, \\ B_1^{(1)} &= \begin{bmatrix} A_0^{(1)} & \end{bmatrix}, \quad B_1^{(p)} = \begin{bmatrix} A_1^{(p)} & \\ b_p & \end{bmatrix}, \quad B_1^{(d)} = \begin{bmatrix} A_1^{(d)} \\ b_d \end{bmatrix}, \end{aligned} \tag{17}$$

where $p = 2, \ldots, d-1$ and $b_q = A_1^{(1)} \ldots A_1^{(q-1)} A_0^{(q)}$ for $q = 2, \ldots, d$. Similarly, the vector

$$c = \begin{bmatrix} a(1) & \ldots & a(2^d - 1) & y \end{bmatrix}^T$$

has the QTT representation $c(k) = c(\overline{k_1 \ldots k_d}) = C_{k_1}^{(1)} \ldots C_{k_d}^{(d)}$ with the following cores

$$\begin{aligned} C_0^{(1)} &= \begin{bmatrix} A_1^{(1)} & \end{bmatrix}, \quad C_0^{(p)} = \begin{bmatrix} A_0^{(p)} & \\ c_p & \end{bmatrix}, \quad C_0^{(d)} = \begin{bmatrix} A_0^{(d)} \\ c_d \end{bmatrix}, \\ C_1^{(1)} &= \begin{bmatrix} & 1 \end{bmatrix}, \quad C_1^{(p)} = \begin{bmatrix} A_1^{(p)} & \\ & 1 \end{bmatrix}, \quad C_1^{(d)} = \begin{bmatrix} A_1^{(d)} \\ y \end{bmatrix}, \end{aligned} \tag{18}$$

where $p = 2, \ldots, d-1$ and $c_q = A_0^{(1)} \ldots A_0^{(q-1)} A_1^{(q)}$ for $q = 2, \ldots, d$.

Proof. We check (17) straightforwardly. For $k = 0$ it holds

$$b(0) = B_0^{(1)} \ldots B_0^{(d)} = \begin{bmatrix} & 1 \end{bmatrix} \ldots \begin{bmatrix} A_0^{(p)} & \\ & 1 \end{bmatrix} \ldots \begin{bmatrix} A_0^{(d)} \\ x \end{bmatrix} = x$$

For $k = \overline{k_1 k_2 \dots k_d}$ with $k_1 = 1$ it holds

$$b(k) = b(\overline{1 k_2 \dots k_d}) = B_1^{(1)} B_{k_2}^{(2)} \dots B_{k_d}^{(d)} = \begin{bmatrix} A_0^{(1)} & \end{bmatrix} \begin{bmatrix} A_{k_2}^{(2)} & \\ * & * \end{bmatrix} \dots \begin{bmatrix} A_{k_d}^{(d)} \\ * \end{bmatrix}$$
$$= A_0^{(1)} A_{k_2}^{(2)} \dots A_{k_d}^{(d)} = a(\overline{0 k_2 \dots k_d}) = a(k-1),$$

where "$*$" denotes arbitrarily zero or non-zero element. For $k = \overline{k_1 k_2 k_3 \dots k_d}$ with $k_1 = 0$ and $k_2 = 1$ it holds

$$b(k) = b(\overline{01 k_3 \dots k_d}) = B_0^{(1)} B_1^{(2)} B_{k_3}^{(3)} \dots B_{k_d}^{(d)} = \begin{bmatrix} & 1 \end{bmatrix} \begin{bmatrix} A_1^{(2)} & \\ b_2 & \end{bmatrix} \begin{bmatrix} A_{k_3}^{(3)} & \\ * & * \end{bmatrix} \dots \begin{bmatrix} A_{k_d}^{(d)} \\ * \end{bmatrix}$$
$$= b_2 A_{k_3}^{(3)} \dots A_{k_d}^{(d)} = A_1^{(1)} A_0^{(2)} A_{k_3}^{(3)} \dots A_{k_d}^{(d)} = a(\overline{10 k_3 \dots k_d}) = a(k-1).$$

Finally, for $k = \overline{k_1 k_2 k_3 \dots k_d}$ with $k_1 = \dots = k_{p-1} = 0$ and $k_p = 1$ it holds

$$b(k) = b(\overline{\underbrace{0 \dots 0}_{p-1 \text{ zeros}} 1 k_{p+1} \dots k_d}) = B_0^{(1)} B_0^{(2)} \dots B_0^{(p-1)} B_1^{(p)} B_{k_{p+1}}^{(p+1)} \dots B_{k_d}^{(d)}$$
$$= \begin{bmatrix} & 1 \end{bmatrix} \begin{bmatrix} A_0^{(2)} & \\ & 1 \end{bmatrix} \dots \begin{bmatrix} A_0^{(p-1)} & \\ & 1 \end{bmatrix} \begin{bmatrix} A_1^{(p)} & \\ b_p & \end{bmatrix} \begin{bmatrix} A_{k_{p+1}}^{(p+1)} & \\ * & * \end{bmatrix} \dots \dots \begin{bmatrix} A_{k_d}^{(d)} \\ * \end{bmatrix}$$
$$= b_p A_{k_{p+1}}^{(p+1)} \dots A_{k_d}^{(d)} = A_1^{(1)} \dots A_1^{(p-1)} A_0^{(p)} A_{k_{p+1}}^{(p+1)} \dots A_{k_d}^{(d)}$$
$$= a(\overline{\underbrace{1 \dots 1}_{p-1 \text{ ones}} 0 k_{p+1} \dots k_d}) = a(k-1).$$

Equation (18) is verified in the same way.

4.4. Divide and conquer algorithm in QTT format

We are now ready to present the version of divide and conquer algorithm which operates with data given approximately in QTT format. Let $n = 2^d$ and consider $A \in \mathcal{T}_n$ defined by the first column $a(k)$ which is represented in the QTT format (16). As previously, let A_t denote $2^t \times 2^t$ leading submatrix of A. For small d_0 we can invert A_{d_0} using standard divide and conquer method and approximate the first column of $A_{d_0}^{-1}$ in QTT format using SVD-based algorithm [17].

Now suppose that for some t the first column of A_t^{-1} is computed in QTT format, and we have to compute the QTT approximation of the first column of A_{t+1}^{-1}, using the recursion (11). It is necessary to describe the Toeplitz matrix C which lies in the lower part of A_{t+1}. The first column of C is $c_+ = [a(2^t), a(2^t+1), \dots, a(2^{t+1}-1)]^T$ and has the following QTT representation

$$c_+(\overline{k_1 \dots k_t}) = a(\overline{k_1 \dots k_t} + 2^t) = A_{k_1}^{(1)} A_{k_2}^{(2)} \dots A_{k_t}^{(t)} A_1^{(t+1)} A_0^{(t+2)} \dots A_0^{(d)}. \qquad (19)$$

The first row of C is $c_- = [a(2^t), a(2^t-1), \dots, a(1)]^T$. To construct the QTT representation for c_-, first write the QTT representation for $a = [a(0), \dots, a(2^t-1)]^T$, which is

$$a(\overline{k_1 \dots k_t}) = A_{k_1}^{(1)} A_{k_2}^{(2)} \dots A_{k_t}^{(t)} A_0^{(t+1)} A_0^{(t+2)} \dots A_0^{(d)}.$$

Then apply the 'pull' operation and construct QTT format for $a' = [a(1), \dots, a(2^t)]^T$ as follows

$$a'(\overline{k_1 \dots k_t}) = C_{k_1}^{(1)} C_{k_2}^{(2)} \dots C_{k_t}^{(t)} A_0^{(t+1)} A_0^{(t+2)} \dots A_0^{(d)},$$

Algorithm 1 Divide and conquer in QTT format

Require: $A \in \mathcal{T}_n$, $n = 2^d$, given by vector $[a(k)]_{k=0}^{n-1}$ in QTT format (16)
Ensure: $B = A^{-1} \in \mathcal{T}_n$ given in QTT format

1: For small d_0, compute the first column of $2^{d_0} \times 2^{d_0}$ leading submatrix A_{d_0}. Compute $B_{d_0} = A_{d_0}^{-1}$ by (10) and approximate in in QTT format by TT–SVD algorithm [17].
2: **for** $t = d_0, \ldots, d-1$ **do**
3: Compute the first row and column of matrix C_t in (11) in QTT format by (19) and (20).
4: Compute the first column of $B_t C_t B_t$ by two convolutions in QTT format, see Sec. 4.2.
5: Combine the first column of B_t and first column of $B_t C_t B_t$ given in QTT format as follows

$$b(\overline{k_1 \ldots k_t}) = B_{k_1}^{(1)} \ldots B_{k_1}^{(t)}, \qquad g(\overline{k_1 \ldots k_t}) = G_{k_1}^{(1)} \ldots G_{k_1}^{(t)},$$

to the single vector b' in QTT format, which is defined as follows

$$b'(\overline{k_1 \ldots k_t k_{t+1}}) = \begin{bmatrix} B_{k_1}^{(1)} & G_{k_1}^{(1)} \end{bmatrix} \begin{bmatrix} B_{k_2}^{(2)} & \\ & G_{k_2}^{(2)} \end{bmatrix} \cdots \begin{bmatrix} B_{k_t}^{(t)} & \\ & G_{k_t}^{(t)} \end{bmatrix} \begin{bmatrix} 1-k_{t+1} \\ k_{t+1} \end{bmatrix}$$

6: Apply TT–truncate algorithm to b' to reduce the ranks of QTT representation.
7: **end for**

where TT–cores $C_{k_q}^{(q)}$ are defined by (18). Finally, revert the ordering of elements in the vector a' to obtain the QTT format for c_- as follows

$$c_-(\overline{k_1 \ldots k_t}) = C_{1-k_1}^{(1)} C_{1-k_2}^{(2)} \ldots C_{1-k_t}^{(t)} A_0^{(t+1)} A_0^{(t+2)} \ldots A_0^{(d)}. \tag{20}$$

We summarize the above steps in Alg. 1. Note that the workhorse of divide and conquer method is the discrete convolution in QTT format, which can be performed by two different methods. This results in two variants of algorithms with different performance, which will be studied in numerical experiments.

4.5. Modified Bini's algorithm in QTT format

The implementation of (12) in the QTT format is very straightforward. It is enough to mention that the QTT format of vector $[\varepsilon^j]_{j=0}^{n-1}$, $n = 2^d$, has QTT–ranks one (see [18]), since

$$\varepsilon^j = \varepsilon^{\overline{j_1 j_2 \cdots j_d}} = \varepsilon^{j_1} \varepsilon^{2 j_2} \ldots \varepsilon^{2^{d-1} j_d}.$$

Therefore, multiplication of a vector in QTT format by diagonal matrix D_ε requires only the appropriate scaling of TT–cores. By Alg. 2 we present the QTT version of the modified Bini's algorithm [28, Alg. 2]. The algorithm includes two Fourier transforms in the QTT format which can not be substituted by discrete convolution. Note that this algorithm contains two approximation errors:

- The first comes form original approximation of triangular Toeplitz matrix A by the diagonally scaled circulant matrix A_ε. The accuracy of this approximation is governed by parameter ε. According to the numerical tests made by the

Algorithm 2 Modified Bini's method in QTT format

Require: $A \in \mathcal{T}_n$, $n = 2^d$, given by vector $[a(k)]_{k=0}^{n-1}$ in QTT format (16)
Ensure: $B_\varepsilon = A_\varepsilon^{-1} \approx A^{-1} \in \mathcal{T}_n$ given in QTT format

1: Choose $0 < \varepsilon < 1$ and let $\hat{a}(k) = \varepsilon^k a(k)$ for $k = 0, \ldots, n-1$ and $\hat{a}(k) = 0$ for $k = n, \ldots, 2n-1$. The QTT representation of $\hat{a}$ is the following

$$\hat{a}(k) = \hat{a}(\overline{k_1 \ldots k_d k_{d+1}}) = \hat{A}^{(1)}_{k_1} \ldots \hat{A}^{(d)}_{k_d}(1-k_{d+1}), \qquad \hat{A}^{(p)}_{k_p} = \varepsilon^{2^{p-1}k_p} A^{(p)}_{k_p}, \quad p = 1, \ldots, d.$$

2: Apply QTT–FFT [16] to compute the size–$2n$ Fourier transform $\lambda = \sqrt{2n} F \hat{a}$.
3: Apply Newton iteration (13) to compute $c = \lambda^{-1}$ in the QTT format. Each iteration step includes the pointwise (Hadamard) multiplication of vectors in QTT format and TT–truncation to reduce the QTT–ranks.
4: Apply QTT–FFT again to compute the size–$2n$ Fourier transform $\hat{b} = F^* c/\sqrt{2n}$ in the QTT format

$$\hat{b}(k) = \hat{b}(\overline{k_1 \ldots k_d k_{d+1}}) = \hat{B}^{(1)}_{k_1} \ldots \hat{B}^{(d)}_{k_d} \hat{B}^{(d+1)}_{k_{d+1}}.$$

5: The QTT representation of the first column of B_ε is the following

$$b_\varepsilon(k) = b_\varepsilon(\overline{k_1 \ldots k_d}) = B^{(1)}_{k_1} \ldots B^{(d)}_{k_d} \hat{B}^{(d+1)}_0, \qquad B^{(p)}_{k_p} = \varepsilon^{-2^{p-1}k_p} \hat{B}^{(p)}_{k_p}, \quad p = 1, \ldots, d.$$

authors of [28], the good choice for Bini's and modified Bini's methods are $\varepsilon^n = 0.5 \times 10^{-8}$ and $\varepsilon^n = 10^{-5}$, respectively.

- The second error comes from TT–truncation algorithm applied in the QTT–FFT algorithm and on each step of Newton iteration. The threshold parameter of TT–truncation should be usually smaller than ε^n in order to maintain the accuracy of the result after diagonal scaling.

5. Numerical experiments

5.1. Timings of inversion algorithms

On Fig. 5 we show the runtime of inversion algorithms for triangular Toeplitz matrices in full and in the QTT format w.r.t. problem size and for different parameters α, m. Standard inversion algorithms have the $\mathcal{O}(n \log n)$ complexity which depends only on problem size. Quite contrarily, the complexity and runtime of QTT algorithms depend on QTT–ranks of input and intermediate vectors, which are sensitive to the fractional order α, mass m and step size h. They also depend crucially on the method used to compute the discrete convolution in the QTT format. We can note that the divide and conquer algorithm 1 which uses QTT–conv algorithm [39] is always significantly faster than the same method which uses QTT–FFT algorithm [16] to compute the convolution. However, QTT–FFT works well in modified Bini's algorithm 2, which appears to be the fastest method when mass is small in modulus. For large mass the divide and conquer algorithm 1 with QTT–conv is preferable to the modified Bini's algorithm 2. For mass $m \sim -1$ these methods have the same asymptotical complexity.

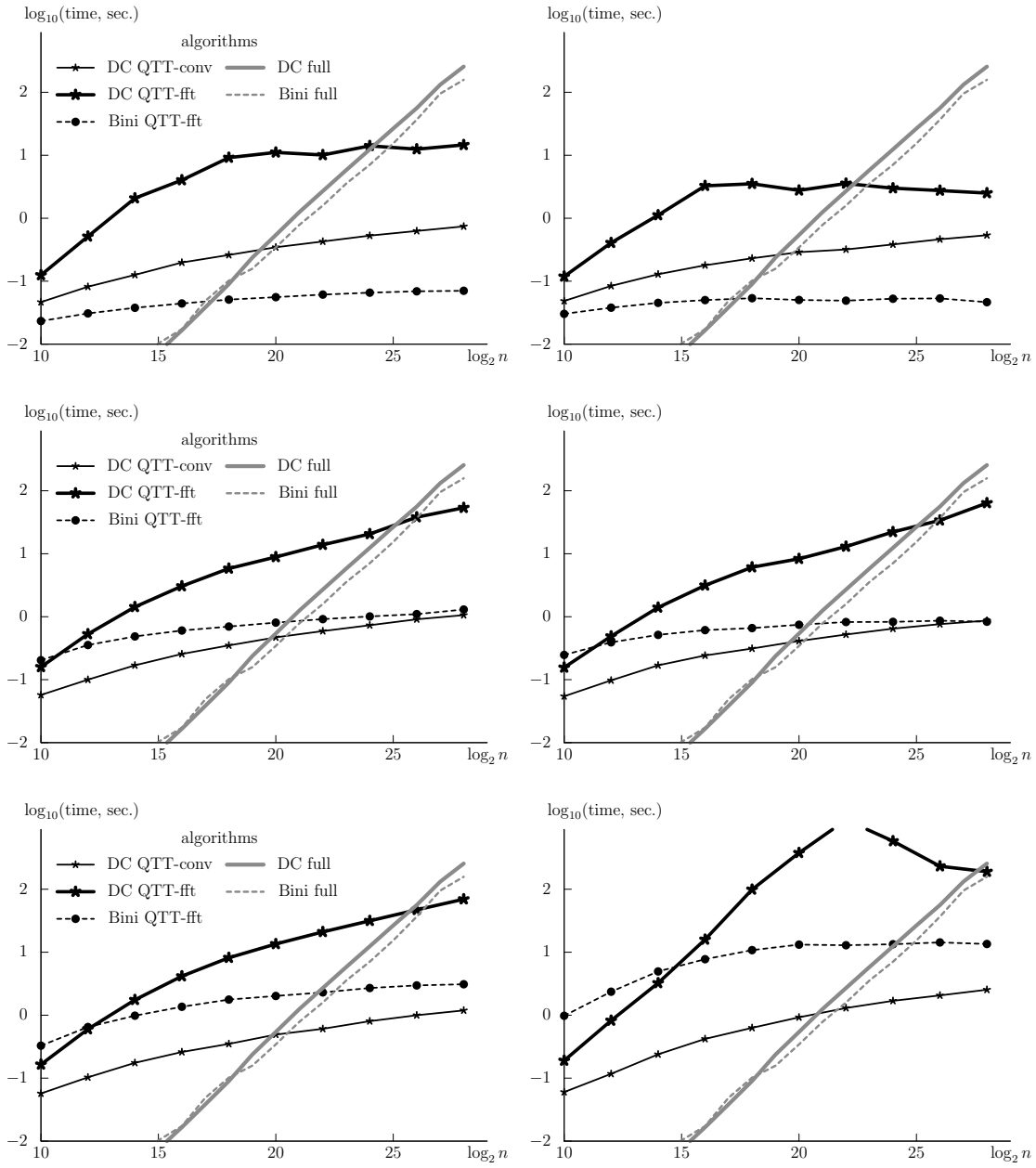

Figure 5: Runtimes of divide and conquer algorithm (solid lines) and modified Bini's algorithm (dashed lines) for the inversion of triangular Toeplitz matrix (3) in full and in the QTT formats (grey and black lines, respectively) w.r.t. problem size n and step size $h = T/n$. Fixed maximum time $T = 10$, fractional order $\alpha = 0.2$ (left) and $\alpha = 0.8$ (right), mass $m = -10^{-5}$ (top), $m = -10^{0}$ (middle), $m = -10^{5}$ (bottom).

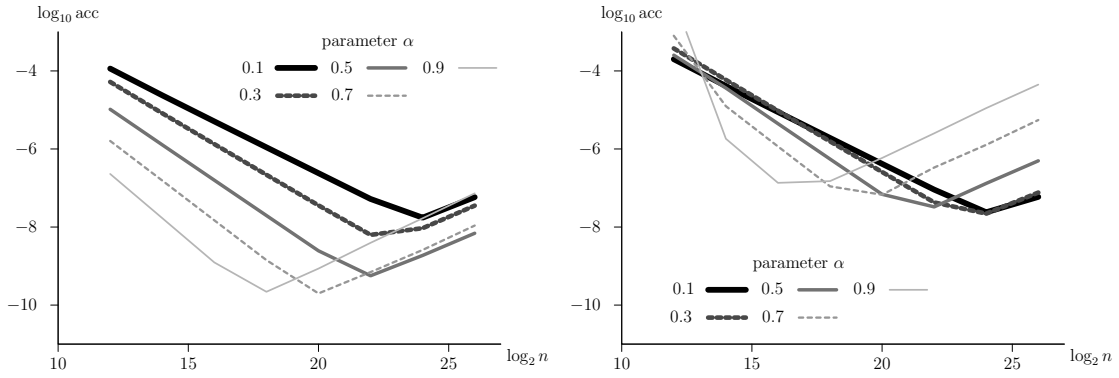

Figure 6: Accuracy of the solution of the test problem (21) in the relative Frobenius norm w.r.t. problem size n and for different fractional parameters α. Fixed maximum time $T = 10$ (left) and $T = 10^5$ (right). Mass $m = -1$.

From Fig. 5 it can be easily seen that the QTT algorithms are asymptotically faster than the algorithms in full format. For practical computations it is very important at which size n there is a *crossover point*, i.e., the minimum value of n for which the QTT algorithms are actually faster than the algorithms in full format. Numerical experiments show that for a wide range of parameters α and m the crossover point between full and QTT divide and conquer methods is $\log_2 n \simeq 20$. This value is about the same as the crossover point between FFT and QTT–FFT algorithm applied to signals with sparse Fourier image [16]. The crossover point between full and QTT versions of the modified Bini's algorithm depends on m and α and can be even smaller, e.g. $\log_2 n \simeq 17$ for m small in modulus.

5.2. Accuracy test for constant forcing

We consider a simple problem for which the analytical solution is available, namely the one with constant forcing term.

$$D_*^\alpha y(t) = my(t) + \lambda, \qquad y(0) = y_0. \tag{21}$$

The analytical solution is written in the following form

$$y(t) = y_0 E_\alpha\left(mt^\alpha\right) + \frac{\lambda}{m} E_\alpha\left(mt^\alpha\right) - \frac{\lambda}{m}, \tag{22}$$

where E_α is the Mittag–Lefler function [41, 42], which can be expressed and computed by certain (sometimes slow-converging) series.

The accuracy verification results are shown on Fig. 6. We see that as the problem size grows, the accuracy improves until certain point and then the error start growing. This is explained by the machine threshold errors amplified by the condition number of the matrix A from (3) which is unbounded as n grows to the infinity.

5.3. Accuracy of the Laplace transform

Consider the following test equation

$$D_*^\alpha y(t) = my(t) + t^{\frac{3}{4}}, \qquad y(0) = 1. \tag{23}$$

Since the forcing term $f(t) = t^{3/4}$ does not have a short Taylor series representation, this problem could be difficult for methods based on it. Unlike the previous example,

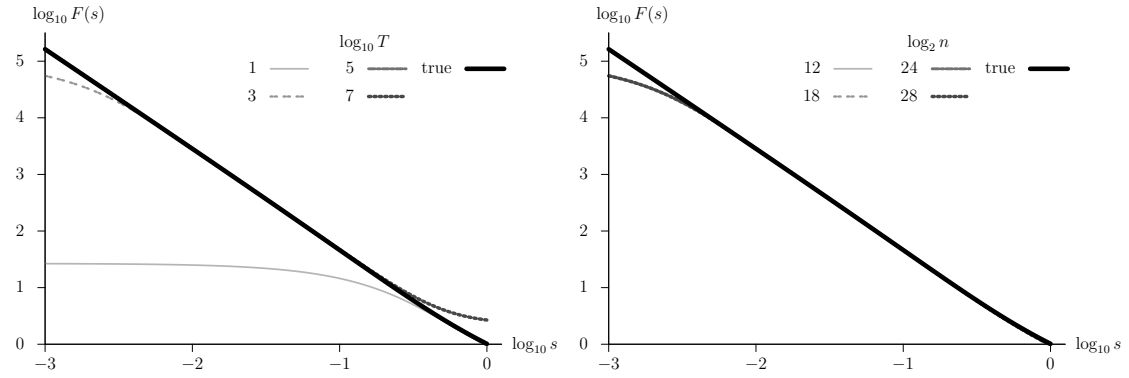

Figure 7: The Laplace transform of the solution (25) and its discrete approximations. Fractional parameter $\alpha = 0.8$, mass $m = -1$. Left: fixed number of grid points $n = 2^{22}$, different maximum time T. Right: fixed maximum time $T = 10^3$, different n.

the analytical solution in space domain is not available. Instead we can solve the problem using the Laplace transform.[4] The Laplace transform of a function $f(t)$ with the appropriate speed of decay is defined by

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t) dt. \tag{24}$$

The Laplace transform of a convolution is the product of Laplace transforms,

$$(f \star g)(t) = \int_0^t f(\tau) g(t-\tau) d\tau \qquad \Leftrightarrow \qquad \mathcal{L}\{f \star g\}(s) = \mathcal{L}\{f\}(s)\, \mathcal{L}\{g\}(s).$$

This allows to simplify the equation (23) and find the Laplace transform of the solution,

$$Y(s) = \frac{1}{s^{1-\alpha}(s\alpha - m)} + \frac{\Gamma(1.75)}{s^{1.75}(s^\alpha - m)}. \tag{25}$$

The inverse Laplace transform $y(t) = \mathcal{L}^{-1}\{Y(s)\}$ is given by the complex contour integral and is difficult for numerical computation. However, we can easily compute the Laplace transform of the discrete solution $\mathcal{L}\,\tilde{y} = \tilde{Y}$ in points $\{s_k\}$ using a rectangle quadrature rule,

$$\tilde{Y}(s_k) \approx \tilde{Y}_k = h \sum_{j=0}^{n} e^{-t_j s_k} \tilde{y}(t_j), \qquad t_j = jh. \tag{26}$$

Then we compare $Y(s_k)$ and $\tilde{Y}_k$ to establish the accuracy of the discrete solution $\tilde{y}(t_j) = y_j$. It is easy to see that equation (26) contains three sources of errors, i.e. the ones of the discrete solution, of the quadrature rule and of the truncation of indefinite integral (24) to the finite interval $[0 : T]$, $T = nh$. To compute $\tilde{Y}(s)$ accurately for small s we should take $T > s^{-1} \log \varepsilon^{-1}$, where ε is a machine precision error. To keep the quadrature rule error small, we should also use grids with small time step h. This is shown on Fig. 7, where the exact Laplace transform (25) is compared with its discrete approximations for different T and n. These factors motivate the use of

[4]History of the Laplace transform and other essential details can be found in, eg. [43].

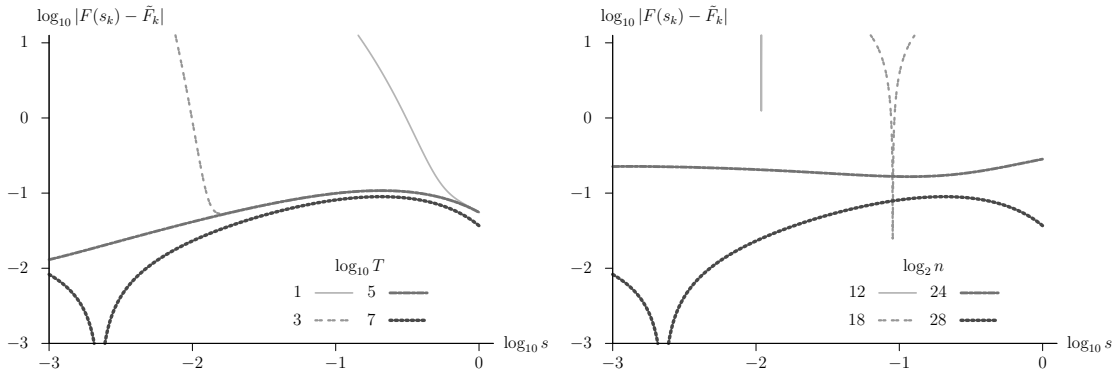

Figure 8: Accuracy of the Laplace transform (25) given by discrete approximation (26). Fractional parameter $\alpha = 0.8$, mass $m = -1$. Left: fixed number of grid points $n = 2^{28}$, different maximum time T. Right: fixed maximum time $T = 10^7$, different n.

very large grid size n and hence the QTT approach. It should be noted that the discrete Laplace transform (26) is computed perfectly in the QTT format since the QTT–ranks of the exponent are all ones.

Finally, on Fig. 8 we show the accuracy of the Laplace transform of the solution (25) for $10^{-3} \leq s \leq 10^0$ and for different T and n. It is clear that large problem size is essential for the accurate representation of the solution in the Laplace transform space.

6. Conclusions and future work

We present the new family of algorithms for the solution of linear fractional ODEs. Our approach develops the framework of matrix algorithms for fractional calculus [9] by embedding the QTT tensor decomposition inside matrices, as proposed in [37]. The proposed algorithms work on matrix level and can be formally applied for the inversion of any triangular Toeplitz matrix, as well as the one obtained by discretisation of a linear fractional calculus problem. The workhorse of the inversion algorithms is the discrete convolution and/or Fourier transform of vectors given/approximated in the compressed QTT form. The success of the proposed algorithms, however, is based on the representability of the initial matrix and intermediate vectors arising in computations in the QTT format with a modest accuracy.

As the motivating example we consider a simple linear fractional differential equation which reduces to the weakly singular convolutional Volterra equation with the Abel-type kernel. The QTT approximation method benefits from both the smoothness and decay of the Abel kernel, which results in efficient QTT–representation of problem matrix with the accuracy up to the machine precision. As shown by numerical experiments, the QTT–ranks of the intermediate vectors in the proposed algorithms remain bounded or grow slowly with the problem size. As the result, our algorithms of the inversion of triangular Toeplitz matrices demonstrate sublinear $o(n)$ complexity, which falls down to the complexity $\mathcal{O}(\log^2 n)$ of the superfast Fourier transform in certain cases. For our implementation the crossover point with the standard algorithms based on the FFTW library for the considered experiments is $17 \lesssim \log_2 n \lesssim 21$, i.e., the developed methods give not only the asymptotical benefit, but also a practical speedup for the problems of moderate size.

The proposed approach opens a new class of algorithms for the fractional calculus, i.e., methods of sublinear complexity. The developed techniques can be applied to the fractional equations with several differential operators of different order. They also can be generalised to fractional PDEs in two and more dimensions and to the nonlinear fractional problem. This would be the topic of further work, which will be reported elsewhere.

[1] K. Diethelm, An algorithm for the numerical solution of differential equations of fractional order, Electronic transactions on numerical analysis 5 (1997) 1–6.

[2] M. Caputo, F. Mainardi, Linear models of dissipation in anelastic solids, Rivista del Nuovo Cimento 1 (1971) 161–198. doi:10.1007/BF02820620.

[3] A. Freed, K. Diethelm, Fractional calculus in biomechanics: a 3D viscoelastic model using regularized fractional-derivative kernels with application to the human calcaneal fat pad, Biomech. Model. Mechanobiol. 5 (2006) 203–215. doi:10.1007/s10237-005-0011-0.

[4] R. L. Bagley, P. J. Torvik, On the appearance of fractional derivatives in the behavior of real materials, J. Appl. Mech. 51 (1984) 294–298.

[5] K. Diethelm, The analysis of fractional differential equations: an application-oriented exposition using differential operators of Caputo type, Lecture Notes in Mathematics, Springer, 2004.

[6] I. Podlubny, Fractional Differential Equations, Academic Press, 1999.

[7] P. Linz, Analytical and numerical methods for Volterra equations, SIAM, Philadelphia, 1985.

[8] R. Gorenflo, R. Rutman, On ultraslow and intermediate processes, in: P. Rusev, I. Dimovski, V. Kiryakova (Eds.), Transform methods and special functions, 1995, pp. 61–81.

[9] I. Podlubny, Matrix approach to discrete fractional calculus, Fractional calculus and applied analysis 3 (4) (2000) 359–386.

[10] L. Blanck, Stability of collocation for weakly singular Volterra equations, IMA J. Numer. Anal. 15 (1995) 357–375. doi:10.1093/imanum/15.3.357.

[11] L. Blanck, Stability results for collocation methods for Volterra integral equations, J. of Appl. Math. and Comp. 79 (1996) 267–288. doi:10.1016/0096-3003(95)00270-7.

[12] K. Diethelm, N. J. Ford, A. D. Freed, Detailed error analysis for a fractional Adams method, Numerical Algorithms 36 (1) (2004) 31–52.

[13] I. Podlubny, Numerical solution of ordinary fractional differential equations by the fractional difference method, in: Proceedings of the Second International Conference in Difference Equations, Gordon and Breach Scientific Publishers, 1997, pp. 507–515.

[14] N. J. Ford, A. C. Simpson, The numerical solution of fractional differential equations: speed versus accuracy, Numerical Algorithms (26) (2001) 333–346. doi:10.1023/A:1016601312158.

[15] K. Diethelm, N. Ford, A. Freed, Y. Luchko, Algorithms for the fractional calculus: a selection of numerical methods, Comput. Methods Appl. Mech. Engrg. 194 (2005) 743–773. doi:10.1016/j.cma.2004.06.006.

[16] S. V. Dolgov, B. N. Khoromskij, D. V. Savostyanov, Superfast Fourier transform using QTT approximation, J. Fourier Anal. Appl. 18 (5) (2012) 915–953. doi:10.1007/s00041-012-9227-4.

[17] I. V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput. 33 (5) (2011) 2295–2317. doi:10.1137/090752286.

[18] B. N. Khoromskij, $\mathcal{O}(d \log n)$–Quantics approximation of N–d tensors in high-dimensional numerical modeling, Constr. Appr. 34 (2) (2011) 257–280. doi:10.1007/s00365-011-9131-1.

[19] A. Ekert, R. Jozsa, Quantum algorithms: entanglement-enhanced information processing, Phil. Trans. R. Soc. Lond. 356 (1998) 1769–1782. doi:10.1098/rsta.1998.0248.

[20] D. V. Savostyanov, QTT-rank-one vectors with QTT-rank-one and full-rank Fourier images, Linear Algebra Appl. 436 (9) (2012) 3215–3224. doi:10.1016/j.laa.2011.11.008.

[21] C. F. Gauss, Nachlass: Theoria interpolationis methodo nova tractata, in: Werke, Vol. 3, Königliche Gesellschaft der Wissenschaften, Göttingem, 1866, pp. 265–330.

[22] J. W. Cooley, J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19 (1965) 297–301. doi:10.2307/2003354.

[23] G. Golub, C. Van Loan, Matrix computations, Johns Hopkins University Press, Baltimore, MD, 1996.

[24] B. J. Murphy, Acceleration of the inversion of triangular Toeplitz matrices and polynomial division, in: Computer Algebra in Scientific Computing, Vol. 6885/2011 of Lecture notes in computer science, Springer-Verlag, 2011, pp. 321–332. doi:10.1007/978-3-642-23568-9_25.

[25] M. Morf, Doubling algorithms for Toeplitz and related equations, in: Acoustics, Speech and Signal Proc., IEEE Conf. ICASSP, 1980, pp. 954–959. doi:10.1109/ICASSP.1980.1171074.

[26] D. Commenges, M. Monsion, Fast inversion of triangular Toeplitz matrices, IEEE Trans. on Automatic Control AC-29 (3) (1984) 250–251. doi:10.1109/TAC.1984.1103499.

[27] D. Bini, Parallel solution of certain Toeplitz linear systems, SIAM J. Comput. 13 (2) (1984) 268–279. doi:10.1137/0213019.

[28] F.-R. Lin, W.-K. Ching, M. K. Ng, Fast inversion of triangular Toeplitz matrices, Theor. Computer Sci. 315 (2-3) (2004) 511–523. doi:10.1016/j.tcs.2004.01.005.

[29] G. Schulz, Iterative Berechung der reziproken Matrix, ZAMM J. Appl. Math. and Mechanics 13 (1) (1933) 57–59. doi:10.1002/zamm.19330130111.

[30] A. Ben-Israel, D. Cohen, On iterative computation of generalized inverses and associated projections, SIAM J. Numer. Anal. 3 (3) (1966) 410–419. doi:10.1137/0703035.

[31] V. Pan, R. Schreiber, An improved Newton iteration for the generalized inverse of a matrix, with applications, SIAM J. Sci. Stat. Comput. 12 (5) (1991) 1109–1130. doi:10.1137/0912058.

[32] V. Y. Pan, Y. Rami, X. Wang, Structured matrices and Newton's iteration: unified approach, Linear Alg. Appl. 343-344 (2002) 233–265. doi:10.1016/S0024-3795(01)00336-6.

[33] W. Hackbusch, B. N. Khoromskij, E. E. Tyrtyshnikov, Approximate iterations for structured matrices, Numer. Mathematik 109 (3) (2008) 365–383. doi:10.1007/s00211-008-0143-0.

[34] V. Olshevsky, I. V. Oseledets, E. E. Tyrtyshnikov, Superfast inversion of two-level Toeplitz matrices using Newton iteration and tensor-displacement structure, Operator Theory: Advances and Applications 179 (2008) 229–240. doi:10.1007/978-3-7643-8539-2_14.

[35] I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, Linear algebra for tensor problems, Computing 85 (3) (2009) 169–188. doi:10.1007/s00607-009-0047-6.

[36] I. V. Oseledets, A new tensor decomposition, Doklady Math. 80 (1) (2009) 495–496. doi:10.1134/S1064562409040115.

[37] I. V. Oseledets, Approximation of $2^d \times 2^d$ matrices using tensor decomposition, SIAM J. Matrix Anal. Appl. 31 (4) (2010) 2130–2145. doi:10.1137/090757861.

[38] D. V. Savostyanov, I. V. Oseledets, Fast adaptive interpolation of multi-dimensional arrays in tensor train format, in: Proceedings of 7th International Workshop on Multidimensional Systems (nDS), IEEE, 2011. doi:10.1109/nDS.2011.6076873.

[39] V. Kazeev, B. Khoromskij, E. Tyrtyshnikov, Multilevel toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity, SIAM Journal on Scientific Computing 35 (3) (2013) A1511–A1536. doi:10.1137/110844830.

[40] I. V. Oseledets, DMRG approach to fast linear algebra in the TT–format, Comput. Meth. Appl. Math. 11 (3) (2011) 382–393. doi:10.2478/cmam-2011-0021.

[41] G. Mittag-Leffler, Une generalisation de l'integrale de Laplace-Abel, C. R. Acad. Sci. Paris (Ser. II) 136 (1903) 537–539.

[42] G. Mittag-Leffler, Sur la nouvelle fonction $e_\alpha(x)$, C. R. Acad. Sci. Paris (Ser. II) 137 (1903) 554–558.

[43] R. Beerends, H. G. ter Morsche, J. van den Berg, E. M. van de Vrie, Fourier and Laplace transforms, Cambridge, 2003.