Verification of the cross 3d algorithm on quantum chemistry data

by

Heinz-Jürgen Flad, Boris N. Khoromskij, Dmitry Savostyanov, and Eugene E. Tyrtyshnikov

Preprint no.: 80 2008

Verification of the cross 3d algorithm on quantum chemistry data

Heinz-Jurgen Flad, Boris N. Khoromskij,
Max-Planck-Institut für Mathematik in den Naturwissenschaften,
Inselstr. 22-26, D-04103 Leipzig, Germany; {flad, bokh}@mis.mpg.de

Dmitry V. Savostyanov, and Eugene E. Tyrtyshnikov
Institute of Numerical Mathematics, Russian Academy of Sciences,
Gubkina 8, 119333 Moscow, Russia; tee@inm.ras.ru

Abstract

A recently developed cross 3d algorithm is applied to approximation of the electron density function. The algorithm is proved to be fast and reliable on a sample of quantum chemistry data produced by the MOLPRO package.

AMS Subject Classification: 65F30, 65F50, 65N35, 65F10
Key words: Tensor approximation, Tucker core, cross 3D algorithm, quantum chemistry, electron density function, separable approximations, matrix approximations, low-rank matrices, numerical methods.

1 Introduction

Beginning from [7] and [14] the idea of tensor approximation of operators and functions has led to agreeable tools in large-scale problems of computational physics and chemistry. Recent papers show that this technique is remarkably efficient in multi-dimensional computations, where the traditional methods fail due to the storage limitations. In particular, we mention results for many-particle models based on the Hartree-Fock/Kohn-Sham equations in electronic structure calculations [12, 15, 2, 3, 17]. In the present paper we discuss numerical and algorithmic aspects of the modern tensor approximation methods with applications to electronic structure calculations.

We consider the *electron density function* $f(x, y, z)$ given as a sum of polynomially weighted gaussians

$$f(x,y,z) = \sum_{i=1}^{m}\sum_{j=1}^{m} \sigma_{ij} \Phi_{ij}^{(1)}(x)\Phi_{ij}^{(2)}(y)\Phi_{ij}^{(3)}(z),$$

where $\Phi_{ij}^{(k)}(t)$ is defined for $k = 1, 2, 3$ as follows:

$$\Phi_{ij}^{(k)}(t) = (t - t_i^{(k)})^{\beta_i^{(k)}}(t - t_j^{(k)})^{\beta_j^{(k)}} \exp\left(-(\alpha_i + \alpha_j)(t - t_{ij}^{(k)})^2\right).$$

The function $f(x, y, z)$ may be written in a form of trilinear decomposition

$$f(x, y, z) = \sum_{t=1}^{R} \sigma_t \varphi(x, x_t^{\text{orb}}, \nu_t, \alpha_t) \varphi(y, y_t^{\text{orb}}, \mu_t, \alpha_t) \varphi(z, z_t^{\text{orb}}, \lambda_t, \alpha_t),$$

or in a simpler way as

$$f(x, y, z) = \sum_{t=1}^{R} a_t(x) b_t(y) c_t(z).$$

This representation is very useful in computations as a data compression tool. After discretization of $f(x, y, z)$ on a given $n \times n \times n$ grid it requires n^3 words of memory, while the trilinear form representation is defined by $3nR$ parameters.

The tensor compression of data looks very promising in chemical computations. However, the value of tensor rank R, obtained by chemical modelling programs (e.g. MOLPRO), is quite large even for simple molecules:

$$R(CH_4) = 1334, \quad R(C_2H_6) = 3744, \quad R(C_2H_5OH) = 6945.$$

To make chemical computations really effective, we are to find a tensor approximation $\tilde{f}(x, y, z)$ to the electron density with the following properties:

$$\|f(x, y, z) - \tilde{f}(x, y, z)\| \le \varepsilon \|f(x, y, z)\|, \tag{$*_a$}$$

$$\tilde{f}(x, y, z) = \sum_{i=1}^{r} \tilde{a}_i(x) \tilde{b}_i(y) \tilde{c}_k(z). \tag{$*_b$}$$

Of course, we expect that $r \ll R$.

The numerical solution of this problem can be carried out as follows.

Algorithm 1.1 *(General scheme for trilinear approximation of a 3D function).*

1. Introduce a grid in a subdomain in $\mathbb{R}^3$. To utilise the tensor structure of data, we choose the grid as tensor product of three one-dimensional grids, and for simplicity of presentation we choose them uniform. Therefore, our grid is a set of points

$$(x_i, y_j, z_k) = (x_0 + ih, y_0 + jh, z_0 + kh), \quad 0 \le i, j, k \le N, \quad h = D/N. \tag{1.1}$$

2. Obtain an initial discretization of the electron density function on the grid

$$f_{ijk} = f(x_i, y_j, z_k) = \sum_{t=1}^{R} a_t(x_i) b_t(y_j) c_t(z_k). \tag{1.2}$$

We choose a collocation approximation method just to simplify the presentation (other grid-based approximations can be used as well). It should be emphasized that we need not to store a full 3D array data. An explicit option is to store just the values $a_t(x_i), b_t(y_j), c_t(z_k)$ using only $3RN$ memory cells. An implicit option is to deal with a procedure that can compute any entry f_{ijk} on demand but actually used to compute a relativly small subset of all the entries.

3. Approximate the data array $\mathcal{F} = [f_{ijk}]$, $0 \le i, j, k \le N$, by another array $\tilde{\mathcal{F}} = [\tilde{f}_{ijk}]$ of a smaller tensor rank. Prior to the computations, we have to define a norm to be used in $(*_a)$ and choose an appropriate discrete norm. Usually we deal with the L_2 norm and construct tensor approximations using the Frobenius norm: for a tensor $\mathcal{A} = [a_{ijk}]$ of size $n_1 \times n_2 \times n_3$ the Frobenius norm reads

$$||\mathcal{A}|| = ||\mathcal{A}||_F = \left(\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \sum_{k=1}^{n_3} a_{ijk}^2 \right)^{1/2}.$$

The approximation problem becomes of the form

$$\|\mathcal{F} - \tilde{\mathcal{F}}\|_F \le \varepsilon \|\mathcal{F}\|_F, \qquad \tilde{f}_{ijk} = \sum_{t=1}^{r} \tilde{a}_{it} \tilde{b}_{jt} \tilde{c}_{kt}, \tag{1.3}$$

where $\tilde{a}_{it}, \tilde{b}_{jt}$ and $\tilde{c}_{kt}$ can be thought of as grid values of functions $\tilde{a}_t(x), \tilde{b}_t(y)$ and $\tilde{c}_t(z)$ from $(*_b)$. We also use a shorter notation for the trilinear decomposition via a triple of matrices: $\mathcal{F} = (A, B, C)$.

The approximation step is not simple, especially for large grids and complicated molecules (when N and R are about some thousands). Most usually, it consists in the two steps:

(a) Compute a *Tucker approximation* for a given data array

$$\|\mathcal{F} - \tilde{\mathcal{F}}\|_F \le \varepsilon \|\mathcal{F}\|_F, \quad \tilde{f}_{ijk} = \sum_{i'=1}^{r_1} \sum_{j'=1}^{r_2} \sum_{k'=1}^{r_3} g_{i'j'k'} u_{ii'} v_{jj'} w_{kk'}. \tag{1.4}$$

Here, the matrices

$$U = [u_{ii'}] \in \mathbb{R}^{N \times r_1}, \quad V = [v_{jj'}] \in \mathbb{R}^{N \times r_2}, \quad W = [w_{kk'}] \in \mathbb{R}^{N \times r_3}$$

(referred to as *Tucker factors*) can be chosen orthogonal, and this is formally required by the definition of the Tucker decomposition. But in our paper we treat the orthogonality of Tucker factors as an additional property and consider Tucker-like decompositions with general U, V and W as well. We also use a shorter tensor-matrix notation:

$$\tilde{\mathcal{F}} = \mathcal{G} \times_1 U \times_2 V \times_3 W.$$

The symbol "$\times_p$" is used to define a tensor-by-matrix contraction along the mode p; for a tensor $\mathcal{A} = [a_{ijk}] \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and a matrix $B \in \mathbb{R}^{m_2 \times n_2}$ the result of "$\mathcal{A} \times_2 B$" is a tensor $\mathcal{C} = [c_{ijk}] \in \mathbb{R}^{n_1 \times m_2 \times n_3}$ with elements

$$c_{ijk} = \sum_{j'=1}^{n} a_{ij'k} b_{jj'}.$$

The Tucker approximation with orthogonal factors is often used as a *dimensionality reduction* tool, in a way similar to the SVD. Despite that all entries of the initial array may not differ much in the magnitude, the entries of the Tucker core array $\mathcal{G}$ get to vary in the magnitude a lot stronger (like singular values of a matrix) and many can be neglected without a big loss in accuracy. In the result, we can work with a reduced core array whose mode sizes are considerably smaller than the initial ones: $r_1, r_2, r_3 \ll N$.

(b) Consider the *reduced core array* $\mathcal{G} = [g_{i'j'k'}]$ and approximate it in the trilinear format:
$$g_{i'j'k'} = \sum_{t=1}^{r} u'_{i't} v'_{jt} w'_{k't}.$$
Substitute the result in (1.4) and build up the trilinear approximation (1.3) for the array $\tilde{\mathcal{F}}$.

4. When a trilinear approximation (1.3) for $\mathcal{F}$ is computed, we can interpolate the trilinear factors u_{it}, v_{it}, w_{kt} and come up with a trilinear approximation $(*)$ to the initial function.

The two-step procedure for computation of (1.3) is more viable, because the trilinear decomposition (which is the most tricky part of computation) is now applied to a small array.

A detailed discussion of the two-level tensor representations (formats) can be found in [16, 18, 19]. All the same, the Tucker approximation of large-scale arrays requires some special tools. A standard method for the Tucker approximation involves three SVD applied to *unfoldings* of $\mathcal{F}$. The cost ammounts to $\mathcal{O}(N^4)$ flops, which we cannot afford. Utilising the initial trilinear decomposition, we reduce the complexity to $\mathcal{O}(NR^2)$, this approach is better but still very slow for large R and N. Thus, we should be interested to apply the cross 3D algorithm proposed in [21] as it requires $\mathcal{O}(Nr^a)$ flops, $1 \le a \le 2$.

2 SVD-based algorithms for Tucker approximation

2.1 Tucker decomposition via SVD

The unfoldings of $n_1 \times n_2 \times n_3$ tensor $\mathcal{A}$ are rectangular matrices $A^{(1)}$ of size $n_1 \times n_2 n_3$, $A^{(2)}$ of size $n_2 \times n_1 n_3$ and $A^{(3)}$ of size $n_3 \times n_1 n_2$, with elements
$$A^{(1)} = [a^{(1)}_{ijk}] = [a_{i(jk)}], \quad A^{(2)} = [a^{(2)}_{ijk}] = [a_{j(ki)}], \quad A^{(3)} = [a^{(3)}_{ijk}] = [a_{k(ij)}].$$
The superscripts $1, 2, 3$ in the definitions of unfoldings point to the mode index (first, second or third), two other mode indices are merged into one "long index".

A well-known method for the computation of the Tucker decomposition is based on the SVD algorithm.

Algorithm 2.1 *(Tucker decomposition/approximation).*

1. Given data array $\mathcal{F}$, consider three rectangular *unfolding* matrices of appropriate sizes $F^{(1)}, F^{(2)}, F^{(3)}$, which contain mode vectors (columns along i, rows along j and *fibers* along k).

Table 2.1: Memory and time requirements for Tucker approximation via 3 SVDs

grid size N	1280	2560	5120
memory for N^3 elements	16Gb	125Gb	1 Tb
time for SVD-s*	7 hours	5 days	76 days

* Time is given for hypothetic $10GHz$ CPU, working at 100% efficiency on SVD algorithm with complexity $32mn\min(m,n)$ flops (where $m \times n$ is matrix size).

2. The left ("short") singular vectors of the SVD-s of these matrices

$$F^{(1)} = U\Sigma_1\Phi_1^T, \qquad F^{(2)} = V\Sigma_2\Phi_2^T, \qquad F^{(3)} = W\Sigma_3\Phi_3^T \tag{2.1}$$

give the factors U, V, W of the Tucker decomposition. A Tucker approximation can be obtained with prescribed accuracy by appropriate truncation of singular vectors in Tucker factors, corresponding to the singular values below the chosen threshold.

3. The core is computed as contraction of the data array with the Tucker factors

$$g_{i'j'k'} = \sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{k=0}^{N} a_{ijk}u_{ii'}v_{jj'}w_{kk'}, \qquad \text{or} \qquad \mathcal{G} = \mathcal{A} \times_1 U^T \times_2 V^T \times_3 W^T. \tag{2.2}$$

This method is quite simple and provide us with a robust tool for approximation of data arrays with prescribed accuracy. However, it is computationally expensive: for an $N \times N \times N$ array $\mathcal{F}$, the complexity for the SVDs is $\mathcal{O}(N^4)$, and memory to keep the unfoldings is $3N^3$. In Table 2.1 it is easy to see, that these performance requirements are tough for typical grids used in the modelling in quantum chemistry. Therefore, the standard Tucker decomposition method is not feasible for our purposes.

2.2 SVD filtering of trilinear factors

To develop a faster method for the Tucker approximation, let us make use of the initial trilinear structure of data (1.2). If the tensor rank R is sufficiently small, we can proceed in a trivial way.

Algorithm 2.2 *(Low-rank 3L $\to$ Tucker)*

1. Compute the QR decomposition of the trilinear factors

$$A = UR_a, \quad B = VR_b, \quad C = WR_c, \qquad R_a, R_b, R_c \in \mathbb{R}^{R\times R}$$

2. Form the core array $\mathcal{G} = [g_{i'j'k'}]$

$$g_{i'j'k'} = \sum_{t=1}^{R}(r_a)_{i't}(r_b)_{j't}(r_c)_{k't}, \qquad 1 \le i', j', k' \le R.$$

3. Apply the Tucker approximation Algorithm 2.1 to the core array and reduce the dimensionality:

$$\|\mathcal{G} - \tilde{\mathcal{G}}\|_F \leq \varepsilon \|\mathcal{G}\|_F, \quad \tilde{\mathcal{G}} = \mathcal{H} \times_1 U' \times_2 V' \times_3 W', \qquad \mathcal{H} \in \mathbb{R}^{r_1 \times r_2 \times r_3}.$$

4. Finally, the Tucker approximation of $\mathcal{F}$ reads

$$\tilde{F} = \mathcal{H} \times_1 UU' \times_2 VV' \times_3 WW'$$

One part of this method is the Tucker decomposition of an $R \times R \times R$ array and therefore requires $\mathcal{O}(NR^2 + R^4)$ flops. Hence, it is also inapplicable when R exceeds some thousands.

To improve the method in case of large R, we perform pre-filtering of the Tucker factors U, V, W prior to computing the core $\mathcal{G}$, i.e. before step 2 of Algorithm 2.2. This filtering changes trilinear factors A, B, C so that we can control accuracies

$$\varepsilon_1 = \|A - \tilde{A}\|_F, \quad \varepsilon_2 = \|B - \tilde{B}\|_F, \quad \varepsilon_3 = \|C - \tilde{C}\|_F.$$

The question is how should we choose $\varepsilon_1, \varepsilon_2$ and ε_3 for the required accuracy in (1.3) to be guaranteed? Let $\mathcal{F} = (A, B, C)$ and $\tilde{\mathcal{F}} = (\tilde{A}, \tilde{B}, \tilde{C})$. Assuming that differences

$$\Delta A = A - \tilde{A}, \qquad \Delta B = B - \tilde{B}, \qquad \Delta C = C - \tilde{C}$$

are small enough, we neglect squares and cubes of them and write down

$$\mathcal{F} - \tilde{\mathcal{F}} \approx (\Delta A, B, C) + (A, \Delta B, C) + (A, B, \Delta C),$$

$$\|\mathcal{F} - \tilde{\mathcal{F}}\|_F \leq \|(\Delta A, B, C)\|_F + \|(A, \Delta B, C)\|_F + \|(A, B, \Delta C)\|_F.$$

To bound $\|(\Delta A, B, C)\|_F$, consider the contraction of this array along first index. This does not change the Frobenius norm

$$\|(\Delta A, B, C)\|_F = \left\| \sum_{t=1}^{R} \Delta a_{it} b_{jt} c_{kt} \right\|_F = \|\Delta A K\|_F \leq \|\Delta A\|_F \|K\|_F,$$

where

$$K = [k_{(jk)t}] \in \mathbb{R}^{N^2 \times R}, \quad k_{(jk)t} = b_{jt} c_{kt}, \quad \|K\|_F^2 = \sum_{t=1}^{R} \|b_t\|_2^2 \|c_t\|_2^2$$

Therefore, we show that

$$\|(\Delta A, B, C)\|_F \leq \|\Delta A\|_F \sqrt{\sum_{t=1}^{R} \|b_t\|_2^2 \|c_t\|_2^2}.$$

Of course, we can write also $\|(\Delta A, B, C)\|_F \leq \|\Delta A\|_F \|B\|_F \|C\|_F$, but this pretty-looking bound is too much overestimated to be good for practice.

Finally, we have

$$\|\mathcal{F} - \tilde{\mathcal{F}}\|_F \le \|\mathrm{A}|_F\|K\|_F + \|\Delta B\|_F\|L\|_F + \|\Delta C\|_F\|M\|_F,$$
$$\|K\|_F^2 = \sum_{t=1}^{R} \|b_t\|_2^2\|c_t\|_2^2, \quad \|L\|_F^2 = \sum_{t=1}^{R} \|a_t\|_2^2\|c_t\|_2^2, \quad \|M\|_F^2 = \sum_{t=1}^{R} \|a_t\|_2^2\|b_t\|_2^2. \tag{2.3}$$

Now we can set

$$\varepsilon_1 = (\varepsilon/3)\|\mathcal{F}\|_F/\|K\|_F, \quad \varepsilon_2 = (\varepsilon/3)\|\mathcal{F}\|_F/\|L\|_F, \quad \varepsilon_3 = (\varepsilon/3)\|\mathcal{F}\|_F/\|M\|_F, \tag{2.4}$$

filter trilinear factors with thresholds

$$\|\Delta A\|_F \le \varepsilon_1, \quad \|\Delta B\|_F \le \varepsilon_2, \quad \|\Delta C\|_F \le \varepsilon_3,$$

and have $\|\mathcal{F} - \tilde{\mathcal{F}}|_F \le \varepsilon\|F\|_F$.

To complete the prescriptions, we should find a method of fast computation for $\|\mathcal{F}\|_F = \|(A, B, C)\|_F$. Here it is:

$$\|\mathcal{F}\|_F^2 = \sum_{i,j,k=0}^{N} \left(\sum_{t=1}^{R} a_{it}b_{jt}c_{kt}\right)^2 = \sum_{t,\tau=1}^{R} \left(\sum_{i=1}^{N} a_{it}a_{i\tau}\right)\left(\sum_{j=1}^{N} b_{jt}b_{j\tau}\right)\left(\sum_{k=1}^{N} c_{kt}c_{k\tau}\right).$$

$$\|\mathcal{F}\|_F^2 = \sum_{t,\tau=1}^{R} (G_A \circ G_B \circ G_C)_{t\tau}\,, \qquad G_A = A^TA, \quad G_B = B^TB, \quad G_C = C^TC, \tag{2.5}$$

where "$\circ$" defines element-by-element multiplication of matrices.

We are ready to present

Algorithm 2.3 *(High-rank 3L $\to$ Tucker)*

1. Compute the norm of a given data array $\mathcal{F} = (A, B, C)$ as shown in (2.5). It requires $6NR^2$ flops for computing the Gram matrices and $3R^2$ flops to compute the sum of elements of their product.

2. Perform SVD-s with the factors A, B and C and filter small singular numbers and vectors using the threshold values $\varepsilon_1, \varepsilon_2$ and ε_3, defined in (2.4):

$$\|A - \tilde{A}\|_F \le \varepsilon_1, \qquad \|B - \tilde{B}\|_F \le \varepsilon_2, \qquad \|C - \tilde{C}\|_F \le \varepsilon_3,$$
$$\tilde{A} = U\Sigma_1\Phi_1, \quad \tilde{B} = V\Sigma_2\Phi_2, \quad \tilde{C} = W\Sigma_3\Phi_3,$$
$$U \in \mathbb{R}^{N\times\rho_1}, \quad V \in \mathbb{R}^{N\times\rho_2}, \quad W \in \mathbb{R}^{N\times\rho_3}.$$

   This step requires $\mathcal{O}(NR\min(N, R))$ flops.

3. Rewrite $\tilde{\mathcal{F}} = (U\Sigma_1\Phi_1, V\Sigma_2\Phi_2, W\Sigma_3\Phi_3)$ as the Tucker decomposition

$$\tilde{\mathcal{F}} = \mathcal{G} \times_1 U \times_2 V \times_3 W, \quad \mathcal{G} = (\Sigma_1\Phi_1, \Sigma_2\Phi_2, \Sigma_3\Phi_3) \in \mathbb{R}^{\rho_1\times\rho_2\times\rho_3}.$$

Table 2.2: Tucker approximation results by Algorithm 2.3, $N = 5120$, $\varepsilon = 10^{-7}$

molecule	R	mode ranks ρ_1, ρ_2, ρ_3	mode ranks r_1, r_2, r_3	time*
CH_4	1334	$79 \times 86 \times 86$	$41 \times 41 \times 41$	5.5min
C_2H_6	3744	$80 \times 100 \times 133$	$30 \times 50 \times 44$	1h40min
C_2H_5OH	6945	$129 \times 198 \times 176$	$70 \times 70 \times 72$	14h 20min

* Time is measured on $2.2GHz$ Pentium4 CPU, with code compiled by g77/GNU Fortran (GCC) 3.4.6 compiler, optimised by `-O2 option and linked with GotoBLAS-1.24 library.`

4. Optionally, we can apply the Tucker approximation algorithm 2.1 to $\mathcal{G}$ and re-approximate the core.

$$\|\mathcal{G} - \tilde{\mathcal{G}}\|_F \leq \varepsilon \|\mathcal{G}\|_F, \qquad \mathcal{G} = \mathcal{H} \times_1 U' \times_2 V' \times_3 W',$$

$$\tilde{F} = \mathcal{H} \times_1 UU' \times_2 VV' \times_3 WW', \qquad \mathcal{H} \in \mathbb{R}^{r_1 \times r_2 \times r_3}.$$

   As a rule, this leads to a significant reduction in the mode ranks, because the thresholds $\varepsilon_1, \varepsilon_2$ and ε_3 applied on the step 2 are not exact. It requires $\mathcal{O}(\rho^4)$ flops with $\rho = \max(\rho_1, \rho_2, \rho_3)$.

The overall complexity of the presented method is $\mathcal{O}(NR^2)$. Numerical results and timings are given in Table 2.2. The performance is acceptable for CH_4, but for bigger molecules the running time is still too large.

3 Cross 3D approximation method

There are four main ideas behind the cross 3d method [21].

3.1 Computation of data array entries only on demand

Preliminary experiments ensure us that data arrays $\mathcal{F}$ computed as discretization of the electron density function are likely to admit the Tucker approximation (1.4) with reduced mode ranks r_1, r_2, r_3 for a given relative accuracy ε. The same data reduction is achieved by using a Tucker-like approximation

$$\tilde{\mathcal{F}} = G' \times_1 U' \times_2 V' \times_3 W', \qquad G' \in \mathbb{R}^{r_1 \times r_2 \times r_3} \tag{3.1}$$

with (non-orthogonal) matrices $U' \in \mathbb{R}^{N \times r_1}$, $V' \in \mathbb{R}^{N \times r_2}$, $W' \in \mathbb{R}^{N \times r_3}$, consisting of columns, rows and fibers of $\mathcal{F}$. The accuracy of such an approximation is $c\varepsilon$, where the *deterioration coefficient* $c > 1$ depends only on sizes and ranks, but not on the array entries [20, 21]. Moreover, the elements of the core tensor G' can be computed from the elements of U', V', W' [20, 21].

A nice property of multidimensional arrays in the chemical applications is that we *do not* need the whole array $\mathcal{F}$ to compute its Tucker approximation $\tilde{F}$. All we need is the knowledge of a small number of *very important* columns, rows and fibers, that represent with prescribed accuracy the subspace of the Tucker factors and therefore the subspace spanned by all columns/rows/fibers of the given array. The existence of such an approximation and setting it as a target is the first key idea on which Cross 3D hinges.

3.2 Maximum-volume principle in matrix approximation

Consider a matrix approximation problem: given a matrix $F \in \mathbb{R}^{n\times n}$, find a low-rank approximation

$$\|F - \tilde{F}\|_F \leq \|F\|_F, \qquad F = UGV^T, \qquad U, V \in \mathbb{R}^{n\times r}, G \in \mathbb{R}^{r\times r}$$

with a prescribed relative accuracy ε. This problem can be solved by the SVD, but it requires n^3 flops and all elements of F to be handled. We are looking for faster methods that do not require all elements of F. Let us restrict ourselves to the *skeleton decomposition*, where U and V consist of columns and rows of F, and $G = B^{-1}$, where B is $r \times r$-submatrix at the intersection of the columns of U and rows of V. In [22], a good choice for B was proposed and substantiated: we should search for a submatrix B with the maximal modulus of determinant (this value is referred as *volume*) among all $r \times r$ submatrices. This *maximal volume* submatrix excels in a good accuracy bound to hold [22]. In practice, maximum-volume approximation is not the best, but is close to the best skeleton and SVD approximations. The maximum-volume strategy is the second key idea in Cross 3D.

3.3 Adaptive search for the good cross

The search for a maximum-volume submatrix is NP-difficult, so there is no fast and robust method to solve this problem as it is exactly. However, we may be satisfied with a "sufficiently good" submatrix and some heuristic algorithms. Since these algorithms are to fetch a cross of some columns and rows, we call them *cross algorithms*. Probably the most simple and effective cross algorithm is the Gauss elimination method using some pivoting technique over dynamically selected sets of the entries of the "active matrix". Using the column and row pivoting considered in [9] we observe that this method is simple but may have breakdowns (quiting when a good approximation is not obtained) if applied as it is. A cheap practical remedy proposed in [24] is a restarted version of this cross method. For the readers convenience, we give here a brief description of the algorithm.

Algorithm 3.1 *(Cross 2D)*
Given a matrix F of approximate rank r, approximate it by a matrix $\tilde{F}_r$, which is a sum of r rank-1 matrices $u_p v_p^T$ (so-called *skeletons*).

1. Calculate column j_p of the matrix F (on first iteration set $p = 1$ and pick random j_p) and subtract from all elements the corresponding elements of already calculated skeletons (on first iteration there are none). In the resulting vector find the largest magnitude element. Suppose it is located in the row i_p.

2. Calculate the row i_p of the residue and the next pivot which is its largest magnitude element with a restriction that the element from the j_p-th column can not be chosen again. Suppose this pivot is located in the j_{p+1}-th column.

Table 3.1: Tucker approximation results by Algorithm Cross 3D, $N = 5120$, $\varepsilon = 10^{-7}$

molecule	R	mode ranks ρ_1, ρ_2, ρ_3	mode ranks r_1, r_2, r_3	time*
CH_4	1334	$48 \times 46 \times 47$	$41 \times 41 \times 41$	14.5min
C_2H_6	3744	$34 \times 60 \times 53$	$30 \times 50 \times 44$	34min
C_2H_5OH	6945	$81 \times 85 \times 87$	$69 \times 70 \times 72$	2h 50min

ρ_1, ρ_2, ρ_3 — number of columns, rows and fibers, computed by Cross 3D method;
r_1, r_2, r_3 — mode ranks after Tucker re-approximation (like in step 4 of Algorithm 2.3).
$\star$ Time is measured on 2.2GHz Pentium4 CPU, with code compiled by g77/GNU Fortran (GCC) 3.4.6 compiler, optimised by `-O2 option and linked with GotoBLAS-1.24 library.`

3. Calculate the new cross with center at (i_p, j_p). Update $\tilde{F} := \tilde{F} + u_p v_p^T$ to make the approximation *exact* on the elements, occupied by new cross.

4. If a stopping criterion $\|F - \tilde{F}\|_F \leq \|F\|_F$ is not satisfied, set $p := p + 1$ and go to step 1.

This algorithm provides us with a fast and robust tool of matrix approximation. We need to generalise it to 3D arrays.

3.4 Internal cross 2d procedure

The central part of the Tucker decomposition algorithm 2.1 is the SVD applied to matrix unfoldings.

Suppose we do the same things with some other tool, i.e. apply Cross 2D algorithm to unfoldings of $\mathcal{F}$. Let us start with $F^{(3)}$. All we have to compute from this matrix are:

columns	(*short vectors* of size N presenting fibers of $\mathcal{F}$)
and rows	(*long vectors* of size N^2 presenting slices of $\mathcal{F}$).

The last key idea of Cross 3D is that we can also apply 'internal' Cross 2D procedure to compute slices in the skeleton decomposition ansatz with common column and row matrices for all slices. On this way we come up with an algorithm of *linear by N* complexity.

3.5 Cross algorithm for 3D arrays

Algorithm 3.2 *(Cross 3D, simplified version)*

Given an array $\mathcal{F}$ approximate it as shown in (3.1)

0. Numbering the steps by p, set p to 1. Choose a slice $A_k = [(a_k)_{ij}]$ in $\mathcal{A}$, for example, and assume its index to be k_1. Set $\tilde{\mathcal{A}} = 0$.

1a. Find an approximation A_{k_p} to the k_p-th slice of the residue $\mathcal{R} = \mathcal{A} - \tilde{\mathcal{A}}_p$ by the cross-method:

$$A_{k_p} = \sum_{q=1}^{r} u_q v_q^T.$$

1b. Find the largest magnitude element in the matrix A_{k_p}, let it be located at (i_p, j_p).

2. Compute the vector w corresponding to the fiber of $\mathcal{R}$ with index (i_p, j_p)

$$w_k = \mathcal{R}_{i_p,j_p,k},$$

perform the scaling

$$w := w/w_{k_p}$$

and find in w the largest magnitude element from those whose index is not equal to k_p. Suppose it is located at the k_{p+1}-th position of w.

3. Compute a new approximation:

$$\tilde{\mathcal{A}} := \tilde{\mathcal{A}} + A_{k_p} \times w = \tilde{\mathcal{A}} + \left(\sum_{q=1}^{r} u_q v_q^T\right) \times w = \tilde{\mathcal{A}} + \sum_{q=1}^{r} u_q \times v_q \times w.$$

4. Check accuracy on a sample of untouched entries. If it is not satisfactory, then set $p := p + 1$, and go to step 1.

This algorithm is linear by N (but every computation of array element f_{ijk} requires R flops, so overall complexity is $\mathcal{O}(NR)$). However, it is far from being robust. We have to solve a lot of interesting matrix problems (for example, at step 1b we need to find a maximum element of a matrix from its low-rank format) and implement a bunch of matrix tricks to boost the speed of method. You can find the mathematical background and all details of this algorithm in [21]. Notice that the so-called multigrid accelerated best orthogonal Tucker approximation of rank-R trilinear decomposition is presented in [19]. This method is proven to have linear scaling in all significant characteristics of the model, $O(drRN)$, where d is the dimensionality parameter and r is the Tucker rank. Extensive numerics on the trilinear and Tucker approximation of operators and functions in electronic structure calculations are presented in [2, 3].

3.6 Numerical results

The numerical results and timings for Cross 3D are given in Table 3.1. Comparing Cross 3D with prefiltering trilinear factors (Algorithm 2.3), we find that Cross 3D method is 3 times faster for C_2H_6 and 6 times faster for C_2H_5OH. You can also compare values of 'actually computed' mode ranks ρ_1, ρ_2, ρ_3 and values of mode ranks r_1, r_2, r_3 after Tucker re-approximation for both methods. It is easy to note, that for Cross 3D method computed mode ranks are closer to 'real' mode ranks. This means that Cross 3D has implemented better stopping criteria, than SVD-based method, and performs less work to provide the approximation with prescribed final accuracy. We can optimistically say, that for more complicated molecules Cross 3D can give even better advance.

We also take a look at mode ranks and timings of Cross 3D approximating electron densities of different molecules with accuracies varied from 10^{-3} to 10^{-10}. The results are in Table 3.2.

Table 3.2: Tucker mode ranks and timings or different accuracies, $N = 5120$

ε	CH_4		C_2H_6		C_2H_5OH	
10^{-3}	$14 \times 14 \times 14$	1:05*	$11 \times 18 \times 15$	3:40	$26 \times 26 \times 25$	23:15
10^{-4}	$20 \times 20 \times 21$	1:45	$14 \times 24 \times 21$	5:40	$34 \times 35 \times 35$	43:00
10^{-5}	$28 \times 28 \times 29$	2:15	$20 \times 36 \times 30$	9:00	$47 \times 47 \times 47$	58:45
10^{-6}	$35 \times 35 \times 35$	3:10	$24 \times 45 \times 37$	11:20	$59 \times 60 \times 61$	1:24:45
10^{-7}	$41 \times 41 \times 41$	4:00	$30 \times 50 \times 44$	14:30	$70 \times 70 \times 71$	2:00:00
10^{-8}	$49 \times 48 \times 48$	5:30	$37 \times 60 \times 52$	20:10	$83 \times 82 \times 85$	2:47:30
10^{-9}	$55 \times 55 \times 55$	7:20	$45 \times 69 \times 59$	27:30	$95 \times 93 \times 98$	3:41:40
10^{-10}	$60 \times 60 \times 60$	10:20	$52 \times 78 \times 67$	32:00	$105 \times 104 \times 110$	4:48:10

* Time is given as [hh]:mm:ss. It is measured on 3.2GHz Pentium4 CPU, with code compiled by ifort (IFORT) 9.0 20060120 Fortran compiler, optimised by `-O3 -tpp7 -axW -aW options and linked with GotoBLAS-1.24 library.`

Table 4.1: Tucker approximation results by the multigrid Cross 3D, $N = 5120$, $\varepsilon = 10^{-7}$

molecule	R	size of VIP sets ρ_1, ρ_2, ρ_3	mode ranks r_1, r_2, r_3	time*
CH_4	1334	$61 \times 70 \times 71$	$41 \times 41 \times 41$	5min
C_2H_6	3744	$61 \times 76 \times 64$	$30 \times 50 \times 44$	12.5min
C_2H_5OH	6945	$87 \times 113 \times 146$	$68 \times 70 \times 72$	1hour

* Time is measured on 2.2GHz Pentium4 CPU, with code compiled by g77/GNU Fortran (GCC) 3.4.6 compiler, optimised by `-O2 option and linked with GotoBLAS-1.24 library.`

4 A miltigrid version of the cross 3D algorithm

The cross 3D algorithm computes the (r_1, r_2, r_3) Tucker approximation of a data array from some r_1 columns, r_2 rows and r_3 fibers of data array. These columns, rows and fibers can be referrred to as "very important vectors", or shortly "VIP vectors". A tricky part is how to choose the VIP vectors. However, if we knew the positions of these vectors in the data array, our work would be almost done. Since the data is quite smooth, we can probably take some information using a coarser grid. A multigrid accelerated best orthogonal Tucker approximation is presented in [19]. It seems just natural to accommodate this sort of idea into the general version of the cross 3D algorithm.

Suppose we are to find the Tucker approximation for a discretization of $F(x, y, z)$ on a grid (1.1) with $N = N_L = n_0 2^L$. To speed up computations, we can first apply Cross 3D to a similar discretization of the same function on a coarser grid with $N = n_0 2^{L-1}$, remember the indices or coordinates $(y, z), (z, x)$ and (x, y) of the VIP columns, rows and fibers, and then exploit these already known indices when computing the Tucker approximation for the data array with $N = N_L$. Thus we repeat Cross 3D for a sequence of grids but so that it works faster due to some additional information taken from the coarser grids.

Algorithm 4.1 *(Cross 3D, multigrid version)*

0. Start with a grid of size $N = N_l = n_0 2^l$ with $l = l_0$. Apply Algorithm 3.2 to compute the Tucker approximation $\tilde{\mathcal{F}}^{[l_0]}$ for $\mathcal{F}^{[l_0]} = [f(x_i, y_j, z_k)]$. Acquire and retain the positions of ρ_1 rows, ρ_2 columns and ρ_3 fibers computed by Cross 3D as VIP sets $\mathfrak{I}_1, \mathfrak{I}_2$ and $\mathfrak{I}_3$.

1. Consider a next-level grid with $N = N_{l+1}$. Array $\mathcal{F}^{[l]}$ from the previous level is a subarray of $\mathcal{F}^{[l+1]} = [f(x_i, y_j, z_k)]$ with all indices i, j, k even. To find the positions of good mode vectors in $\mathcal{F}^{[l+1]}$, multiply all indices in VIP sets $\mathfrak{I}_1, \mathfrak{I}_2$ and $\mathfrak{I}_3$ by two. Compute them and from matrices U, V, W as a non-orthogonal basis for the Tucker approximation.

2. Othogonalize U, V and W and find maximum-volume submatrices $U_\square, V_\square$ and $W_\square$ in them (a suitable algorithm is presented in [13, 20, 21]). Compute a subarray $\mathcal{F}_\square = [f_{ijk}]$ with indices i, j, k corresponding to positions of $U_\square$ in U, $V_\square$ in V and $W_\square$ in W respectively. Compute the Tucker approximation $\tilde{\mathcal{F}}$ with the factors U, V, W, so that it is *exact* on the elements of $\mathcal{F}_\square$:
$$\tilde{\mathcal{F}}^{[l]} = \mathcal{G} \times_1 U \times_2 V \times_3 W, \qquad \mathcal{G} = \mathcal{F}_\square \times_1 U_\square^{-1} \times_2 V_\square^{-1} \times_3 W_\square^{-1}.$$

3. Use the approximation $\tilde{\mathcal{F}}^{[l]}$ as an initial guess to start Cross 3D for the next-level grid. On return we have an updated approximation $\tilde{\mathcal{F}}^{[l]}$ and additionally computed $\rho_1^{[l]}$ rows, $\rho_2^{[l]}$ columns and $\rho_3^{[l]}$ fibers. Add the positions of these additional vectors to VIP sets $\mathfrak{I}_1, \mathfrak{I}_2$ and $\mathfrak{I}_3$. Set $\rho_p := \rho_p + \rho_p^{[l]}$, $p = 1, 2, 3$.

4. Set $l := l + 1$ and if $l < L$, go back to step 1.

This algorithm is linear in N, R and $\max(r_1, r_2, r_3)$,, where r_1, r_2, r_3 are the Tucker mode ranks. For smooth functions $F(x, y, z)$ this multigrid version outperforms the standard Cross 3D. It can be also used to define the optimal grid size N for a given approximation accuracy ε without an *a priori* information about the smoothness of $F(x, y, z)$.

The numerical results and timings are given in Table 4.1. Note that sizes of VIP sets ρ_1, ρ_2, ρ_3 are considerably large. This may signify that some of VIP vectors obtained from low levels are not very good as approximations on finer levels. Some filtering of sets $\mathfrak{I}_1, \mathfrak{I}_2, \mathfrak{I}_3$ can be applied on Step 3, but this still needs to be studied in more detail. Nevertheless, the current version of the multigrid Cross 3D algorithm is 3 times faster than the standard Cross 3D for all considered molecules. It is also interesting to compare timings of the multigrid Cross 3D (given in Table 4.2) and timings of the plain Cross 3D (Table 3.2) for different accuracies and molecules.

Acknowledgements. The authors are appreciative to M. Phil. S. R. Chinnamsettey (MPI MIS, Leipzig) for providing the numerical data obtained from MOLPRO package.

References

[1] R. A. Harshman, Foundations of the Parafac procedure: Models and conditions for an explanatory multimodal factor analysis, *UCLA Working Papers in Phonetics.* V. 16. P. 1-84(1970).

Table 4.2: Sizes of VIP sets and timings on different accuracies, $N = 5120$

ε	CH_4		C_2H_6		C_2H_5OH	
10^{-3}	$38 \times 19 \times 35$	1:00*	$33 \times 40 \times 21$	3:25	$38 \times 37 \times 47$	07:00
10^{-4}	$35 \times 43 \times 33$	1:15	$23 \times 41 \times 38$	3:00	$59 \times 52 \times 66$	11:00
10^{-5}	$55 \times 42 \times 50$	1:50	$34 \times 51 \times 50$	3:50	$71 \times 72 \times 79$	16:20
10^{-6}	$58 \times 59 \times 55$	2:20	$52 \times 66 \times 63$	6:00	$89 \times 94 \times 99$	25:10
10^{-7}	$69 \times 62 \times 69$	3:10	$56 \times 71 \times 68$	7:30	$108 \times 92 \times 135$	38:00
10^{-8}	$79 \times 74 \times 85$	4:50	$60 \times 84 \times 82$	9:10	$127 \times 123 \times 136$	58:00
10^{-9}	$91 \times 81 \times 82$	5:40	$69 \times 102 \times 87$	12:20	$136 \times 135 \times 164$	1:25:10
10^{-10}	$99 \times 98 \times 99$	7:20	$84 \times 103 \times 129$	19:20	$176 \times 166 \times 161$	2:04:20

* Time is given as [hh]:mm:ss. It is measured on 3.2GHz Pentium4 CPU, with code compiled by ifort (IFORT) 9.0 20060120 Fortran compiler, optimised by `-O3 -tpp7 -axW -aW options and linked with GotoBLAS-1.24 library.`

[2] S.R. Chinnamsetty, M. Espig, W. Hackbusch, B.N. Khoromskij and H.-J- Flad: *Optimal Kronecker tensor-product approximation in quantum chemistry.* J. Chem. Phys. **127**, 084110 (2007).

[3] S.R. Chinnamsetty, H.-J- Flad, V. Khoromskaia and B.N. Khoromskij: *Tensor decomposition in electronic structure calculations on Cartesian grids.* MPI MIS 65, Leipzig 2007 (submitted).

[4] P. Comon, Tensor decomposition: State of the Art and Applications, *IMA Conf. Math. in Signal Proc.* Warwick, UK, Dec. 18-20, 2000.
`http://www.i3s.fr/~comon/FichiersPs/ima2000.ps`

[5] J. D. Caroll, J. J. Chang, Analysis of individual differences in multidimensional scaling via n-way generalization of Eckart-Young decomposition, *Psychometrica.* V.35. P. 283-319(1970).

[6] R. Bro, PARAFAC: Tutorial and applications *Chemom. Intelligent Lab. Systems.*, V. 38. pp. 149-171 (1997).

[7] G. Beylkin, M.M. Mohlenkamp, Numerical operator calculus in higher dimensions, *PNAS*, V. 99, No. 16, pp. 10246-10251(2002)

[8] M. A. O. Vasilescu, D. Terzopoulos, Multilinear Image Analysis for Facial Recognition, Proc. of Int. Conf. on Pattern Recognition (ICPR 2002), V. 2, Quebec City, Canada, Aug, pp. 511-514 (2002).

[9] M. Bebendorf, Approximation of boundary element matrices, Numer. Math., V. 86, No. 4, P. 565–589 (2000).

[10] L. De Lathauwer, B. De Moor and J. Vandewalle, A multilinear singular value decomposition, *SIAM J. Matrix Analysis Appl.*, 21, pp. 1253–1278 (2000).

[11] L. De Lathauwer, B. De Moor and J. Vandewalle, On best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of high-order tensors, *SIAM J. Matrix Analysis Appl.*, 21, pp. 1324–1342 (2000).

[12] H.-J. Flad, W. Hackbusch, B.N. Khoromskij, and R. Schneider: *Concept of data-sparse tensor-product approximation in many-particle modeling.* MPI MIS 3, Leipzig 2008 (submitted).

[13] S. A. Goreinov, Pseudoskeleton approximations of block matrices generated by asymptotically smooth kernels, Ph. D. Thesis, INM RAS, 2001 (in Russian).

[14] W. Hackbusch, B.N. Khoromskij, and E. Tyrtyshnikov: *Hierarchical Kronecker tensor-product approximation.* J. Numer. Math. **13** (2005), 119-156.

[15] R.J. Harrison, G.I. Fann, T. Yanai, Z. Gan, and G. Beylkin: *Multiresolution quantum chemistry: Basic theory and initial applications.* J. Chem. Phys., 121 (23): 11587-11598, 2004.

[16] B.N. Khoromskij: *Structured Rank-$(r_1, ..., r_d)$ Decomposition of Function-related Tensors in $\mathbb{R}^d$.* Comp. Meth. in Applied Math., **6** (2006), 2, 194-220.

[17] B. N. Khoromskij: *On tensor approximation of Green iterations for Kohn-Sham equations.* Computing and Visualization in Scince, **11**:259-271 (2008); DOI:10.1007/s00791-008-0097-x, 2008.

[18] B.N. Khoromskij and V. Khoromskaia: *Low Rank Tucker Tensor Approximation to the Classical Potentials.* Central European J. of Math., **5**(3) 2007, 1-28.

[19] B.N. Khoromskij and V. Khoromskaia: *Multigrid Accelerated Tensor Approximation of Function Related Multi-dimensional Arrays.* Prepriont 40, MPI MIS, Leipzig 2008 (SIAM J. Sci. Comp., submitted).

[20] D. V. Savostyanov, Polilinear approximation of matrices and integral equations, Ph. D. Thesis, INM RAS, 2006 (in Russian).

[21] I. V. Oseledets, D. V. Savostianov, E. E. Tyrtyshnikov, Tucker dimensionality reduction of three-dimensional arrays in linear time, *SIMAX*, **30**, 3, 939-956 (2008).

[22] S. A. Goreinov, E. E. Tyrtyshnikov, The maximal-volume concept in approximation by low-rank matrices, *Contemporary Mathematics*, V. 208, P. 47–51 (2001).

[23] S. A. Goreinov, E. E. Tyrtyshnikov, N. L. Zamarashkin, A theory of pseudo–skeleton approximations, *Linear Algebra Appl.* 261, P. 1–21 (1997).

[24] D. V. Savostyanov, *Mosaic-skeleton approximations*, Master Thesis, INM RAS, 2001 (in Russian).

[25] E. E. Tyrtyshnikov, Mosaic–skeleton approximations, *Calcolo*, V. 33 (1-2), P. 47–57 (1996).

[26] E. E. Tyrtyshnikov, Incomplete cross approximation in the mosaic–skeleton method, *Computing*, V. 4, P. 367–380 (2000).

[27] E. E. Tyrtyshnikov, Kronecker-product approximations for some function-related matrices, *Linear Algebra Appl.*, V. 379, P. 423–437 (2004).

[28] E. E. Tyrtyshnikov, Tensor approximations of matrices generated by asymptotically smooth functions, *Sbornik: Mathematics* **194**, No. 5-6, 941–954 (2003).

[29] L. R. Tucker, Some mathematical notes on three-mode factor analysis, *Psychometrika*, V. 31, P. 279–311 (1966).