Крестовая аппроксимация тензоров и алгебры Клиффорда

Семинар “Алгебры Клиффорда и приложения”

06.04.2024

Катя Филимошина

Skoltech

Agenda

1. Skeleton decomposition and CUR approximation of matrices
2. Tensors: Basic Definitions
3. Skeleton decomposition and CUR approximation of tensors
4. Problem of choosing tubes for CUR approximation
5. Existing approaches
6. Proposed solution: Clifford scores for CUR using SVD
7. Implementation
8. Experiments on image completion task

01

Skeleton decomposition and CUR approximation of matrices

Presentation about CUR for matrices: https://personal.math.vt.edu/embree/cur_talk.pdf

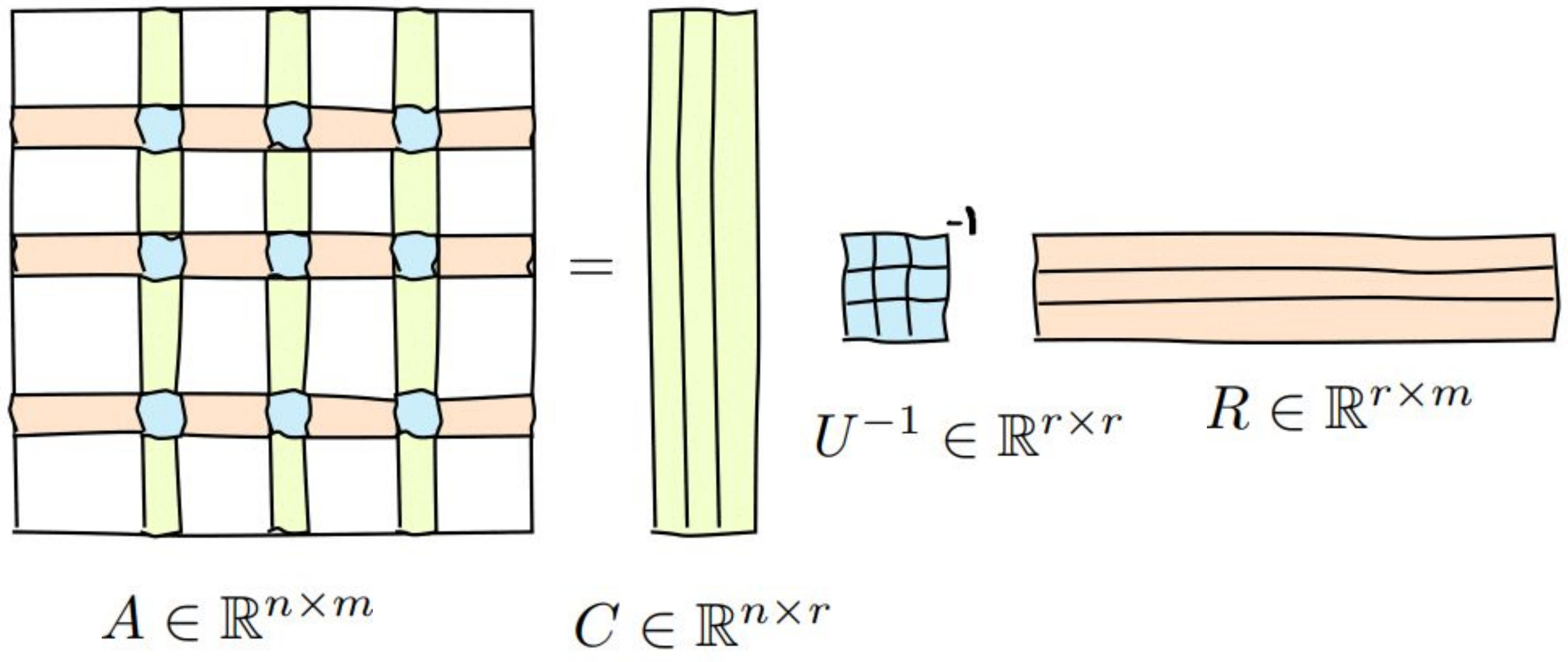
Skeleton decomposition for matrices

(Скелетное разложение)

Theorem 1.1 (Skeleton decomposition). *Suppose* $A \in \mathbb{R}^{n\times m}$, $\mathrm{rank}(A) = r$. *It can be written in a form*

$$A = CU^{-1}R, \tag{1.1}$$

where $C \in \mathbb{R}^{n\times r}$, $U^{-1} \in \mathbb{R}^{r\times r}$, *and* $R \in \mathbb{R}^{r\times m}$.



Skoltech

Skeleton decomposition for matrices

(Скелетное разложение)

Remark 1.2. Skeleton decomposition has another form:

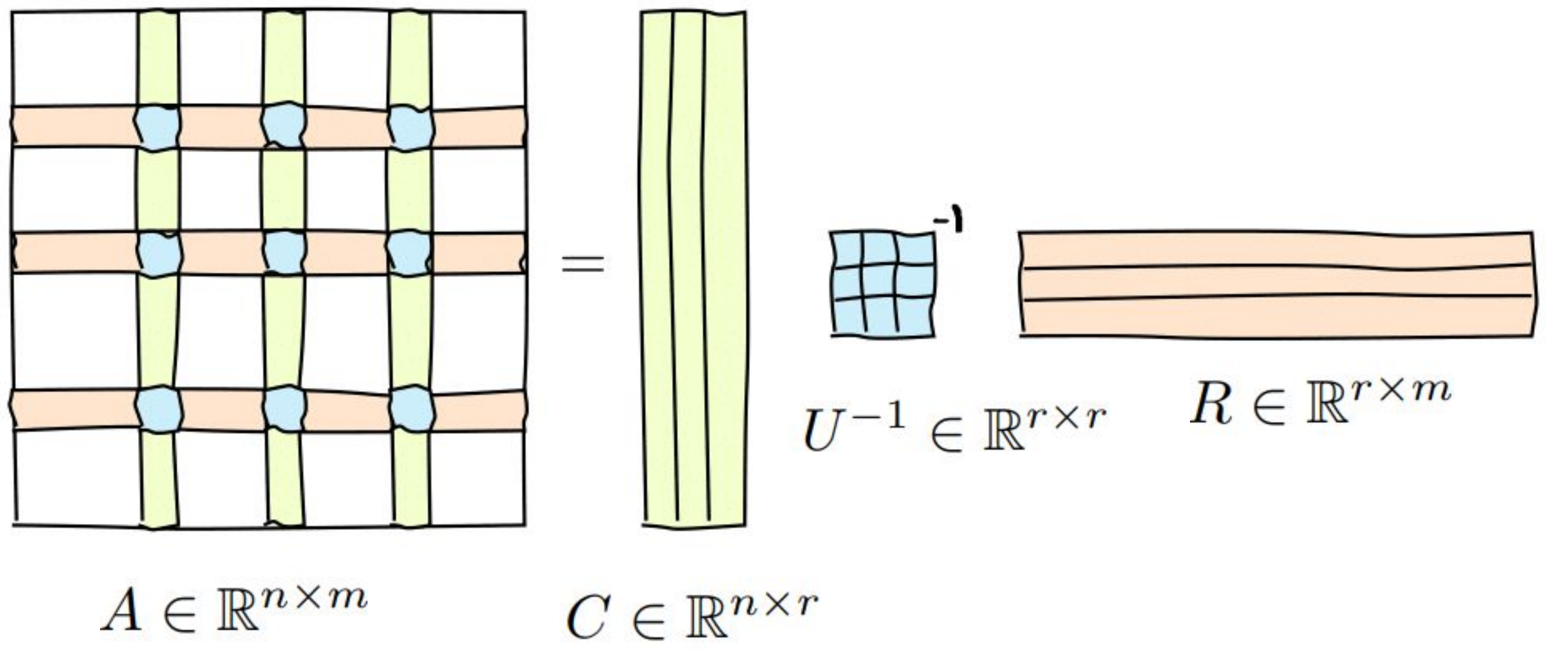
$$A = XY, \qquad X \in \mathbb{R}^{n\times r}, \quad Y \in \mathbb{R}^{r\times m}, \tag{1.2}$$

where U, V can be chosen, for example, as

$$X = CU^{-1}, \qquad Y = R \tag{1.3}$$

or

$$X = C, \qquad Y = U^{-1}R. \tag{1.4}$$



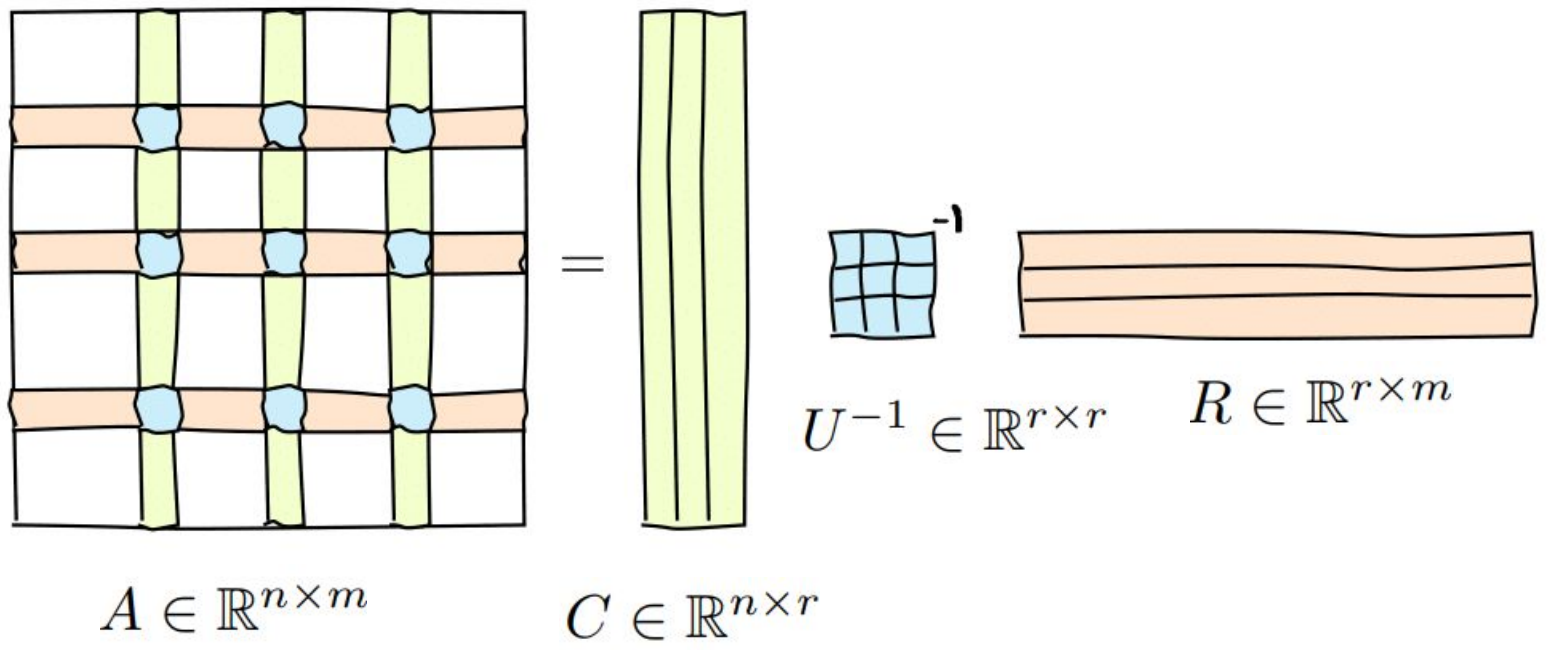
Skeleton decomposition for matrices

(Скелетное разложение)

Remark 1.3 (Existance). Skeleton decomposition exists for any $A \in \mathbb{R}^{n\times m}$.

Remark 1.4 (Uniqueness). Skeleton decomposition is not unique (except the case $n = m = r$):

1. We can choose any r linearly independent rows and columns for $A = CU^{-1}R$.
2. $A = UV = (US)(S^{-1}V) = U'V'$ for any S: $\det(S) \neq 0$.

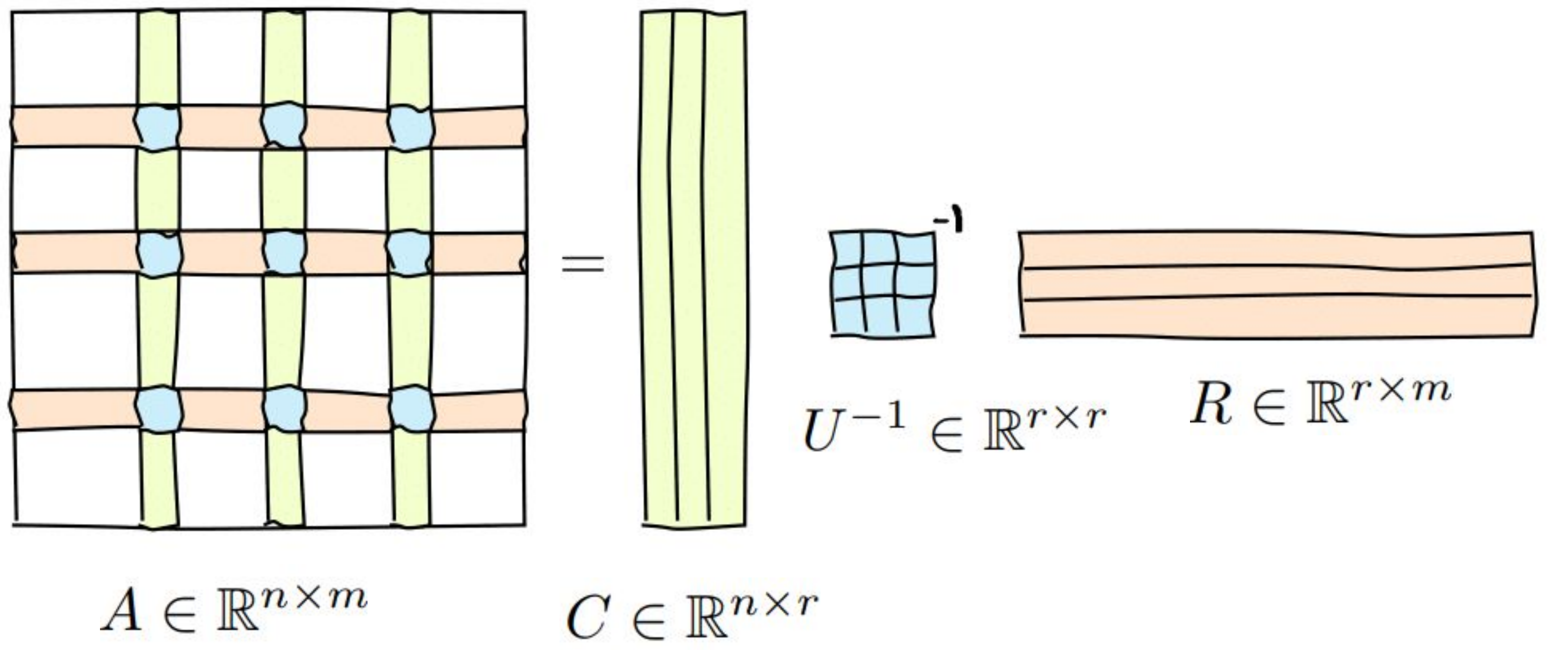


Skeleton decomposition for matrices

(Скелетное разложение)

Remark 1.5 (Use Cases). Main applications of Skeleton decomposition:

- **Data storage and compression:** Consider a matrix $A \in \mathbb{R}^{n \times m}$. Instead of storing nm parameters, we can decompose it as $A = UV$ and store $nr + rm = (n+m)r$ parameters. It is useful when $r < \frac{nm}{n+m}$.
- **Fast computation of matvec:** To compute a matrix-vector product Ax of $A \in \mathbb{R}^{n \times m}$ and $x \in \mathbb{R}^{m \times 1}$, we need to perform $O(mn)$ operations. Using $A = UV$, we need to compute $U(Vx)$, which takes $O(mr)$ operations for Vx and then $O(nr)$ operations for $U(Vx)$.



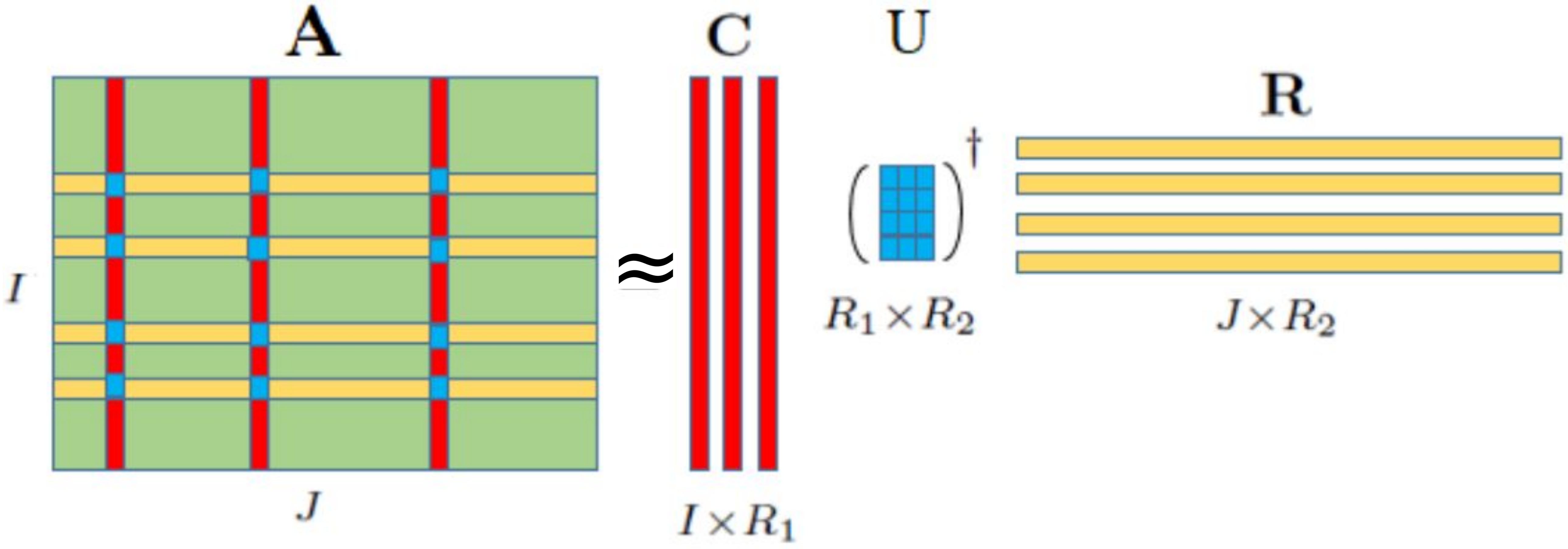
CUR approximation of matrices

(Крестовая аппроксимация)

Select R1 columns and R2 rows of A.

If R1 = R2 = rank(A), then we get exact Skeleton decomposition.

If R1 and R2 do not form a basis of column space and row space of A respectively, then the decomposition is not exact. It is called **CUR approximation**.

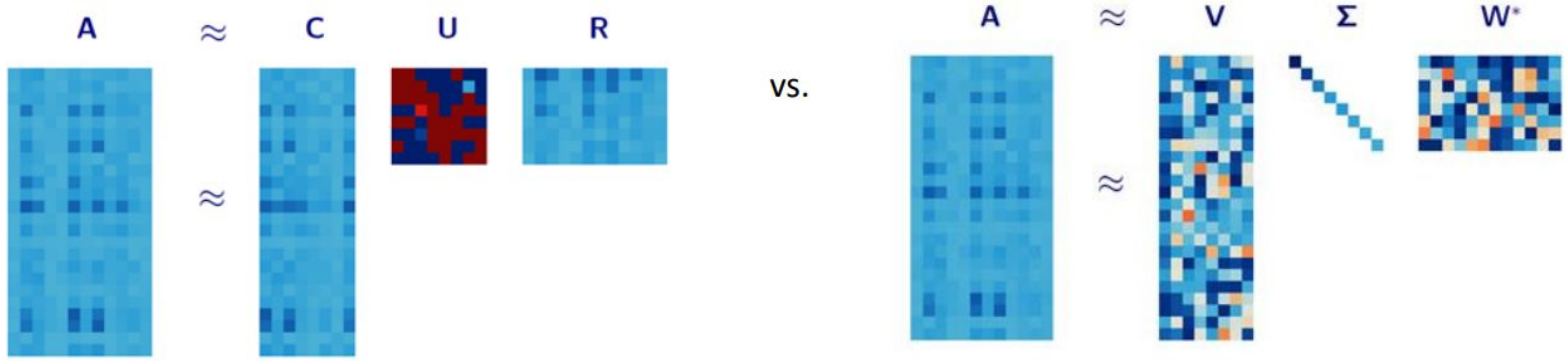


CUR approximation of matrices

(Крестовая аппроксимация)

CUR matrix approximation is used for low-rank matrix approximation.

CUR approximation is more representative of the data than the orthonormal singular vectors.



CUR

SVD

02

Tensors: Basic Definitions

Tensors

Definition. We call a **tensor** of order k a multi-dimensional matrix

$$A \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_k}. \tag{2.1}$$

Examples of tensors:

Tensor of order 0

$$a \in \mathbb{R}$$

Tensor of order 1

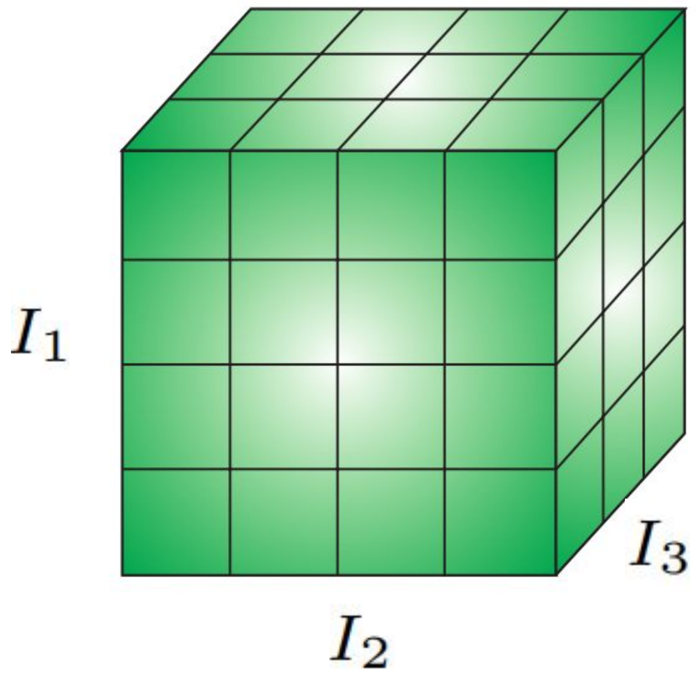
$$\begin{bmatrix} a_1 \\ \vdots \\ a_{I_1} \end{bmatrix} \in \mathbb{R}^{I_1}$$

Tensor of order 2

$$\begin{bmatrix} a_{11} & \cdots & a_{1I_2} \\ \vdots & \ddots & \vdots \\ a_{I_1 1} & \cdots & a_{I_1 I_2} \end{bmatrix}$$

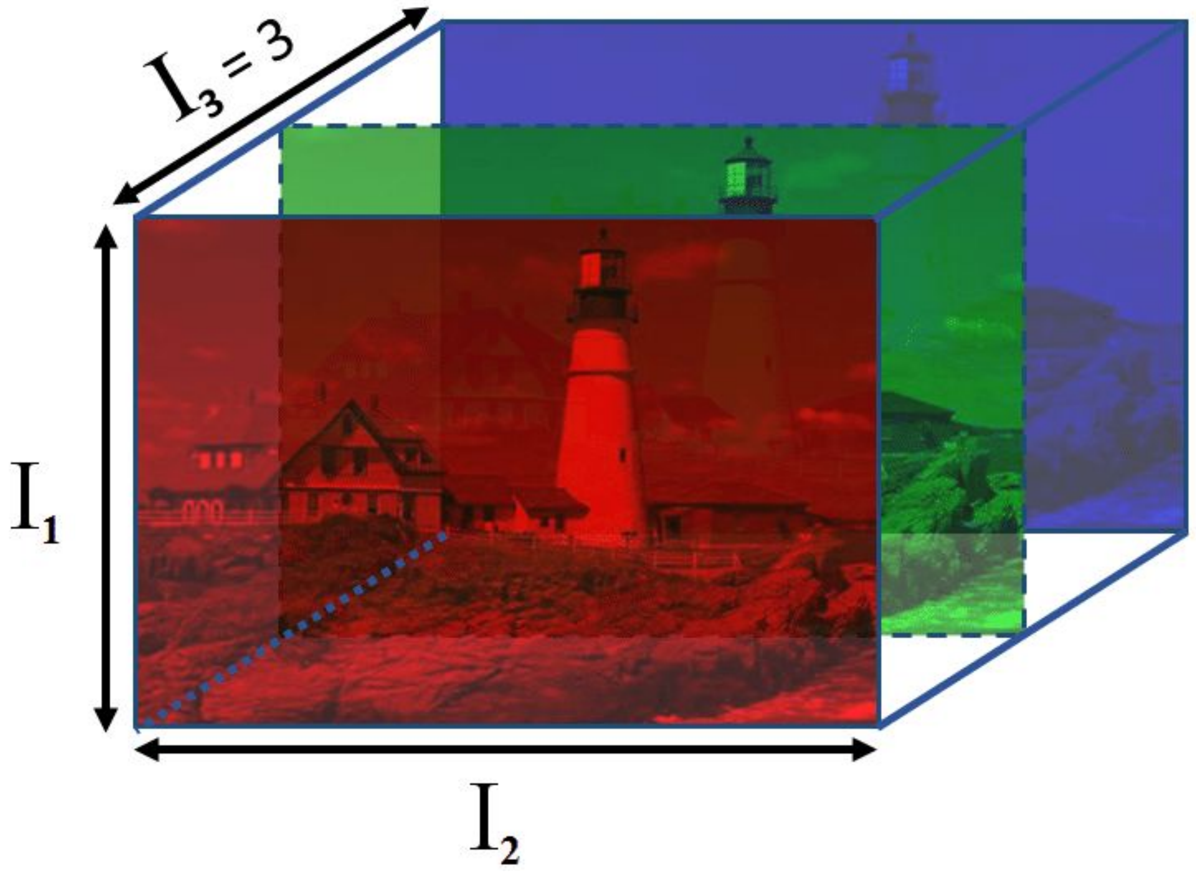
$$\in \mathbb{R}^{I_1 \times I_2}$$

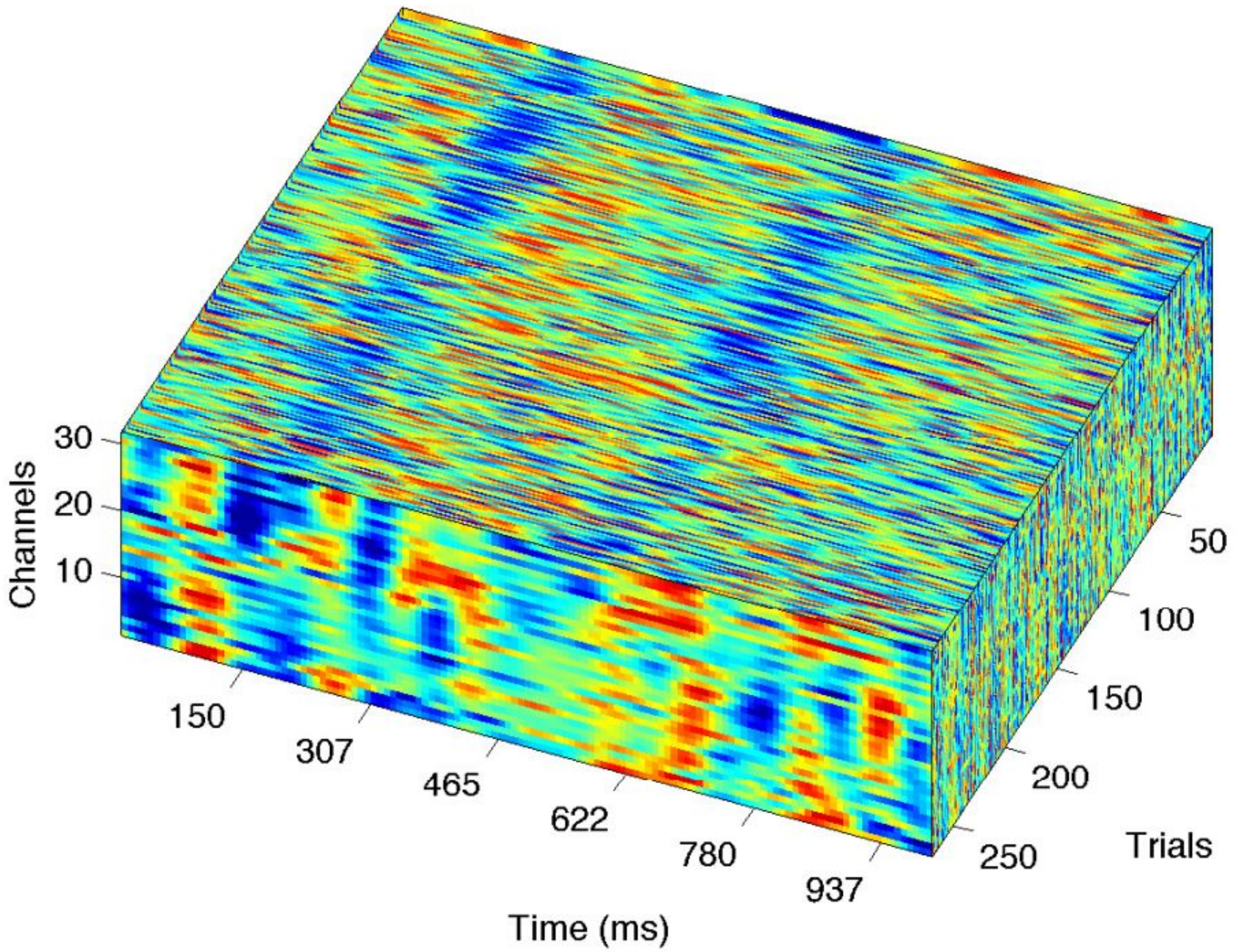
Tensor of order 3



$$\in \mathbb{R}^{I_1 \times I_2 \times I_3}$$

Tensors: More examples





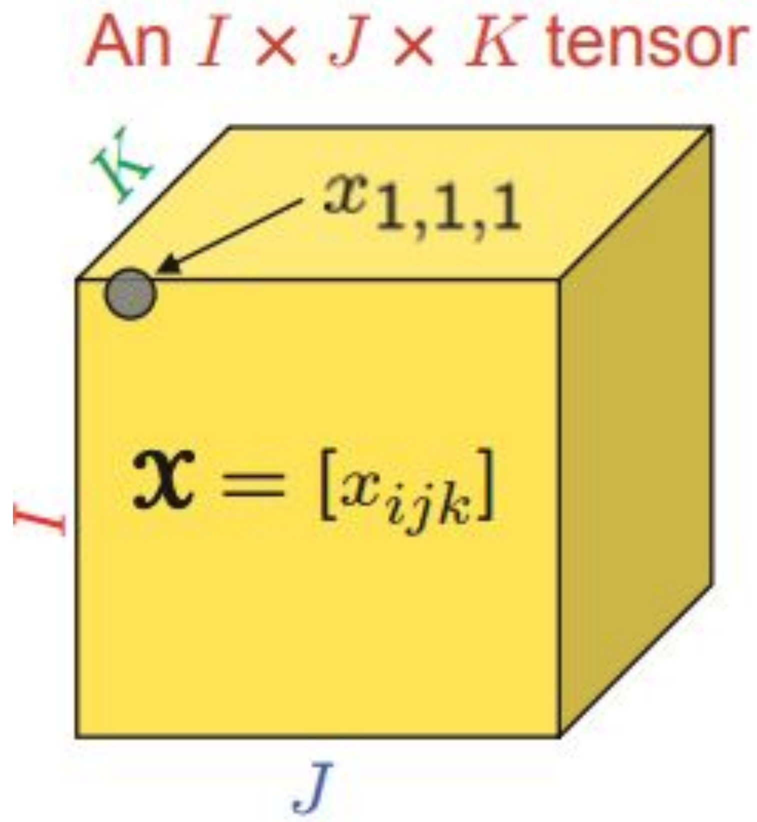
Images

EEG signals

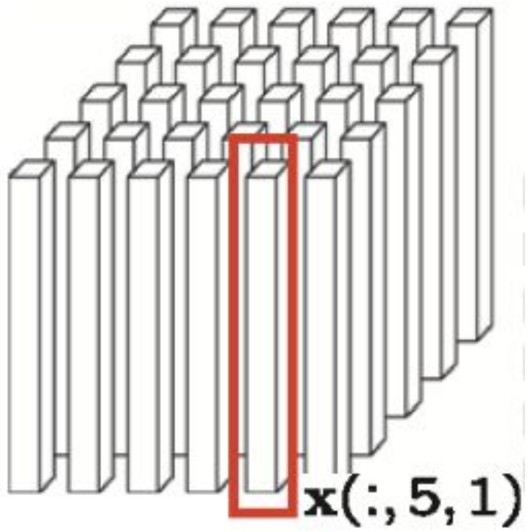
....

Skoltech

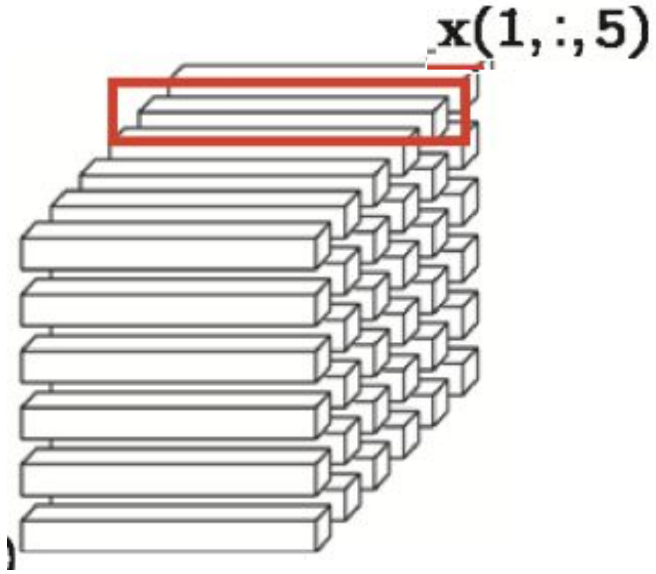
Fibers and Slices



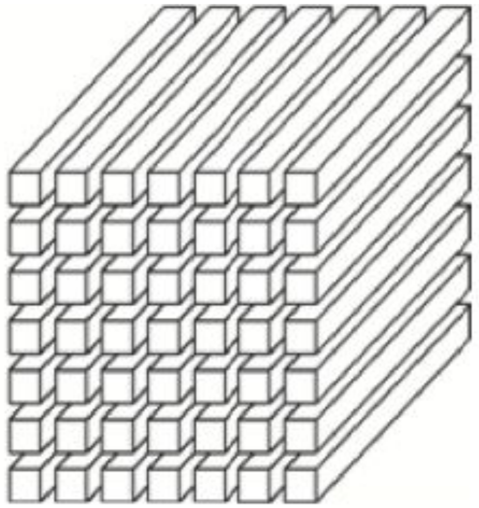
Fibers (tubes) of a 3rd order tensor:



Columns (mode 1)

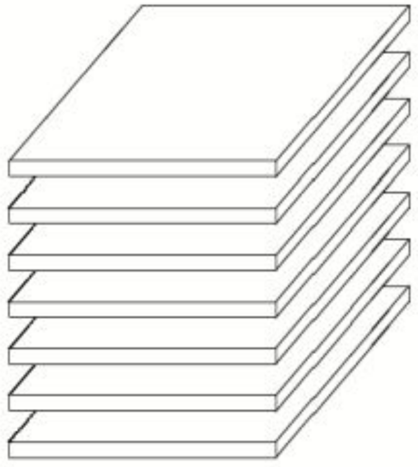


Rows (mode 2)

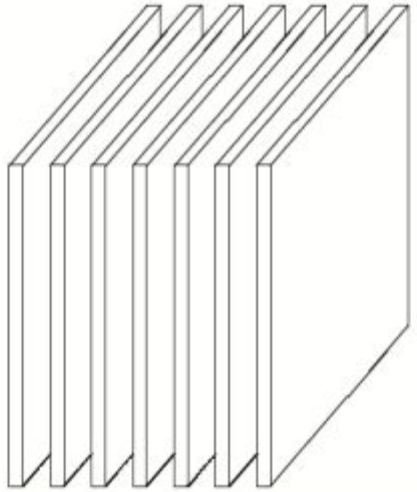


Tubes (mode 3)

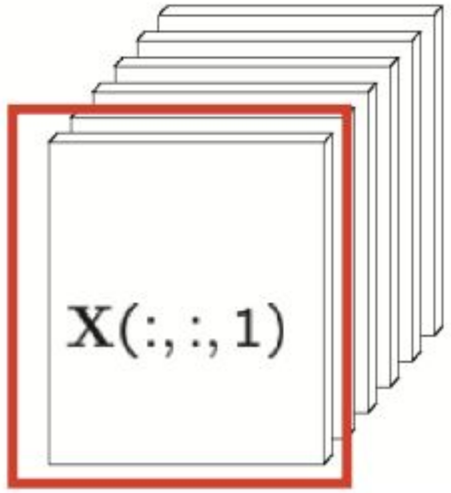
Slices of a 3rd order tensor:



Horizontal slices



Vertical slices



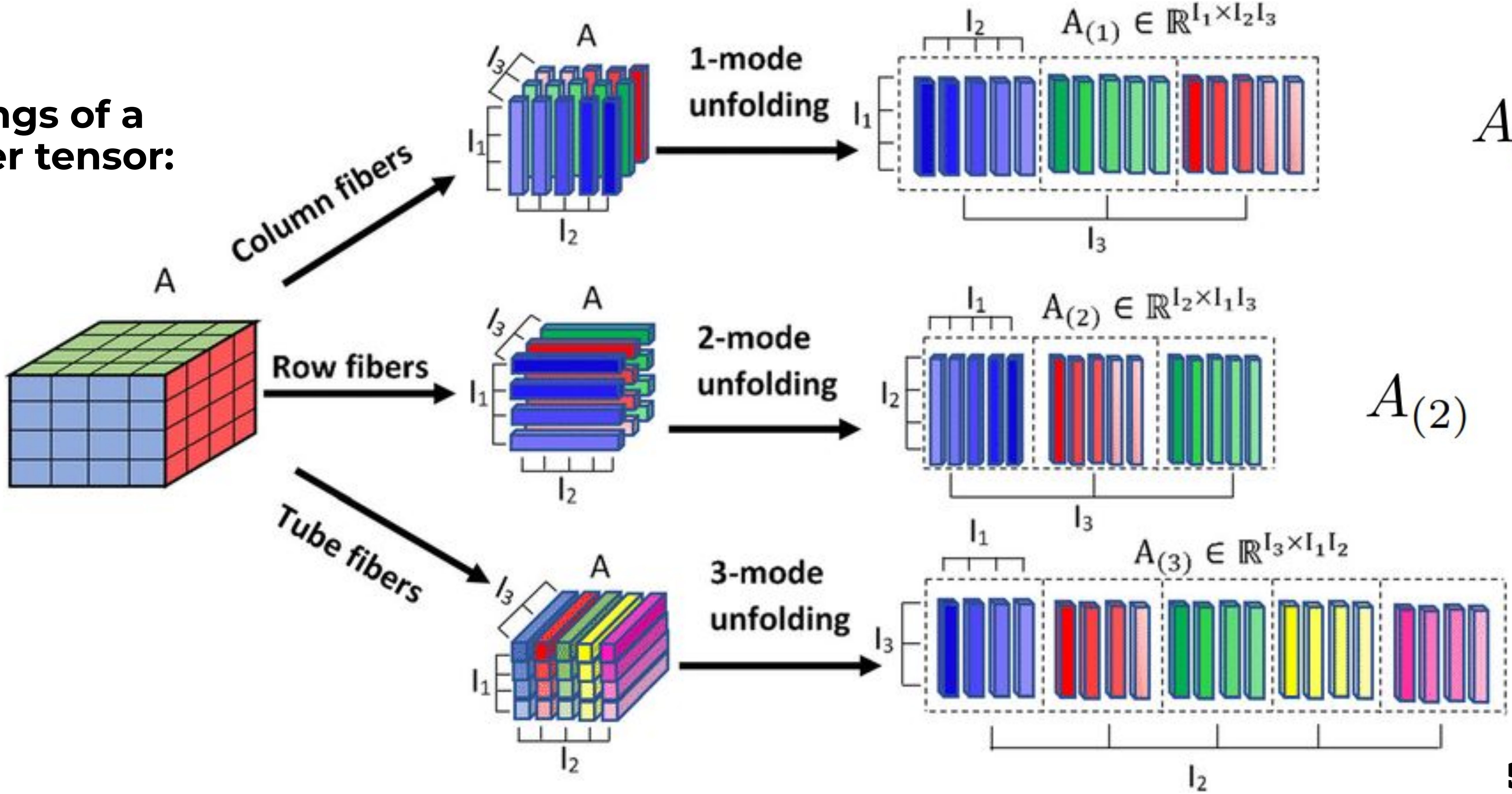
Frontal slices

Skoltech

Tensor Unfoldings

Definition. **Mode-n unfolding** of a tensor $A \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an operation $\mathbb{R}^{I_1 \times \cdots \times I_N} \to \mathbb{R}^{I_n \times (\Pi_{j \neq n} I_j)}$ that horizontally concatenates all tubes of A along mode n.

Unfoldings of a 3rd order tensor:



$A_{(1)}$

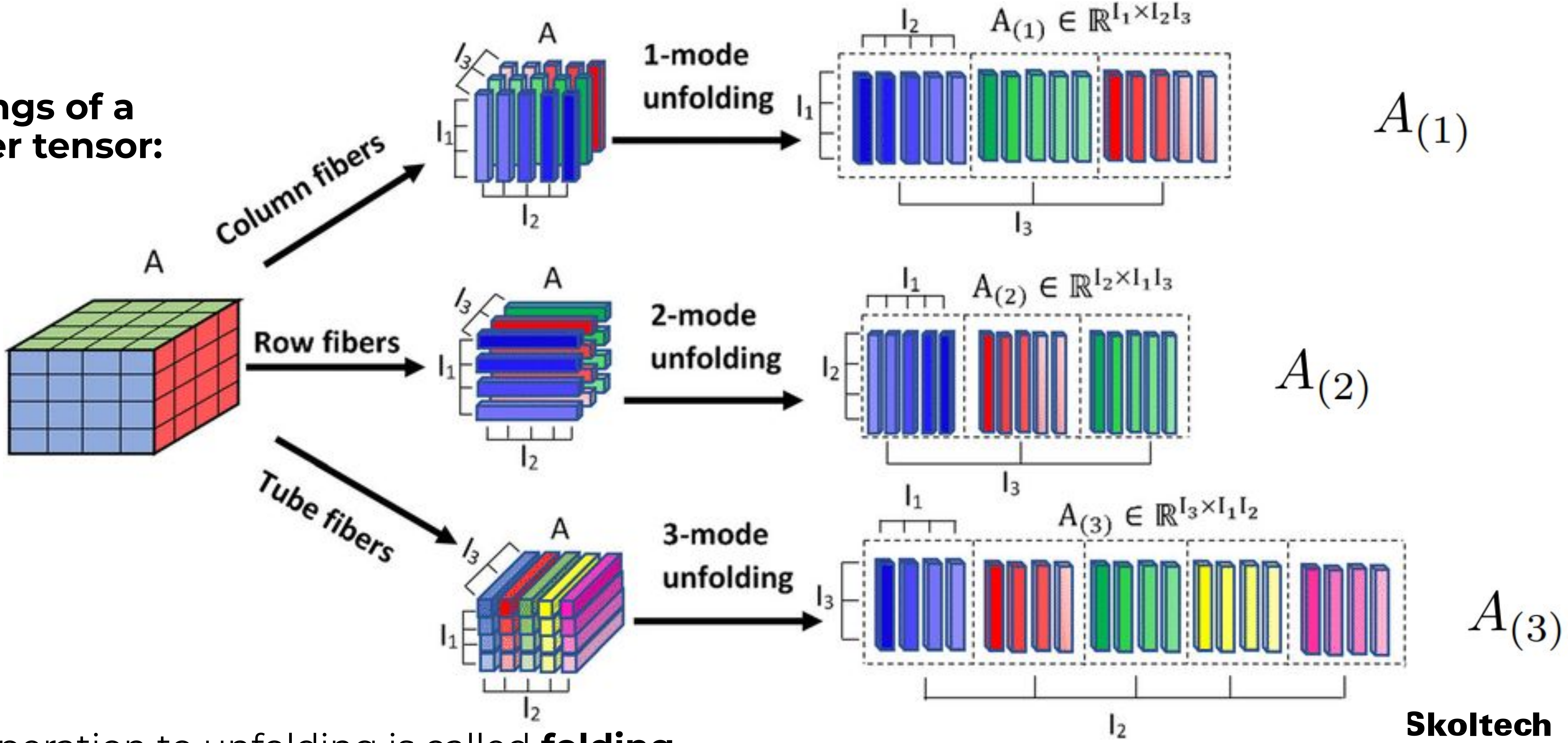
$A_{(2)}$

$A_{(3)}$

Tensor Unfoldings

Definition. Mode-n unfolding of a tensor $A \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an operation $\mathbb{R}^{I_1 \times \cdots \times I_N} \to \mathbb{R}^{I_n \times (\Pi_{j \neq n} I_j)}$ that horizontally concatenates all tubes of A along mode n.

Unfoldings of a 3rd order tensor:



Inverse operation to unfolding is called **folding**.

Tensor-matrix product

Definition. Consider a tensor $G \in \mathbb{R}^{I_1 \times \cdots \times I_k \times \cdots \times I_N}$ and a matrix $A \in \mathbb{R}^{J \times I_k}$. **Mode-$k$ tensor-matrix product** $\mathbb{R}^{I_1 \cdots \times I_k \times \cdots I_N} \times \mathbb{R}^{J \times I_k} \to \mathbb{R}^{I_1 \cdots \times J \times \cdots I_N}$ of G and A is denoted by

$$Y = G \times_k A \tag{2.7}$$

and defined as

$$Y_{(k)} = AG_{(k)}. \tag{2.8}$$

So, to calculate mode-k tensor-matrix product of G and A, we should find unfolding $G_{(k)}$, compute matrix product $AG_{(k)} \in \mathbb{R}^{J \times (\Pi_{l \neq k} I_l)}$, and make folding back.

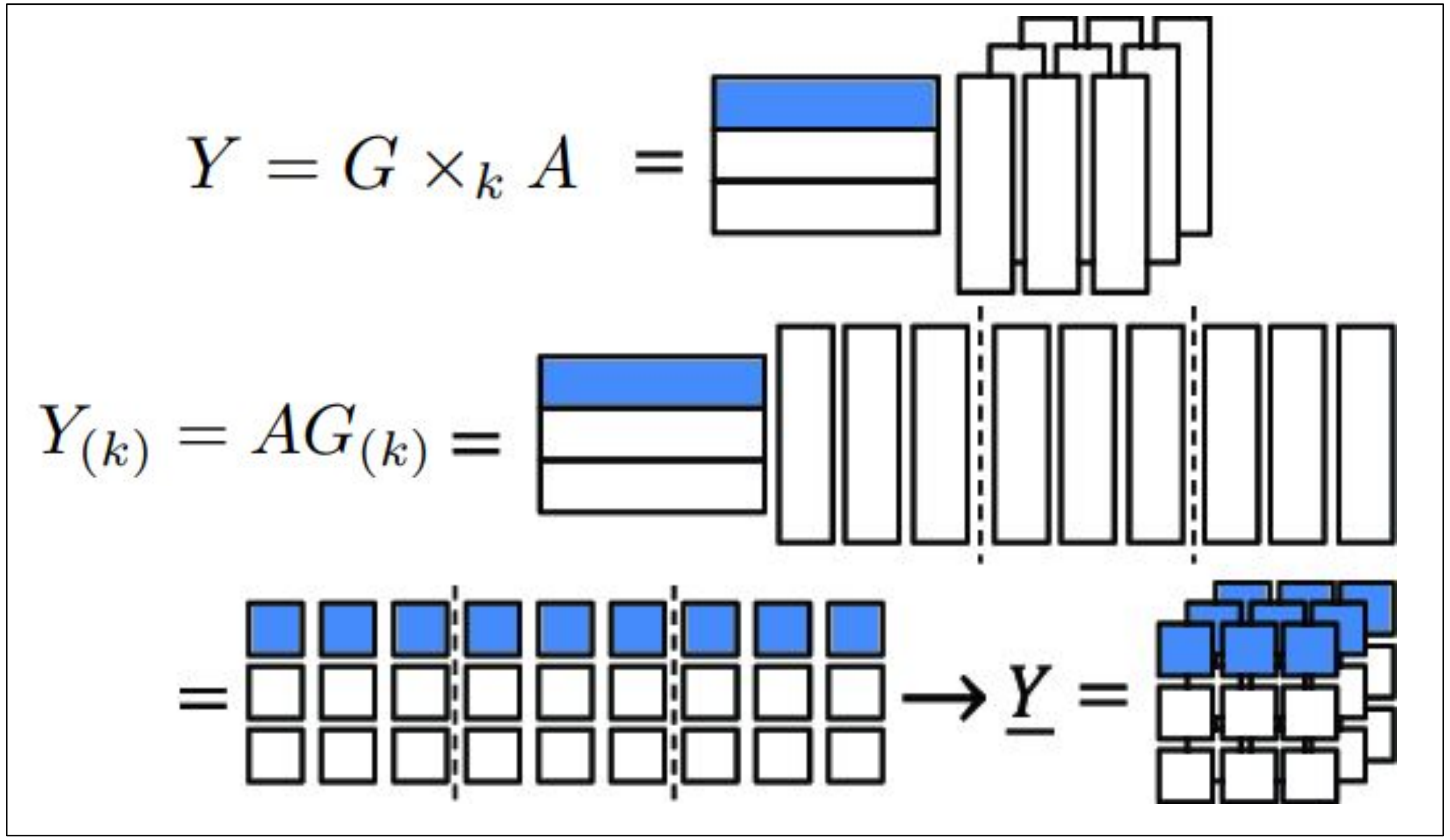
Tensor-matrix product

Definition. Consider a tensor $G \in \mathbb{R}^{I_1 \times \cdots \times I_k \times \cdots \times I_N}$ and a matrix $A \in \mathbb{R}^{J \times I_k}$. **Mode-$k$ tensor-matrix product** $\mathbb{R}^{I_1 \cdots \times I_k \times \cdots I_N} \times \mathbb{R}^{J \times I_k} \to \mathbb{R}^{I_1 \cdots \times J \times \cdots I_N}$ of G and A is denoted by

$$Y = G \times_k A$$

and defined as

$$Y_{(k)} = AG_{(k)}.$$



So, to calculate mode-k tensor-matrix product of G and A, we should find unfolding $G_{(k)}$, compute matrix product $AG_{(k)} \in \mathbb{R}^{J \times (\Pi_{l \neq k} I_l)}$, and make folding back.

03

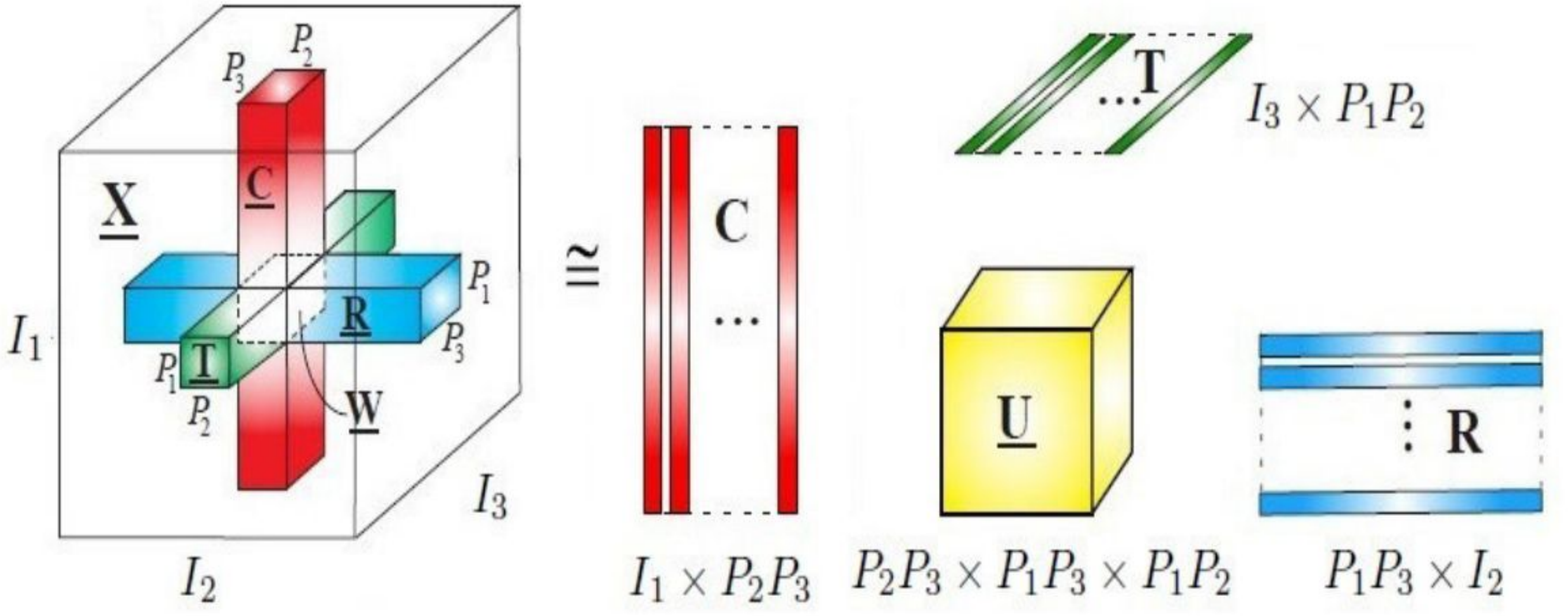
Skeleton decomposition and CUR approximation of tensors

Presentation about CUR for matrices: https://personal.math.vt.edu/embree/cur_talk.pdf

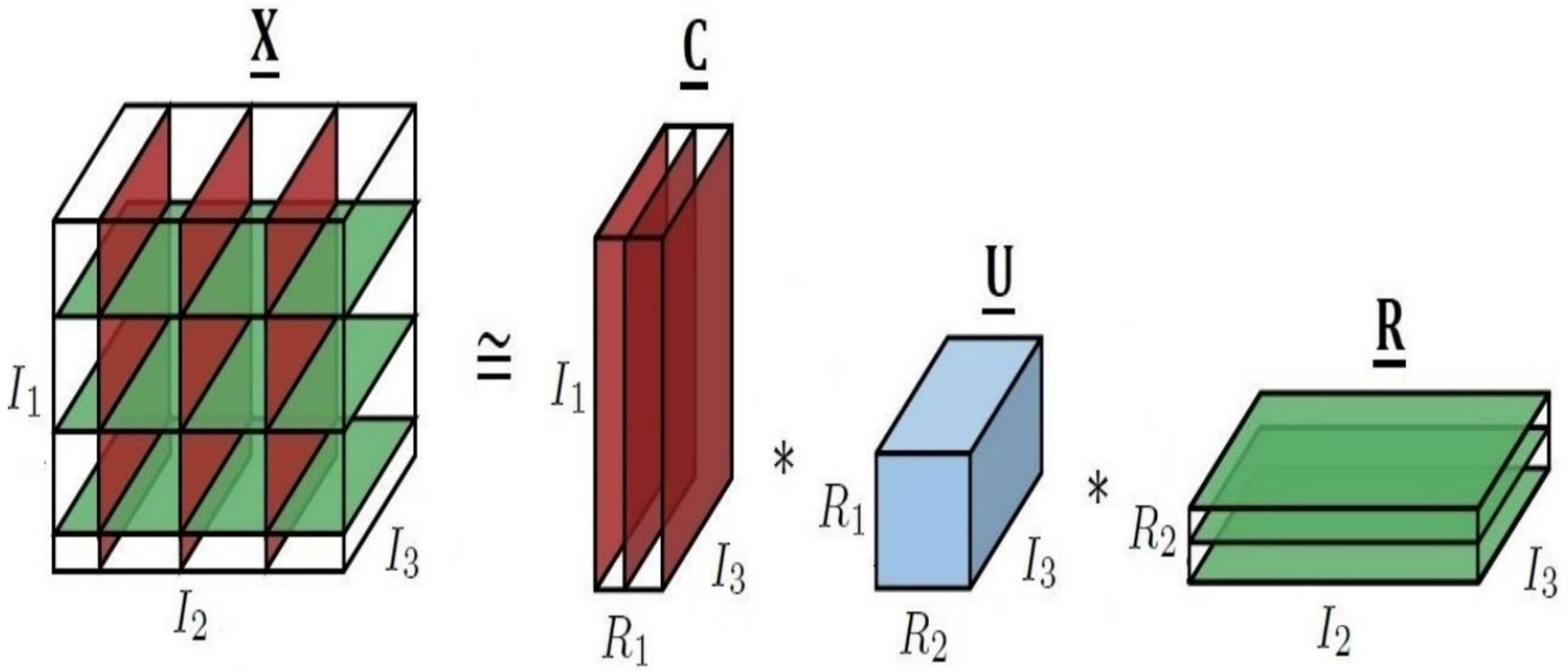
CUR of tensors: 3 types

S. Ahmadi-Asl et al., "**Cross Tensor Approximation Methods for Compression and Dimensionality Reduction**," in IEEE, vol. 9, 2021

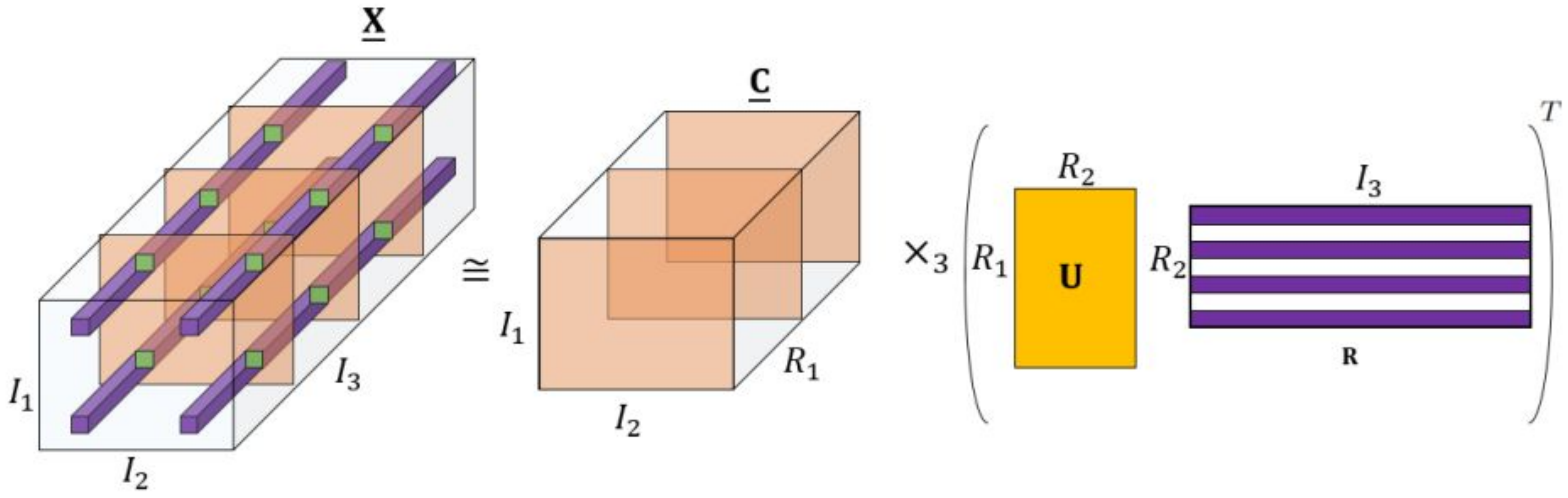
1. Tubes selection



3. Slice/Slice selection



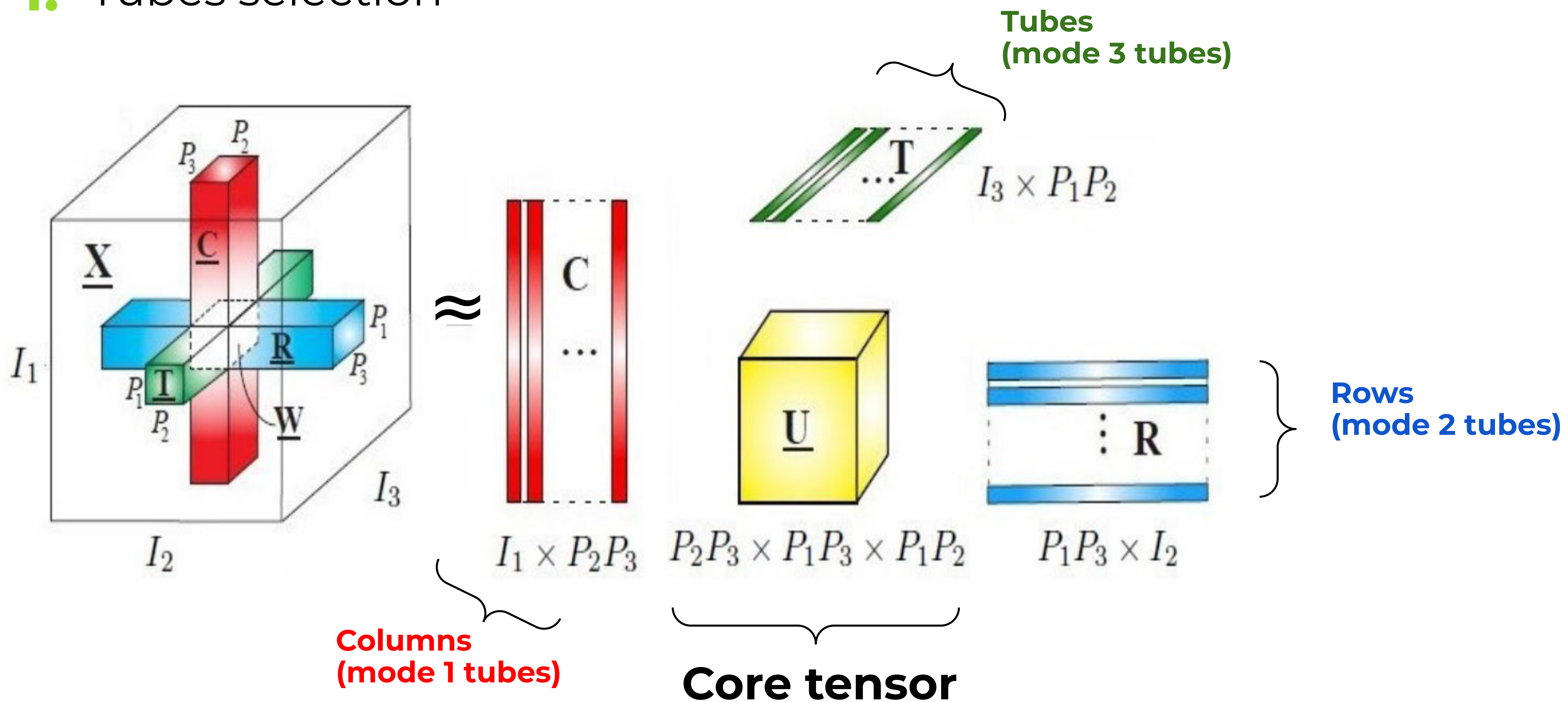
2. Tube/Slice selection



Skoltech

CUR of tensors

1. Tubes selection



Skoltech

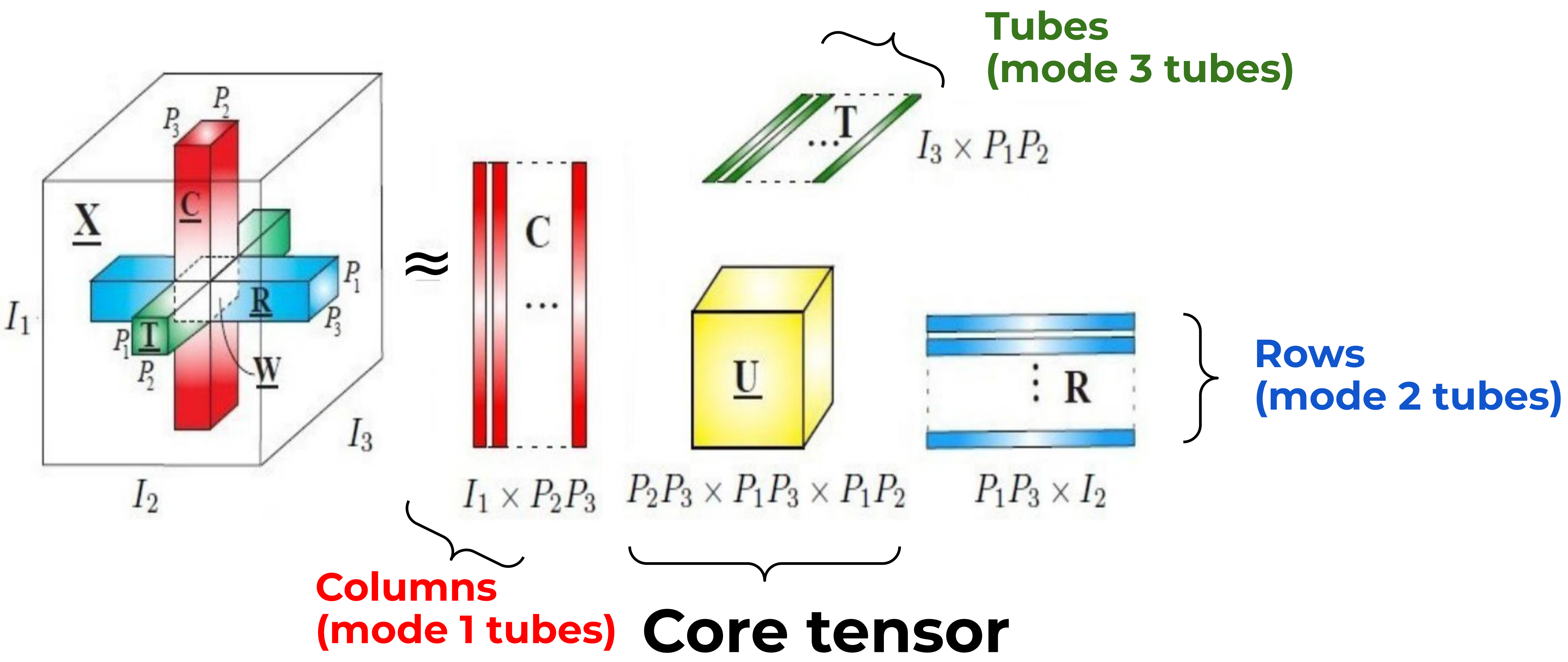
CUR of tensors

We can get an approximation of the initial tensor X as the tensor-matrix product of the three matrices C, T, R and a core tensor U:

$$X \approx U \times_1 C \times_2 R \times_3 T. \tag{2.9}$$

This product can be also denoted as

$$X \approx [\![U; C, R, T]\!]. \tag{2.10}$$



Skeleton decomposition for tensors

Theorem 2.1. *Suppose we know that a tensor $Y \in \mathbb{R}^{I \times J \times K}$ can be exactly represeneted as*

$$Y = [\![G; A_1, A_2, A_3]\!], \qquad (2.11)$$

i.e. there exist a tensor $G \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ and matrices $A_1 \in \mathbb{R}^{I \times R_1}$, $A_2 \in \mathbb{R}^{J \times R_2}$, and $A_3 \in \mathbb{R}^{K \times R_3}$. Then

1. *There exist such subsets for each dimension*

$$\mathcal{I} = \{i_1, \dots, i_{P_1}\}, \quad \mathcal{J} = \{j_1, \dots, j_{P_2}\}, \quad \mathcal{K} = \{k_1, \dots, k_{P_3}\}, \qquad (2.12)$$

($P_1 \geq R_1$, $P_2 \geq R_2$, $P_3 \geq R_3$), that the unfolding matrices of the intersection subtensor $W = Y(\mathcal{I}, \mathcal{J}, \mathcal{K})$ have ranks R_1, R_2, and R_3 respectively:

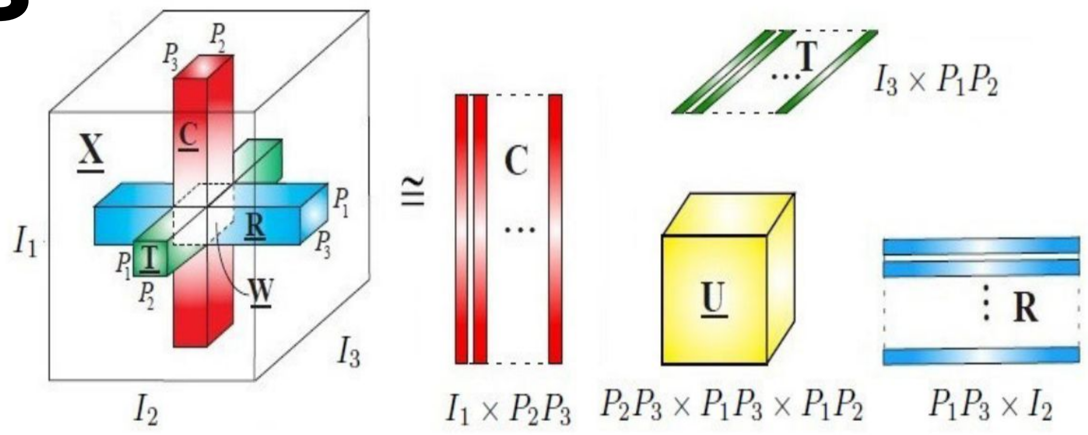
$$\mathrm{rank}(W_{(1)}) = R_1, \quad \mathrm{rank}(W_{(2)}) = R_2, \quad \mathrm{rank}(W_{(3)}) = R_3. \qquad (2.13)$$

2. *The following exact decomposition can be obtained:*

$$Y = [\![U; C_1, C_2, C_3]\!], \quad \text{with} \quad U = [\![W; W_{(1)}^{\dagger}, W_{(2)}^{\dagger}, W_{(3)}^{\dagger}]\!], \qquad (2.14)$$

where matrices $C_1 \in \mathbb{R}^{I \times P_2 P_3}$, $C_2 \in \mathbb{R}^{J \times P_1 P_3}$, and $C_3 \in \mathbb{R}^{K \times P_1 P_2}$ are defined as

$$C_1 = Y_{(1)}(:, \mathcal{J} \times \mathcal{K}), \quad C_2 = Y_{(2)}(:, \mathcal{I} \times \mathcal{K}), \quad C_3 = Y_{(3)}(:, \mathcal{I} \times \mathcal{J}). \qquad (2.15)$$



Generalizing the column–row matrix decomposition to multi-way arrays

Cesar F. Caiafa [*,1], Andrzej Cichocki [2,3]

LABSP, RIKEN Brain Science Institute, Wako, Saitama 351-0198, Japan

https://core.ac.uk/download/pdf/82355805.pdf

FSTD algorithm

Applications of CUR for tensors

S. Ahmadi-Asl et al., "**Cross Tensor Approximation Methods for Compression and Dimensionality Reduction**," in IEEE, vol. 9, 2021

CUR can be used in applications where the low-rank tensor approximation is required:

01. Tensor data compression

To reduce number of parameters of a tensor with a given approximation error.

02. Tensor Completion

To estimate missing or uncertain elements of a tensor (for image inpainting, video completion, time series completion, etc.)

03. Denoising

To denoise a data tensor.

Cross Tensor Approximation Methods for Compression and Dimensionality Reduction

SALMAN AHMADI-ASL[1], CESAR F. CAIAFA[2], ANDRZEJ CICHOCKI[1,3] (Life Fellow, IEEE), ANH HUY PHAN[1], (Member, IEEE), TOSHIHISA TANAKA[4], (Senior Member, IEEE), IVAN OSELEDETS[1], AND JUN WANG[1]

[1]CDISE, Skolkovo Institute of Science and Technology (SKOLTECH), 121205 Moscow, Russia
[2]Instituto Argentino de Radioastronomía—CCT La Plata, CONICET/CIC-PBA/UNLP, Villa Elisa 1894, Argentina
[3]RIKEN Center for Advanced Intelligence Project (AIP), Tokyo 103-0027, Japan
[4]Department of Electrical and Electronic Engineering, Tokyo University of Agriculture and Technology, Tokyo 183-8509, Japan

04

Problem of CUR approximation

Skoltech

Problem and its relevance

Approximation accuracy of CUR approximation for matrices / tensors **significantly depends on chosen** rows and columns / **tubes and slices.**

How to Find a Good Submatrix*

S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin

Institute of Numerical Mathematics of Russian Academy of Sciences, Gubkina 8, 119333 Moscow, Russia
{sergei,ivan,draug,tee,kolya}@bach.inm.ras.ru

Generalizing the column–row matrix decomposition to multi-way arrays

Cesar F. Caiafa *,1, Andrzej Cichocki 2,3

LABSP, RIKEN Brain Science Institute, Wako, Saitama 351-0198, Japan

....

- Column pivoted QR factorizations [Stewart 1999], [Voronin and Martinsson 2015]
  cf. Rank Revealing QR factorizations [Gu, Eisenstat 1996]
- Volume optimization [Goreinov, Tyrtyshnikov, Zamarashkin 1997], . . . , [Goreinov, Oseledets, Savostyanov, Tyrtyshnikov, Zamarashkin 2010], [Thurau, Kersting, Bauckhage 2012]
- Uniform sampling of columns e.g., [Chiu, Demanet 2012]
- Leverage scores (norms of rows of singular vector matrices) [Drineas, Mahoney, Muthukrishnan 2008], [Mahoney, Drineas 2009], . . . , [Boutsidis, Woodruff 2014]
- Empirical Interpolation approaches [Sorensen & E.], Q-DEIM method of [Drmač, Gugercin 2015]

05

Existing approaches

Some of the existing solutions for rows/columns choice

Uniform distribution sampling

$$p_j = \frac{1}{J},\ j = 1, 2, \ldots, J$$

$$p_i = \frac{1}{I},\ i = 1, 2, \ldots, I$$

Length-squared distribution sampling

$$p_j = \frac{\|\mathbf{A}(:,j)\|_2^2}{\|\mathbf{A}\|_F^2},\ j = 1, 2, \ldots, J$$

$$p_i = \frac{\|\mathbf{A}(i,:)\|_2^2}{\|\mathbf{A}\|_F^2},\ i = 1, 2, \ldots, I$$

Discrete Empirical Interpolation Method (DEIM)

D. C. Sorensen, M. Embree. ***A DEIM Induced CUR Factorization.*** *https://arxiv.org/abs/1407.5516*

Leverage scores sampling

Suppose we have $\mathbf{A} = \mathbf{USV}^T$, $\mathbf{U} \in \mathbb{R}^{I \times R}$ and $\mathbf{V} \in \mathbb{R}^{J \times R}$.
To rank the importance of the rows, take the 2-norm of each row of $\mathbf{U}$:

Row leverage score$= l_{R,i} = \|\mathbf{U}(i,:)\|_2^2$ $\qquad p_i = \frac{l_{R,i}}{R},\ i = 1, 2, \ldots, I.$

Column leverage score$= \hat{l}_{R,j} = \|\mathbf{V}(j,:)\|_2^2$ $\qquad p_j = \frac{\hat{l}_{R,j}}{R},\ j = 1, 2, \ldots, J.$

Volume optimization, Cross 2D

...

Some of the existing solutions for tubes choice

Uniform distribution sampling

$$p_j = \frac{1}{J},\ j = 1, 2, \ldots, J$$

$$p_i = \frac{1}{I},\ i = 1, 2, \ldots, I$$

Length-squared distribution sampling

$$p_i = \frac{\|\mathbf{X}(:,:,i_3)\|_F^2}{\|\mathbf{X}\|_F^2},\ i_3 = 1, 2, \ldots, I_3,$$

$$q_j = \frac{\mathbf{X}(j_1, j_2, :)}{\|\mathbf{X}\|_F^2},\ j_1, j_2 \in J_1, J_2$$

Michel Mahoney et.al, ***Tensor-CUR decompositions for tensor-based data****, SIAM Journal on Matrix Analysis and Applications, 2008.*

Cross 3D

I. V. OSELEDETS, D. V. SAVOSTIANOV, AND E. E. TYRTYSHNIKOV. ***TUCKER DIMENSIONALITY REDUCTION OF THREE-DIMENSIONAL ARRAYS IN LINEAR TIME****, 2008*

Adaptive algorithm

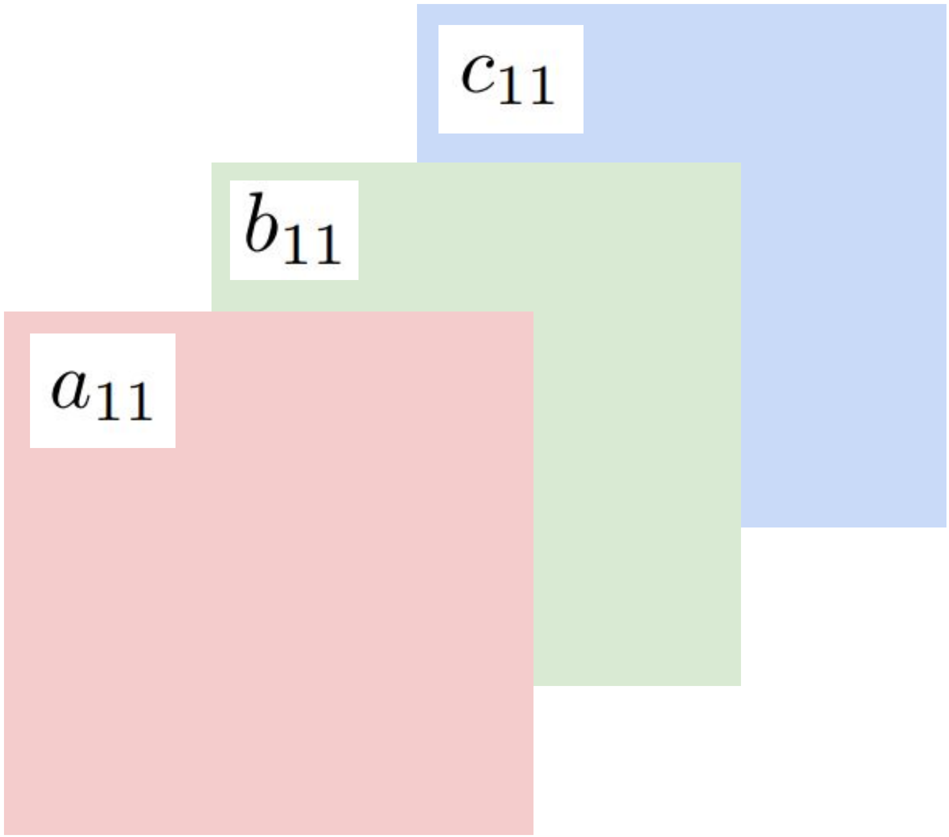
Cesar F. Caiafa, Andrzej Cichocki, ***Generalizing the column–row matrix decomposition to multi-way arrays****, Linear Algebra and its Applications, 2010*

06

Proposed solution: Clifford scores for tensors CUR

Images and Quaternions

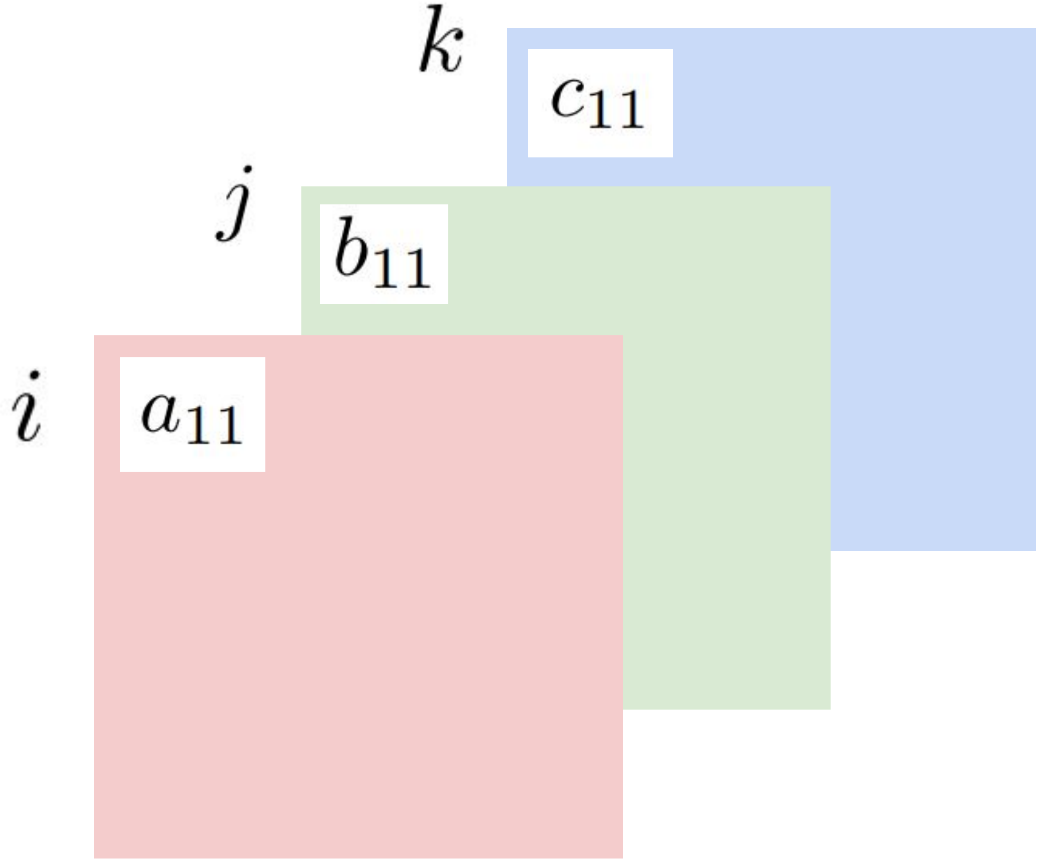
Image as a tensor with coefficients in $\mathbb{R}$



$$\in \mathbb{R}^{I_1 \times I_2 \times 3}$$

Images and Quaternions

Image as a tensor with coefficients in $\mathbb{R}$

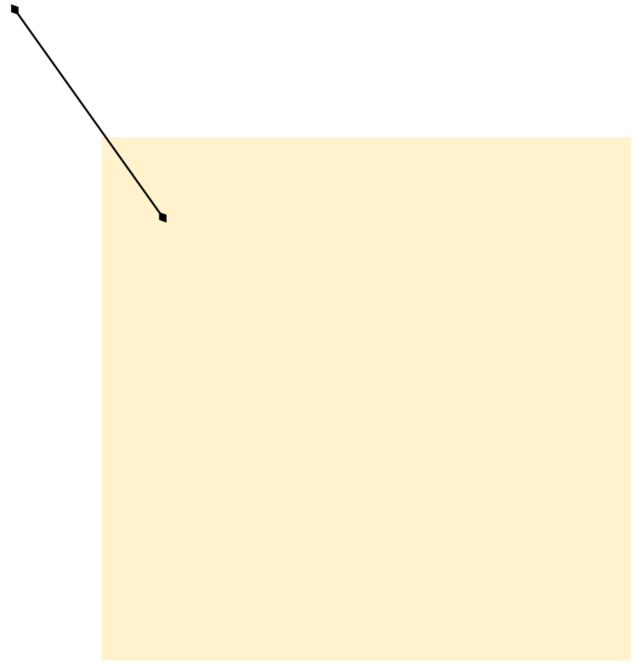


$$\in \mathbb{R}^{I_1 \times I_2 \times 3}$$

$$\mapsto$$

Image as a matrix with coefficients in $\mathbb{H} \cong \; C\ell_{0,2}(\mathbb{R})$

$$q_{11} = a_{11}\, i + b_{11}\, j + c_{11}\, k$$

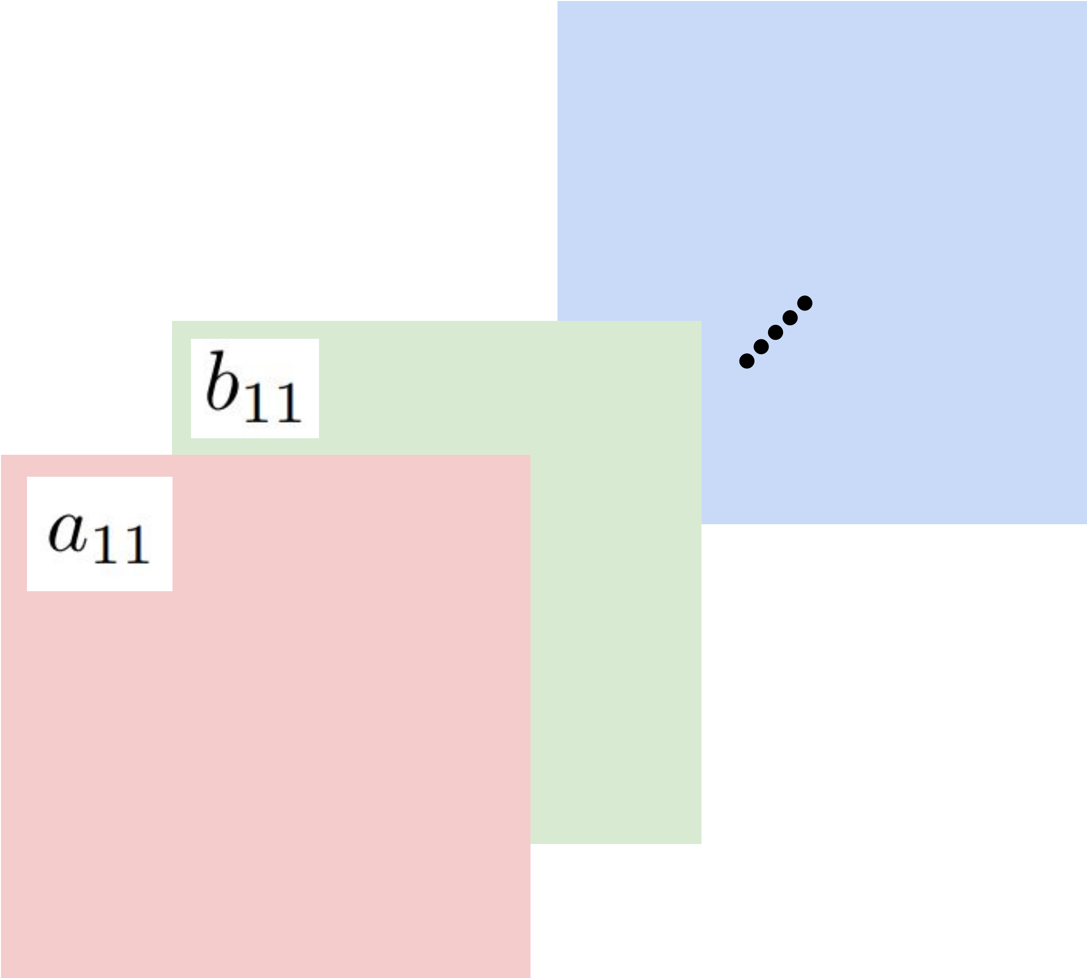


$$\in \mathbb{H}^{I_1 \times I_2}$$

Tensors as Clifford algebra matrices

Tensor with coefficients in $\mathbb{R}$

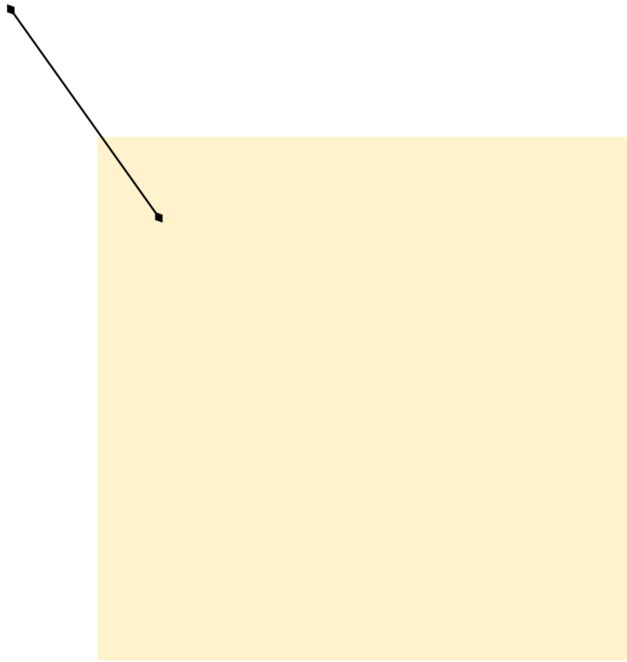
Matrix with coefficients in $C\ell_{p,q}(\mathbb{R})$



$\in \mathbb{R}^{I_1 \times I_2 \times N}$

$\mapsto$

$q_{11} = a_{11}\, e_1 + b_{11} e_2 + \cdots$



$\in [C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$

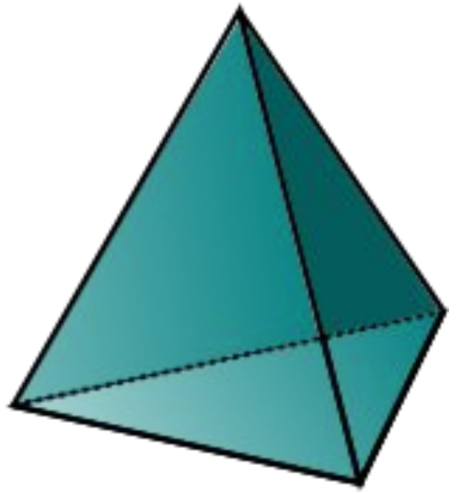
Tensors as Clifford algebra matrices

For which tasks it might be useful to apply such an approach?

The most natural applications:

- Images: select basis element of $C\ell_{p,q}$ for each channel.
- Data with coordinates of some points or vectors.

Example:

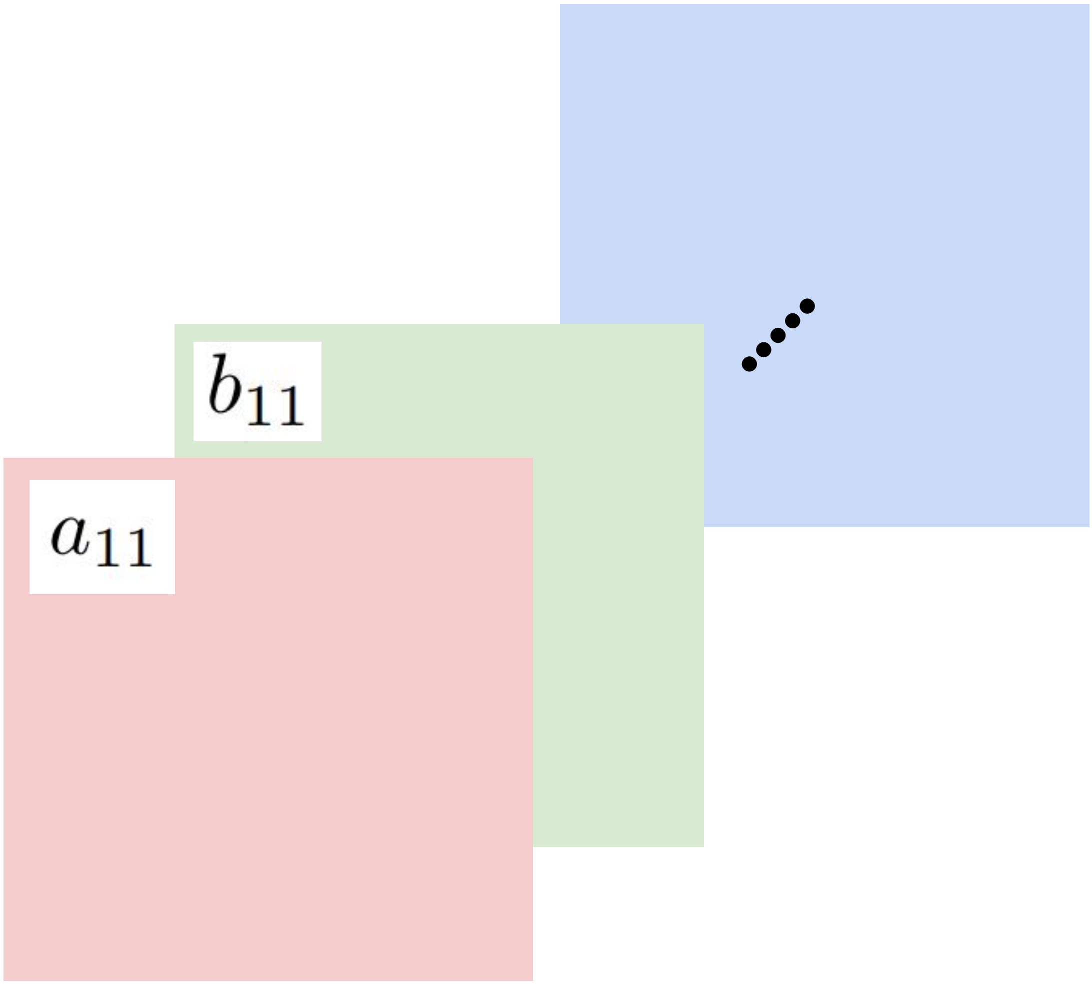


$$\mathbb{R}^{\text{num. of tethraedra} \times 3 \times 3}$$

Skoltech

Proposed solution: Clifford Scores for CUR

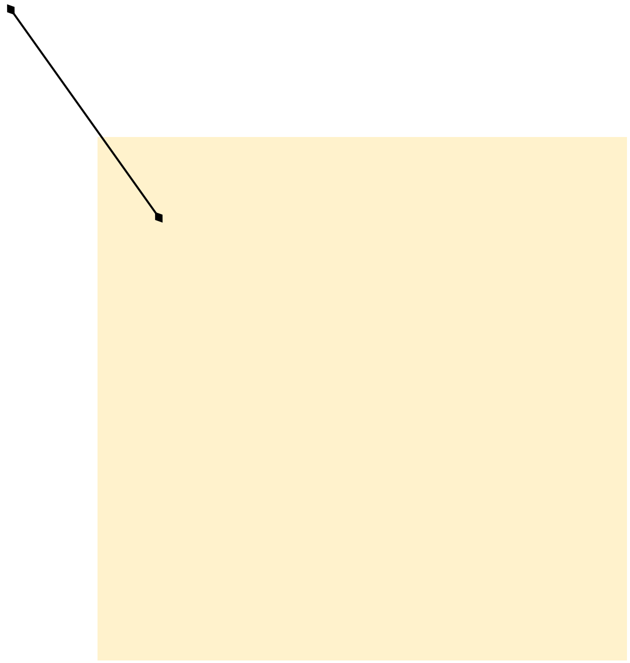
01. **Tensor in** $\mathbb{R}^{I_1 \times I_2 \times N}$ $\mapsto$ **Matrix in** $[C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$



$\in \mathbb{R}^{I_1 \times I_2 \times N}$

$\mapsto$

$$q_{11} = a_{11}\, e_1 + b_{11}\, e_2 + \cdots$$



$\in [C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$

Proposed solution: Clifford Scores for CUR

02. SVD of the matrix in $[C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$

$$q_{11} = b_{11}e_1 + c_{11}e_2 + \cdots$$

$$\in [C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2} \quad \mapsto \quad U \; \Sigma \; V^T$$

U: Unitary in $[C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$

Σ: Real (?)

V^T: Unitary in $[C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$

SVD for Quaternion Matrices

Quaternion Singular Value Decomposition based on Bidiagonalization to a Real Matrix using Quaternion Householder Transformations.

S. J. Sangwine[†§] N. Le Bihan[‡]

October 16, 2018

Abstract

We present a practical and efficient means to compute the singular value decomposition (SVD) of a quaternion matrix **A** based on bidiagonalization of **A** to a *real* bidiagonal matrix **B** using quaternionic Householder transformations. Computation of the SVD of **B** using an existing subroutine library such as LAPACK provides the singular values of **A**. The singular vectors of **A** are obtained trivially from the product of the Householder transformations and the real singular vectors of **B**. We show in the paper that left and right quaternionic Householder transformations are different because of the non-commutative multiplication of quaternions and we present formulae for computing the Householder vector and matrix in each case.

https://arxiv.org/pdf/math/0603251.pdf

SVD for Quaternion Matrices

Algorithm 2: Quaternion singular value decomposition

`Input  :` $\mathbf{A} \in \mathbb{H}^{r \times c}$
`Output :` $\mathbf{U} \in \mathbb{H}^{r \times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r \times c}$, $\mathbf{V} \in \mathbb{H}^{c \times c}$

Bidiagonalize $\mathbf{A}$ using Algorithm 1, to obtain $\overline{\mathbf{L}}^T$, $\mathbf{B}$ and $\overline{\mathbf{R}}^T$, such that
$\overline{\mathbf{L}}^T \mathbf{B} \overline{\mathbf{R}}^T = \mathbf{A}$

Compute the SVD of $\mathbf{B}$, to obtain $\mathbf{W} \in \mathbb{R}^{r \times r}$, $\mathbf{\Sigma}$, $\mathbf{X} \in \mathbb{R}^{c \times c}$, such that
$\mathbf{B} = \mathbf{W} \mathbf{\Sigma} \mathbf{X}^T$

$\mathbf{U} = \overline{\mathbf{L}}^T \mathbf{W} \qquad \overline{\mathbf{V}}^T = \mathbf{X}^T \overline{\mathbf{R}}^T$

$$A = \overline{\mathbf{L}}^T B \bar{R}^T$$

$$A = \overline{\mathbf{L}}^T \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \bar{R}^T$$

L, R – quaternion, unitary

B – real, bidiagonal

SVD for Quaternion Matrices

Theorem 3. *Given an arbitrary quaternion matrix* $\mathbf{A} \in \mathbb{H}^{r\times c}$ *with* r *rows and* c *columns, there exists a pair of unitary quaternion matrices* $\mathbf{L} \in \mathbb{H}^{r\times r}$ *and* $\mathbf{R} \in \mathbb{H}^{c\times c}$, *and a* real *bidiagonal matrix* $\mathbf{B} \in \mathbb{R}^{r\times c}$ *such that*

$$\mathbf{LAR} = \mathbf{B}.$$

SVD for Quaternion Matrices

Theorem 3. *Given an arbitrary quaternion matrix* $\mathbf{A} \in \mathbb{H}^{r \times c}$ *with* r *rows and* c *columns, there exists a pair of unitary quaternion matrices* $\mathbf{L} \in \mathbb{H}^{r \times r}$ *and* $\mathbf{R} \in \mathbb{H}^{c \times c}$, *and a* real *bidiagonal matrix* $\mathbf{B} \in \mathbb{R}^{r \times c}$ *such that*

$$\mathbf{LAR} = \mathbf{B}.$$

Theorem 4. *Given an arbitrary quaternion matrix* $\mathbf{A} \in \mathbb{H}^{r \times c}$, *and a real bidiagonal matrix* $\mathbf{B} \in \mathbb{R}^{r \times c}$ *as defined in Theorem 3, the singular values of* $\mathbf{A}$ *are the same as the singular values of* $\mathbf{B}$.

Proof. From Theorem 3 there exist unitary quaternion matrices $\mathbf{L}$ and $\mathbf{R}$ that will transform $\mathbf{A}$ to $\mathbf{B}$, that is $\mathbf{LAR} = \mathbf{B}$ and since $\mathbf{L}$ and $\mathbf{R}$ are unitary, $\mathbf{A} = \overline{\mathbf{L}}^T \mathbf{B} \overline{\mathbf{R}}^T$. The singular value decomposition of $\mathbf{B} = \mathbf{U \Sigma V}^T$ where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal, hence $\mathbf{A} = \overline{\mathbf{L}}^T \mathbf{U \Sigma V}^T \overline{\mathbf{R}}^T$. From the uniqueness of the singular values, and from the fact that $\overline{\mathbf{L}}^T \mathbf{U}$ is unitary and $\mathbf{V}^T \overline{\mathbf{R}}^T$ is unitary, it follows that $\mathbf{\Sigma}$ contains the singular values of the quaternion matrix $\mathbf{A}$. $\square$

SVD for Quaternion Matrices

Theorem 3. *Given an arbitrary quaternion matrix* $\mathbf{A} \in \mathbb{H}^{r\times c}$ *with* r *rows and* c *columns, there exists a pair of unitary quaternion matrices* $\mathbf{L} \in \mathbb{H}^{r\times r}$ *and* $\mathbf{R} \in \mathbb{H}^{c\times c}$, *and a* real *bidiagonal matrix* $\mathbf{B} \in \mathbb{R}^{r\times c}$ *such that*

$$\mathbf{LAR} = \mathbf{B}.$$

Theorem 3.1. *Consider an arbitrary quaternion vector* $a \in [C\ell_{0,2}]^r$ *and a real vector* $v \in \mathbb{R}^r$ *with unit norm,* $v^T v = 1$. *There exist a quaternion vector* $u \in [C\ell_{0,2}]^r$, $\|u\| = \sqrt{2}$, *and a unit quaternion scalar* $z \in C\ell_{0,2}$, $\widehat{\widetilde{z}} z = 1$, *such that for*

$$H := z(I_r - u\widehat{\widetilde{u}}^T) \tag{3.5}$$

we have

$$Ha = \|a\| v. \tag{3.6}$$

We call u left Householder vector and H left Householder matrix.

SVD for Quaternion Matrices

Theorem 3.1. *Consider an arbitrary quaternion vector* $a \in [C\ell_{0,2}]^r$ *and a real vector* $v \in \mathbb{R}^r$ *with unit norm,* $v^T v = 1$*. There exist a quaternion vector* $u \in [C\ell_{0,2}]^r$, $\|u\| = \sqrt{2}$, *and a unit quaternion scalar* $z \in C\ell_{0,2}$, $\widehat{\widetilde{z}}z = 1$, *such that for*

$$H := z(I_r - u\widehat{\widetilde{u}}^T) \tag{3.5}$$

we have

$$Ha = \|a\|v. \tag{3.6}$$

Remark 3.2. If we take $v = (1, 0, \ldots, 0) \in \mathbb{R}^r$, then for a fixed $a \in [C\ell_{0,2}]^r$,

$$Ha = \begin{bmatrix} \|a\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^r. \tag{3.14}$$

SVD for Quaternion Matrices

Theorem 3.1. *Consider an arbitrary quaternion vector* $a \in [C\ell_{0,2}]^r$ *and a real vector* $v \in \mathbb{R}^r$ *with unit norm,* $v^T v = 1$*. There exist a quaternion vector* $u \in [C\ell_{0,2}]^r$, $\|u\| = \sqrt{2}$, *and a unit quaternion scalar* $z \in C\ell_{0,2}$, $\widehat{\widetilde{z}} z = 1$, *such that for*

$$H := z(I_r - u\widehat{\widetilde{u}}^T) \tag{3.5}$$

we have

$$Ha = \|a\|v. \tag{3.6}$$

Remark 3.3. Constructed H is unitary because

$$\begin{aligned} (\widehat{\widetilde{H}})^T H &= (I_r - u\widehat{\widetilde{u}}^T)\widehat{\widetilde{z}} z (I_r - u\widehat{\widetilde{u}}^T) = (I_r - u\widehat{\widetilde{u}}^T)(I_r - u\widehat{\widetilde{u}}^T) && (3.15) \\ &= I_r - 2u\widehat{\widetilde{u}}^T + u\widehat{\widetilde{u}}^T u\widehat{\widetilde{u}}^T = I_r - 2u\widehat{\widetilde{u}}^T + 2u\widehat{\widetilde{u}}^T = I_r. && (3.16) \end{aligned}$$

and

$$H(\widehat{\widetilde{H}})^T = z(I_r - u\widehat{\widetilde{u}}^T)(I_r - u\widehat{\widetilde{u}}^T)\widehat{\widetilde{z}} = z\widehat{\widetilde{z}} I_r = \langle z\widehat{\widetilde{z}}\rangle_0 I_r = \langle \widehat{\widetilde{z}} z\rangle_0 I_r = I_r.$$

SVD for Quaternion Matrices

Theorem 3.1. *Consider an arbitrary quaternion vector* $a \in [C\ell_{0,2}]^r$ *and a real vector* $v \in \mathbb{R}^r$ *with unit norm,* $v^T v = 1$. *There exist a quaternion vector* $u \in [C\ell_{0,2}]^r$, $\|u\| = \sqrt{2}$, *and a unit quaternion scalar* $z \in C\ell_{0,2}$, $\widehat{\widetilde{z}} z = 1$, *such that for*

$$H := z(I_r - u\widehat{\widetilde{u}}^T) \tag{3.5}$$

we have

$$Ha = \|a\|v. \tag{3.6}$$

Sequence of steps to find u and z:

1. $\alpha = \|\mathbf{a}\|$

2. $r = |\mathbf{a}^T \mathbf{v}|$

3. $\zeta = \begin{cases} 1 & : \quad r = 0 \\ -\frac{\mathbf{a}^T \mathbf{v}}{r} & : \quad r > 0 \end{cases}$

4. $\mu = \sqrt{\alpha(\alpha + r)}$

5. $\mathbf{u} = \frac{1}{\mu}(\mathbf{a} - \zeta \mathbf{v} \alpha)$

6. $z = \zeta^{-1}$

Proposed solution: Clifford Scores for CUR

03. Clifford Scores

U — Unitary $[C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$

Σ — Real

V^T — Unitary $[C\ell_{p,q}(\mathbb{R})]^{I_1 \times I_2}$

$\mapsto$

$U\Sigma$: Clifford score of a row = $\dfrac{\text{Clifford norm of a row}}{\text{Clifford norm of } U\Sigma}$

ΣV^T: Clifford score of a column = $\dfrac{\text{Clifford norm of a column}}{\text{Clifford norm of } \Sigma V^T}$

Sample R1 indices of rows and R2 indices of columns
with probabilities represented by Clifford scores.

Proposed solution: Clifford Scores for CUR

04. Apply standard methods of CUR for tensors

FSTD

Algorithm 4: Fast Sampling Tucker Decomposition (FSTD) Algorithm for 3rd-Order Tensors [61]

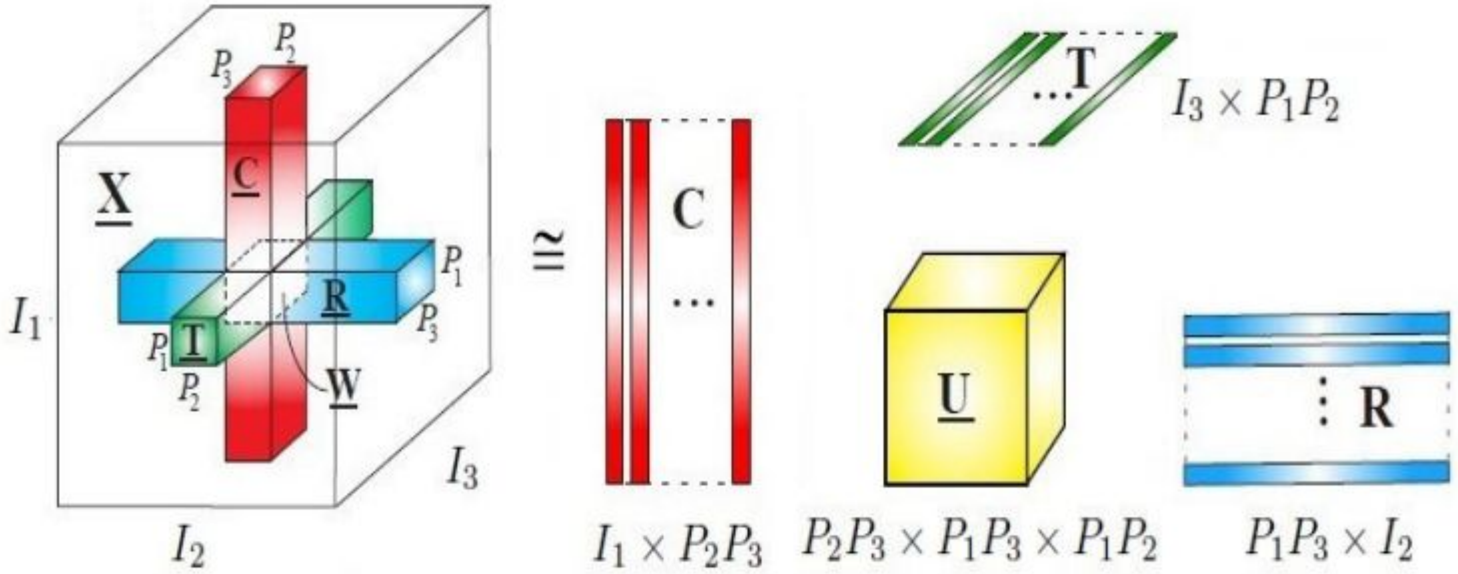
Input : A data tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, indices $\mathcal{I}_n \subseteq [I_n],\ n = 1, 2, 3$

Output: Tucker approximation of the tensor $\underline{\mathbf{X}}$

1 Generate the Intersection Subtensor $\underline{\mathbf{W}} = \underline{\mathbf{U}}(\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3)$

2 Generate the Subsampled Matrices $\mathbf{A}_1 = \mathbf{X}_{(1)}(:, \mathcal{I}_2, \mathcal{I}_3)$, $\mathbf{A}_2 = \mathbf{X}_{(2)}(\mathcal{I}_1, :, \mathcal{I}_3)$ and $\mathbf{A}_3 = \mathbf{X}_{(3)}(\mathcal{I}_1, \mathcal{I}_2, :)$

3 $\underline{\mathbf{X}} \cong [\![\underline{\mathbf{W}}, \mathbf{A}_1\mathbf{W}^{+}_{(1)}, \mathbf{A}_2\mathbf{W}^{+}_{(2)}, \mathbf{A}_3\mathbf{W}^{+}_{(3)}]\!]$



Cesar F. Caiafa, Andrzej Cichocki,
***Generalizing the column–row matrix decomposition to multi-way arrays**,*
Linear Algebra and its Applications, 2010

S. Ahmadi-Asl et al., "**Cross Tensor Approximation Methods for Compression and Dimensionality Reduction**," in IEEE, vol. 9, 2021

Skoltech

** Recent paper on CUR method for quaternion matrices

29.02.2024

Efficient quaternion CUR method for low-rank approximation to quaternion matrix

Pengling Wu[1], Kit Ian Kou[1*], Hongmin Cai[2], Zhaoyuan Yu[3]

[1*]Department of Mathematics, Faculty of Science and Technology, University of Macau, Macau 100190, China.
[2]School of Computer Science & Engineering, South China University of Technology, Guangzhou 510006, China.
[3]Department, School of Geography, Nanjing Normal University, Nanjing 210023, China.

2 methods of rows/columns sampling:

- max norm sampling
- uniform sampling

https://arxiv.org/pdf/2402.19147.pdf

07

Implementation

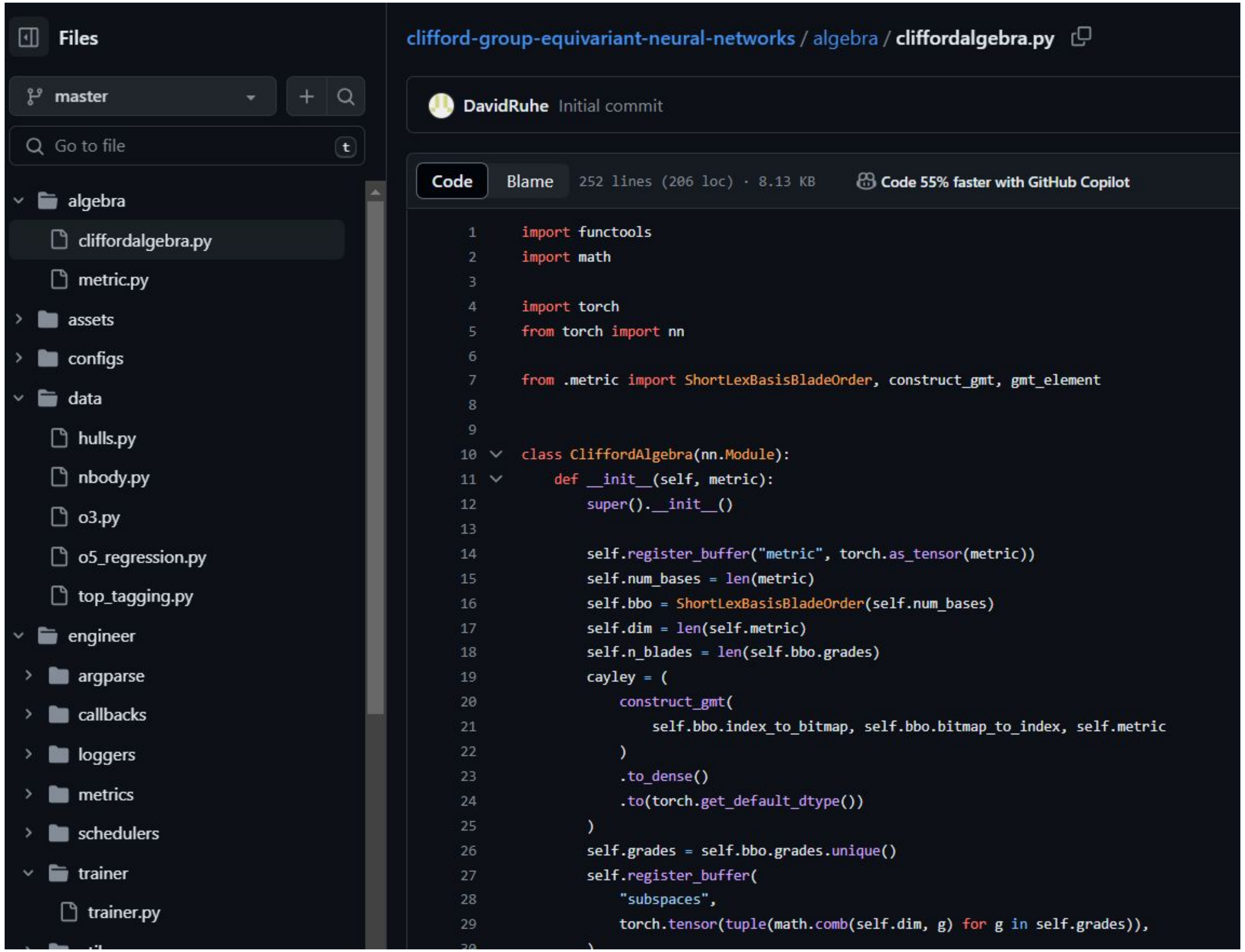
My implementation and experiments:

https://colab.research.google.com/drive/1k85y1e5sbqqZ9zeljF9phGcRpfRH_SbO?usp=sharing

What I used:

Implemented product of elements of Clifford algebra

Elements of $C\ell_{p,q}(\mathbb{R})$ are represented as torch tensors



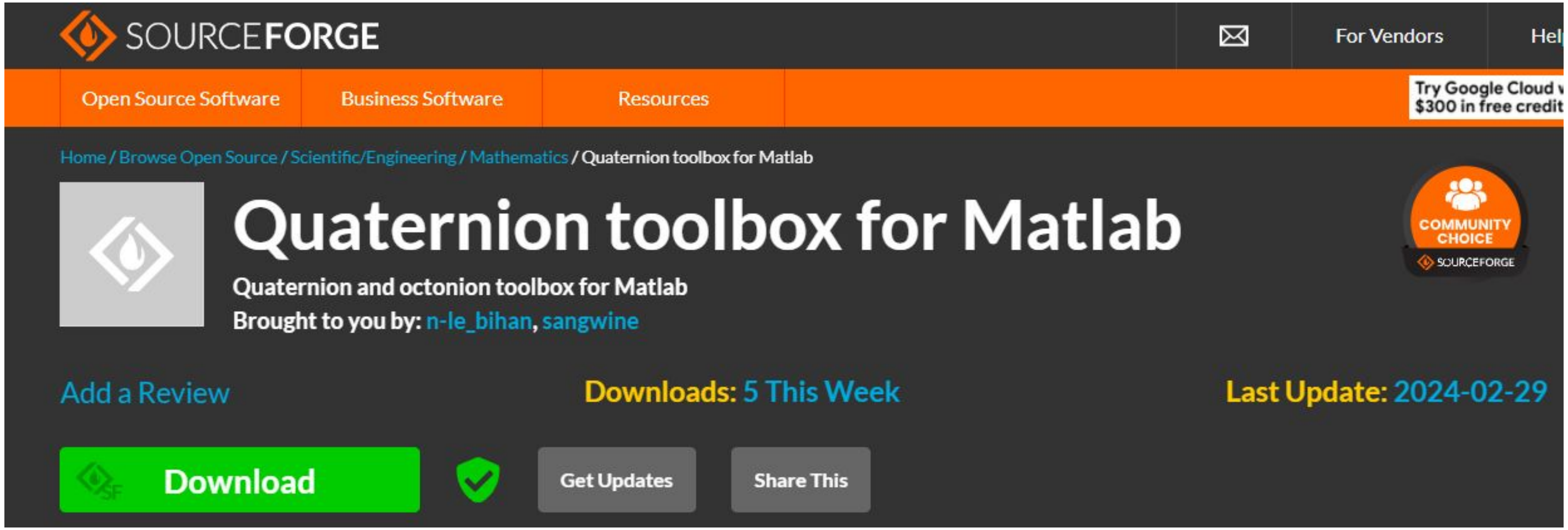

```python
import functools
import math

import torch
from torch import nn

from .metric import ShortLexBasisBladeOrder, construct_gmt, gmt_element


class CliffordAlgebra(nn.Module):
    def __init__(self, metric):
        super().__init__()

        self.register_buffer("metric", torch.as_tensor(metric))
        self.num_bases = len(metric)
        self.bbo = ShortLexBasisBladeOrder(self.num_bases)
        self.dim = len(self.metric)
        self.n_blades = len(self.bbo.grades)
        cayley = (
            construct_gmt(
                self.bbo.index_to_bitmap, self.bbo.bitmap_to_index, self.metric
            )
            .to_dense()
            .to(torch.get_default_dtype())
        )
        self.grades = self.bbo.grades.unique()
        self.register_buffer(
            "subspaces",
            torch.tensor(tuple(math.comb(self.dim, g) for g in self.grades)),
```

https://github.com/DavidRuhe/clifford-group-equivariant-neural-networks/blob/master/algebra/cliffordalgebra.py

What I used for inspiration:

https://sourceforge.net/projects/qtfm/

SVD for Quaternions in Matlab



Quaternion Singular Value Decomposition based on Bidiagonalization to a Real Matrix using Quaternion Householder Transformations.

S. J. Sangwine[†§] N. Le Bihan[‡]

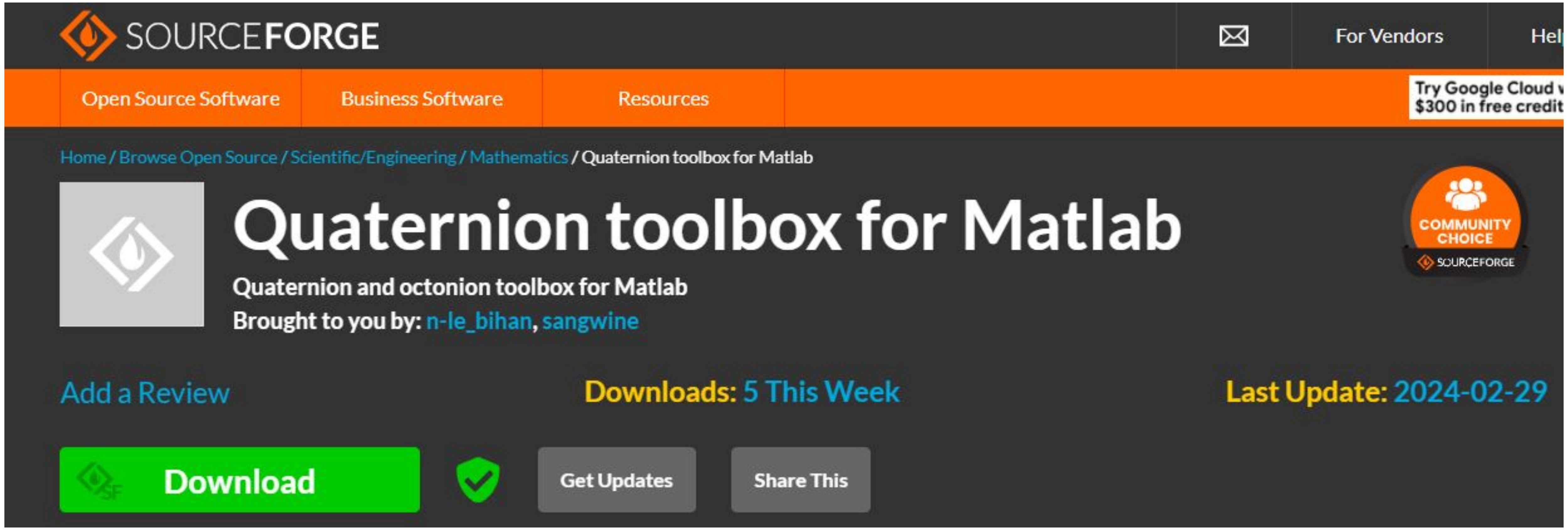
October 16, 2018

Skoltech

What I used for inspiration:

https://sourceforge.net/projects/qtfm/

SVD for Quaternions in Matlab



I have not found any open source implementation of SVD for Clifford algebra matrices in Python

What I have implemented

Representation of a torch tensor as a matrix with Clifford algebra elements

```
def hyper_from_arb_tensor(X):
  '''
  Builds a hypermatrix with quaternion values from a 3rd order tensor (I,J,4)
  '''
  hyper = []

  for i in range(X.shape[0]):
    t_in_row = []
    for j in range(X.shape[1]):
      t_in_row.append(algebra.embed(torch.as_tensor(X[i,j,:]), torch.tensor([0,1,2,3])))
    row = torch.stack(t_in_row)
    hyper.append(row)

  hyper_matrix = torch.stack(hyper)

  return hyper_matrix
```

What I have implemented

Clifford norm for Clifford algebra elements

```
def cl_norm_2(a):
  '''
  \tilde{\hat{a}} * a
  '''
  a_bar = clifford_conj(a)
  res = algebra.geometric_product(a_bar, a)
  return res
```

Clifford norm for vectors with Clifford algebra elements

```
def cl_vector_norm_2(vec):
  '''
  ||a|| for a -- multivector
  '''
  res = algebra.embed(torch.zeros(2**(algebra.dim)), torch.tensor([0,1,2,3]))
  for i in range(vec.shape[0]):
    res += cl_norm_2(vec[i])
  return res
```

What I have implemented

Product of matrices with Clifford algebra elements

```
def vectors_product(u, v):
  '''
  u^T * v
  '''
  product = torch.zeros((u.shape[0], v.shape[0], 2**(algebra.dim)))
  for i in range(u.shape[0]):
    for j in range(v.shape[0]):
      product[i,j] = algebra.geometric_product(u[i], v[j])
  return product
```

```
def quat_matmul(A, B):
  res = torch.zeros((A.shape[0], B.shape[1], 2**(algebra.dim)))
  for i in range(A.shape[0]):
    for j in range(B.shape[1]):
      el = torch.zeros_like(A[0,0])
      for s in range(A.shape[1]):
        el += algebra.geometric_product(A[i,s], B[s,j])
      res[i,j] = el
  return res
```

What I have implemented

Householder vector and matrix for a Clifford algebra matrix

```
def householder_vector(a):
  r = torch.sqrt(cl_norm_2(a[0]))[0]
  if r == 0:
    dz = 1
  else:
    dz = - a[0] / r
  alpha = torch.sqrt(cl_vector_norm_2(a))[0]
  mu = torch.sqrt(alpha * (alpha + r))
  dz_v = torch.zeros((a.shape[0], a.shape[1]))
  dz_v[0] = dz
  u = (1 / mu) * (a -  alpha * dz_v)
  return u, dz
```

```
def householder_matrix(u, dz):
  uuj = vectors_product(u, clifford_conj(u))
  id = torch.zeros((uuj.shape[0], uuj.shape[1], uuj.shape[2]))
  id[torch.arange(uuj.shape[0]), torch.arange(uuj.shape[1])] = torch.tensor([1., 0., 0., 0.])
  z = clifford_conj(dz)
  H = scalar_matrix_product(z, id - uuj)
  return H
```

What I have implemented

Bidiagonalization of Clifford algebra matrices

```
def bidiagonalize(A):
  u, dz = householder_vector(A[:,0,:])

  L = householder_matrix(u, dz)
  A = quat_matmul(L, A)

  R = torch.zeros((A.shape[1], A.shape[1], 2**(algebra.dim)))
  R[torch.arange(A.shape[1]), torch.arange(A.shape[1])] = torch.tensor([1., 0., 0., 0.])

  B = torch.zeros_like(A)
  B[:,:,0] = A[:,:,0]
  return A, L, B, R
```

```
def back_process(R, L, R_1, L_1, A, T):
  R[1:R.shape[0], 1:R.shape[1], :] = L_1
  L = quat_matmul(R_1, L)
  A[:,1:A.shape[1],:] = T.permute(1,0,2)
  return R, L, A
```

```
def full_bidiagonalize(mat):
  '''
  For a matrix X returns the factors R, A, L, such that L^{conjugate transpose} * A * R = X,
  where L^{conjugate transpose} * L = I,
  R^{conjugate transpose} * R^{conjugate transpose} = I,
  A is real
  '''
  A, L, _, R = bidiagonalize(mat)
  if A.shape[1] > 1:
    A_m_1 = clifford_conj(A[:,1:A.shape[1],:].permute(1,0,2))
    R_1, A_m_2, L_1 = full_bidiagonalize(A_m_1)
    B_m_2 = A_m_2
    R, L, A = back_process(R, L, R_1, L_1, A, B_m_2)
  return R, A, L
```

```
def check_bidiagonalization(mat, R, A, L):
  return np.allclose(A, quat_matmul(quat_matmul(L, mat), clifford_conj(R.permute(1,0,2))))
```

What I have implemented

SVD of matrices with Clifford algebra elements

```
def qSVD(mat):
  r, a, l = full_bidiagonalize(mat)

  if check_bidiagonalization(mat, r, a, l):
    W, S, Xt = np.linalg.svd(a[:,:,0], full_matrices=True)

    # get U
    W_ = torch.zeros((W.shape[0], W.shape[1], 2**(algebra.dim)))
    W_[:,:,0] = torch.tensor(W)
    U = quat_matmul(clifford_conj(l.permute(1,0,2)), hyper_from_arb_tensor(W_))

    # get Sigma
    Sigma = torch.zeros((S.shape[0], S.shape[0], 2**(algebra.dim)))
    Sigma[:,:,0] = torch.tensor(np.diag(S))
    Sigma = hyper_from_arb_tensor(Sigma)

    # get V
    Xt_ = torch.zeros((Xt.shape[0], Xt.shape[1], 2**(algebra.dim)))
    Xt_[:,:,0] = torch.tensor(Xt)
    V_conj_T = quat_matmul(hyper_from_arb_tensor(Xt_), r)

  return U, Sigma, V_conj_T
```

```
def qSVD_random_sampling(R1, R2, U, Sigma, V_conj_T):
  US = quat_matmul(U, Sigma)
  SV = quat_matmul(Sigma, V_conj_T)

  norms_US = torch.zeros((US.shape[0], 2**(algebra.dim)))
  for i in range(US.shape[0]):
    norms_US[i] = cl_vector_norm_2(US[i,:,:])
  norms_US /= norms_US.sum()

  norms_SV = torch.zeros((SV.shape[0], 2**(algebra.dim)))
  for i in range(SV.shape[0]):
    norms_SV[i] = cl_vector_norm_2(SV[i,:,:])
  norms_SV /= norms_SV.sum()

  r1 = np.random.choice(U.shape[0], R1, replace=False, p=norms_US[:,0])
  r2 = np.random.choice(V_conj_T.shape[1], R2, replace=False, p=norms_SV[:,0])

  return r1, r2
```

```
def check_qSVD(mat, U, Sigma, V_conj_T):
  return np.allclose(mat, quat_matmul(quat_matmul(U, Sigma), V_conj_T))
```

What I have implemented

Classic methods of sampling indices of tubes for CUR

Sampling indices of rows and columns from uniform distribution

```
def uniform_sampling(X, R1, R2, R3):
    r1 = np.random.choice(X.shape[0], R1, replace=False)
    r2 = np.random.choice(X.shape[1], R2, replace=False)
    r3 = np.random.choice(X.shape[2], R3, replace=False)
    return r1, r2, r3
```

Sampling indices of rows and columns depending on the norms of rows and columns

```
def max_norm_random_sampling(X, R1, R2, R3):
  norms_0, norms_1, norms_2 = torch.empty(X.shape[0]), torch.empty(X.shape[1]), torch.empty(X.shape[2])
  for i in range(X.shape[0]):
    norms_0[i] = np.linalg.norm(X[i,:,:])
  for j in range(X.shape[1]):
    norms_1[j] = np.linalg.norm(X[:,j,:])
  for k in range(X.shape[2]):
    norms_2[k] = np.linalg.norm(X[:,:,k])
  norms_0 /= norms_0.sum()
  norms_1 /= norms_1.sum()
  norms_2 /= norms_2.sum()

  r1 = np.random.choice(X.shape[0], R1, replace=False, p=norms_0)
  r2 = np.random.choice(X.shape[1], R2, replace=False, p=norms_1)
  ce(X.shape[2], R3, replace=False, p=norms_2)
```

Choose rows and columns with max norm (deterministic)

```
def max_norm_sampling(X, R1, R2, R3):
  norms_0, norms_1, norms_2 = torch.empty(X.shape[0]), torch.empty(X.shape[1]), torch.empty(X.shape[2])
  for i in range(X.shape[0]):
    norms_0[i] = np.linalg.norm(X[i,:,:])
  for j in range(X.shape[1]):
    norms_1[j] = np.linalg.norm(X[:,j,:])
  for k in range(X.shape[2]):
    norms_2[k] = np.linalg.norm(X[:,:,k])
  norms_0 /= norms_0.sum()
  norms_1 /= norms_1.sum()
  norms_2 /= norms_2.sum()

  r1 = torch.topk(torch.arange(X.shape[0]), R1).indices
  r2 = torch.topk(torch.arange(X.shape[1]), R2).indices
  r3 = torch.topk(torch.arange(X.shape[2]), R3).indices

  return r1, r2, r3
```

Skoltech

What I have implemented

FSTD for CUR

```
def FSTD(X, U, Sigma, V_conj_T, method, R1, R2, R3=3):

    if method == 'uniform':
      r1, r2, r3 = uniform_sampling(X.numpy(), R1, R2, R3)

    elif method == 'max_random_norm':
      r1, r2, r3 = max_norm_random_sampling(X.numpy(), R1, R2, R3)

    elif method == 'max_norm':
      r1, r2, r3 = max_norm_sampling(X.numpy(), R1, R2, R3)

    elif method == 'qsvd':
      r1, r2 = qSVD_random_sampling(R1, R2, U, Sigma, V_conj_T)
      r3 = torch.arange(3)

    X = X.numpy()

    W = X[r1[:, None, None], r2[None, :, None], r3[None, None, :]]

    A1 = base.unfold(X[:, r2, :][:, :, r3], 0)
    A2 = base.unfold(X[r1, :, :][:, :, r3], 1)
    A3 = base.unfold(X[r1, :, :][:, r2, :], 2)

    U1 = A1 @ np.linalg.pinv(base.unfold(W,0))
    U2 = A2 @ np.linalg.pinv(base.unfold(W,1))
    U3 = A3 @ np.linalg.pinv(base.unfold(W,2))

    return W, U1, U2, U3
```

08

Experiments

Experiment 1: Check on Random Tensors

$\mathbb{R}^{3\times3\times3}$

```
1 X = np.random.rand(*(3,3,3))
2 X
```

```
array([[[0.75319981, 0.1955606 , 0.33991231],
        [0.78667539, 0.28490217, 0.98396345],
        [0.46448057, 0.54748648, 0.69796232]],

       [[0.26664786, 0.75646392, 0.78040978],
        [0.12004971, 0.94907363, 0.92413078],
        [0.72959673, 0.78549808, 0.4727191 ]],

       [[0.40657765, 0.74970375, 0.15798577],
        [0.30541381, 0.23429091, 0.63628207],
        [0.37323747, 0.71960061, 0.32840664]]])
```

Choose indices of 2 rows and 2 columns:

```
Uniform tubes sampling (dim0, dim1, dim2):  (array([1, 2]), array([1, 0]),
Max norm tubes sampling (dim0, dim1, dim2):  (array([0, 1]), array([0, 1])
Max norm tubes choice (dim0, dim1, dim2):  (array([2, 0]), array([1, 0]),
QSVD tubes sampling (dim0, dim1): (array([0, 1]), array([0, 1]))
```

Approximation error

```
0.571565473944327
0.3739391603252732
0.3146209934000139
0.3582166095037097
```

Experiment 2: Check on Random Tensors $\mathbb{R}^{10\times 10\times 3}$

Choose indices of 7 rows and 7

```
----------------------------------------
Uniform tubes sampling (dim0, dim1, dim2):  (array([0, 2, 9, 6, 3, 4, 1]), array([6, 2, 1, 4, 5, 0, 7]),
Max norm tubes sampling (dim0, dim1, dim2):  (array([6, 4, 0, 5, 9, 2, 8]), array([8, 5, 4, 6, 7, 3, 9])
Max norm tubes choice (dim0, dim1, dim2):  (array([2, 1, 4, 9, 8, 6, 7]), array([0, 5, 2, 3, 9, 7, 1]),
QSVD tubes sampling (dim0, dim1): (array([6, 1, 3, 7, 2, 4, 8]), array([0, 1, 2, 3, 6, 5, 4]))
----------------------------------------
Approximation error with uniform random sampling 0.41687292509366924
Approximation error with max norm choice 0.33934765604219397
Approximation error with max norm random sampling 0.3265552826218394
Approximation error with QSVD sampling 0.4059891826109999
```

Choose indices of 2 rows and 2 columns:

```
Number of tubes along dimensions: 2, 2, 3
----------------------------------------
Uniform tubes sampling (dim0, dim1, dim2):  (array([2, 7]), array([7, 5]),
Max norm tubes sampling (dim0, dim1, dim2):  (array([6, 9]), array([6, 9])
Max norm tubes choice (dim0, dim1, dim2):  (array([2, 5]), array([5, 9]),
QSVD tubes sampling (dim0, dim1): (array([4, 5]), array([0, 3]))
----------------------------------------
Approximation error with uniform random sampling 0.6976090679137001
Approximation error with max norm choice 0.5777500171931693
Approximation error with max norm random sampling 0.587445523345739
Approximation error with QSVD sampling 0.6266513453568201
```

CUR for Image Completion

Algorithm 1: Tensor CUR algorithm for Nth-order tensor completion.

Input : An incomplete data tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, Tensor Rank $\mathbf{R}$, the set of observed components $\mathbf{\Omega}$, error bound ε and $\mathrm{MaxIter}$.

Output : Completed data tensor $\underline{\mathbf{X}}^*$

$\underline{\mathbf{X}}^{(0)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the original data tensor with missing pixels;

$\underline{\mathbf{Y}}^{(0)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is a zero tensor;

for $n = 0, 1, 2, \ldots$ **do**

  $\underline{\mathbf{Y}}^{(n+1)} \leftarrow$ Compute CUR approximation of the data tensor $\underline{\mathbf{X}}^{(n)}$ using selected fibers/slices and smoothing them,

  $\underline{\mathbf{X}}^{(n+1)} \leftarrow \mathbf{P}_{\underline{\mathbf{\Omega}}}(\underline{\mathbf{X}}^{(n)}) + \mathbf{P}_{\underline{\mathbf{\Omega}}^{\perp}}(\underline{\mathbf{Y}}^{(n+1)})$,

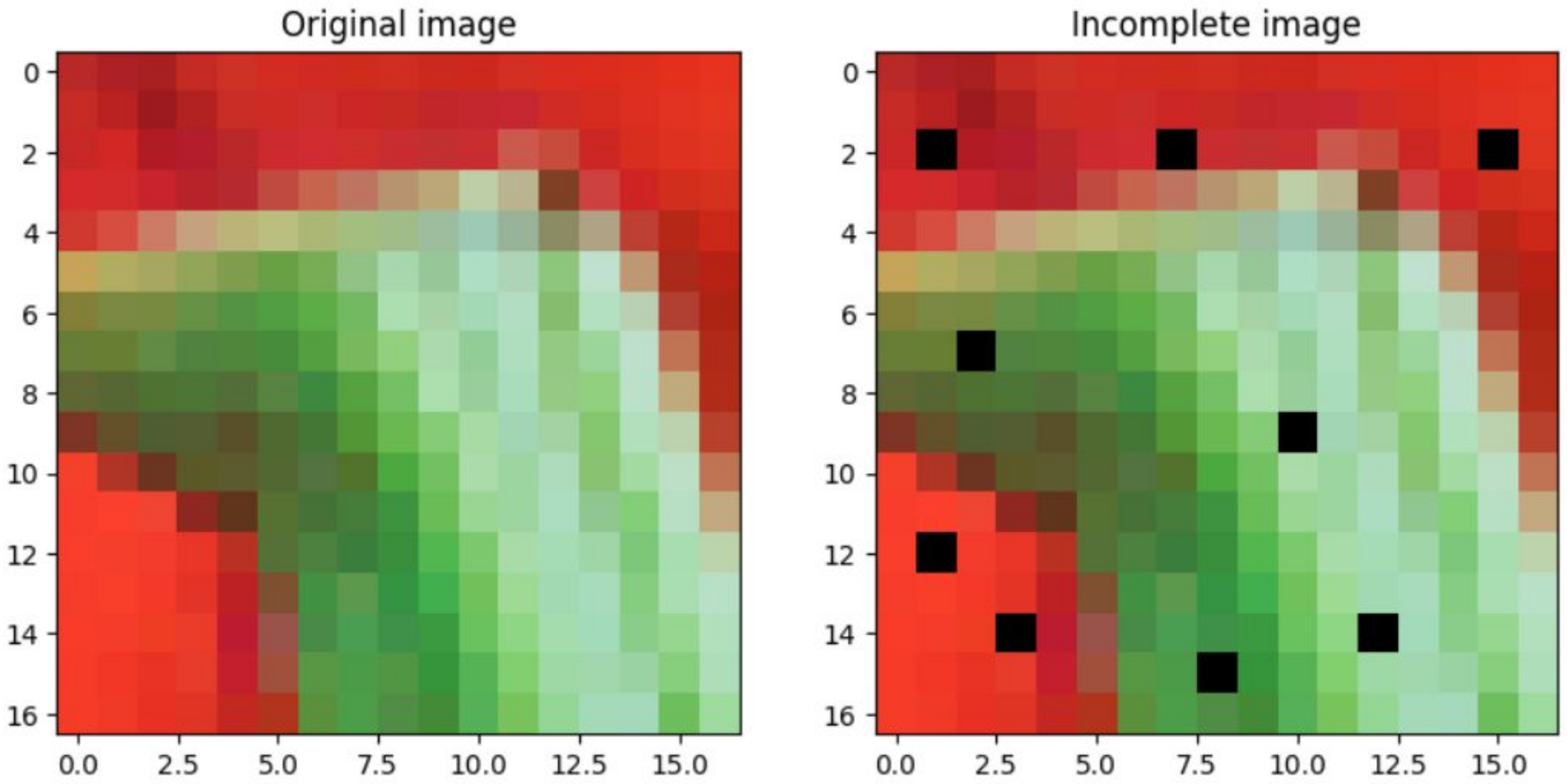
  if $\frac{\left\|\underline{\mathbf{X}}^{(n+1)} - \underline{\mathbf{X}}^{(n)}\right\|_F}{\left\|\underline{\mathbf{X}}^{(n+1)}\right\|_F} < \varepsilon \; or \; n > \mathrm{MaxIter}$ **then**

    $\underline{\mathbf{X}}^* = \underline{\mathbf{X}}^{(n+1)}$ and **break**,

  end

end

Experiment 3: Image Completion on Peppers

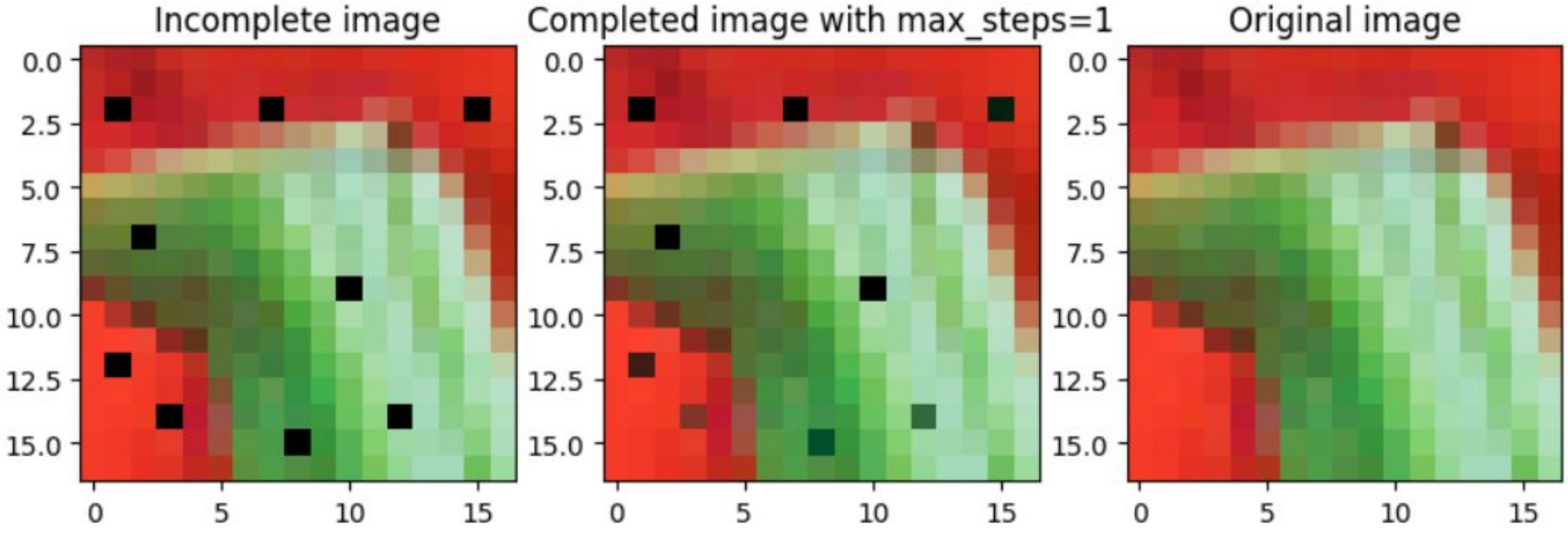


17x17 px

Experiment 3: Image Completion on Peppers

Clifford scores method for CUR
rows = 12, # columns = 12, # steps = 1

17x17 px

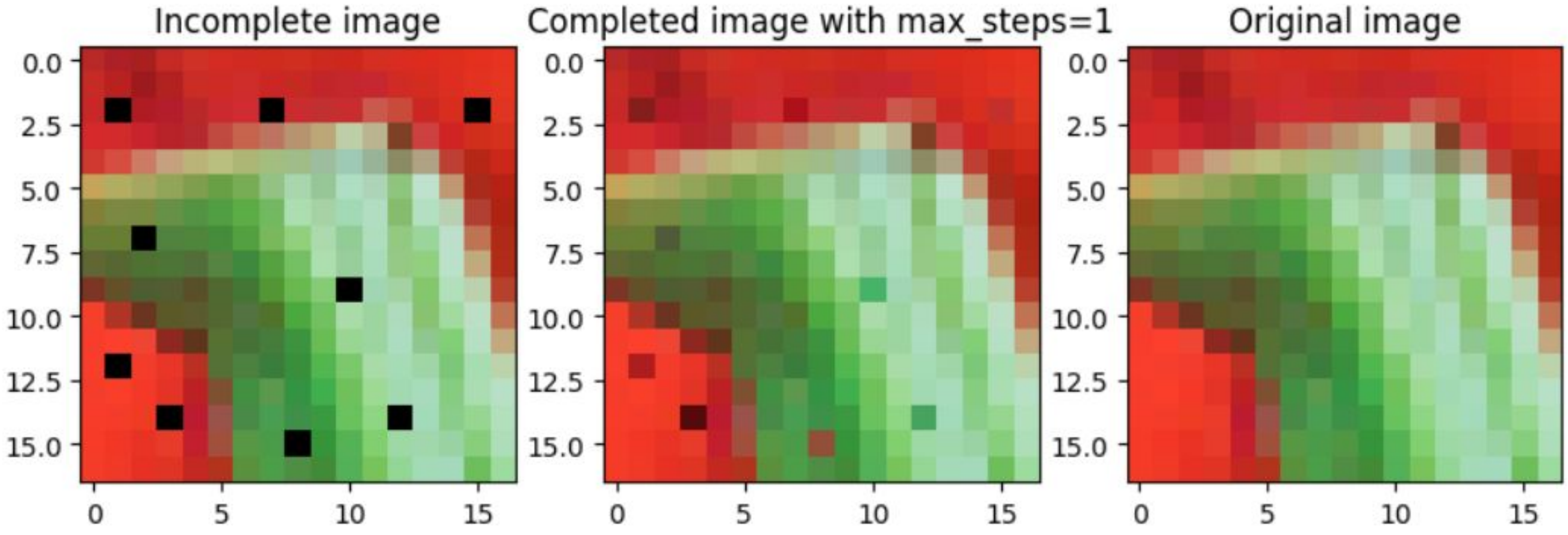


Skoltech

Experiment 3: Image Completion on Peppers

Clifford scores method for CUR
rows = 5, # columns = 5, # steps = 1

17x17 px



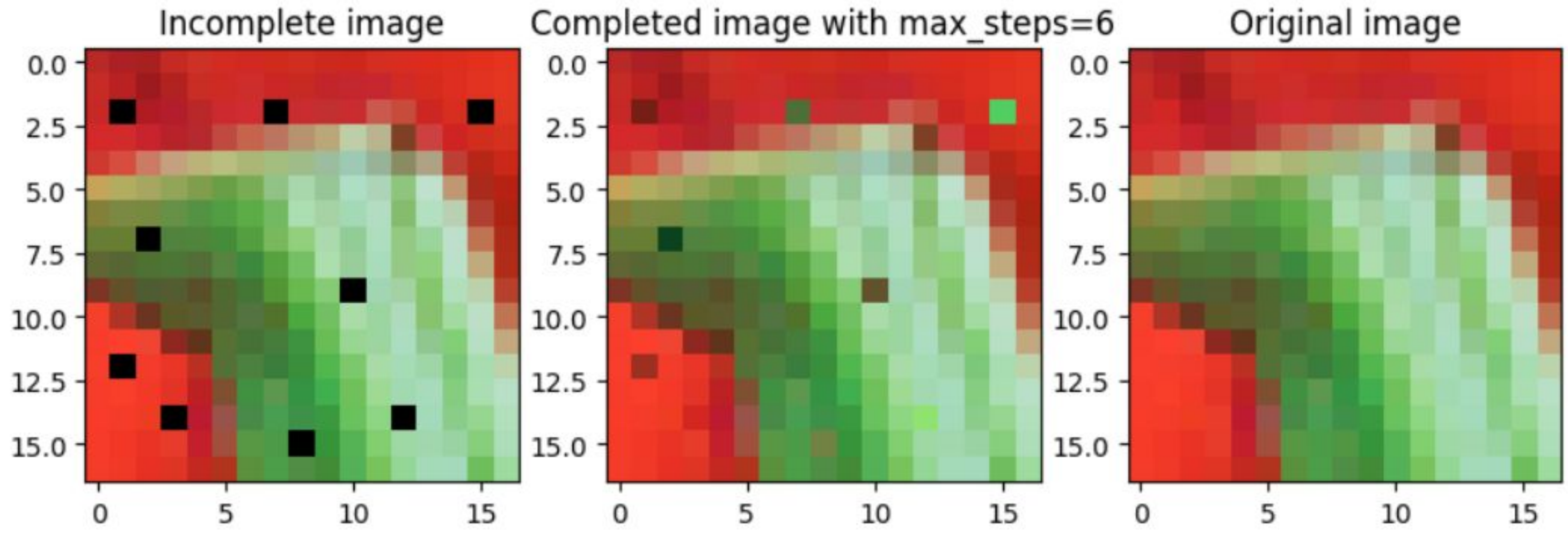
Skoltech

Experiment 3: Image Completion on Peppers

Clifford scores method for CUR
rows = 5, # columns = 5, # steps = 6

17x17 px

```
Difference between steps 0 and 1: 0.16846819749941785
Difference between steps 1 and 2: 0.11226734138371386
Difference between steps 2 and 3: 0.10900644388221012
Difference between steps 3 and 4: 0.09848333310423607
Difference between steps 4 and 5: 0.08370596207945351
Difference between steps 5 and 6: 0.13714777063063543
```

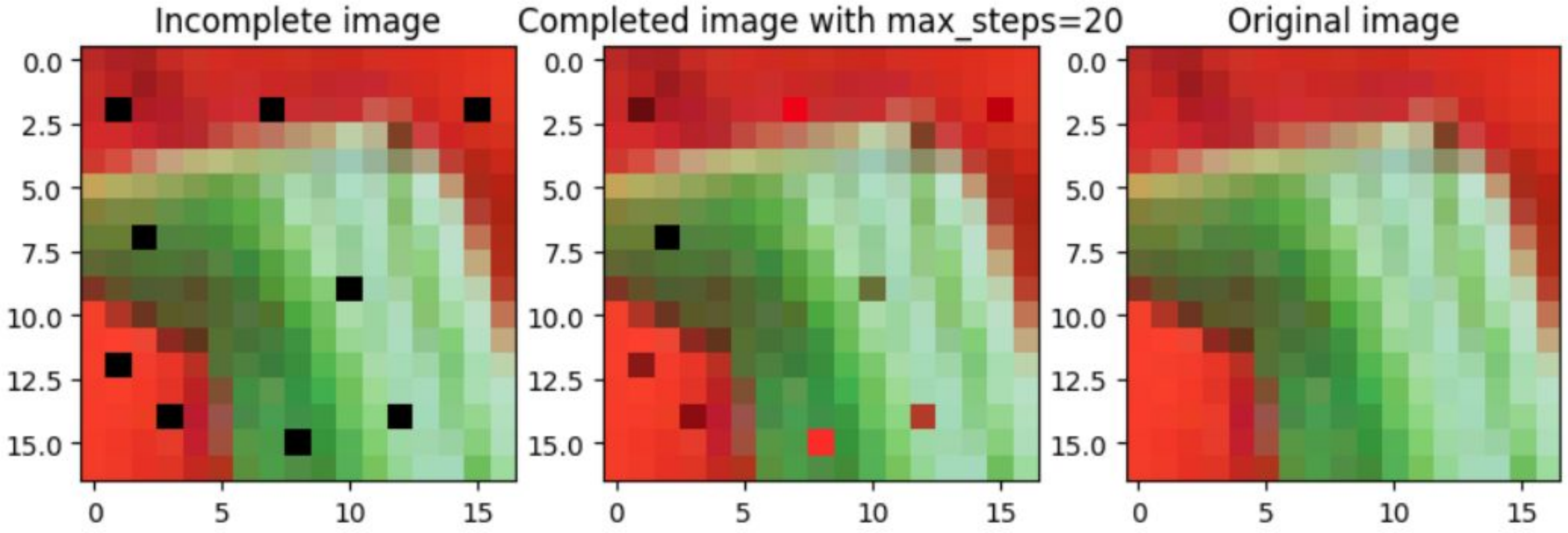


Skoltech

Experiment 3: Image Completion on Peppers

Clifford scores method for CUR
rows = 5, # columns = 5, # steps = 20

17x17 px

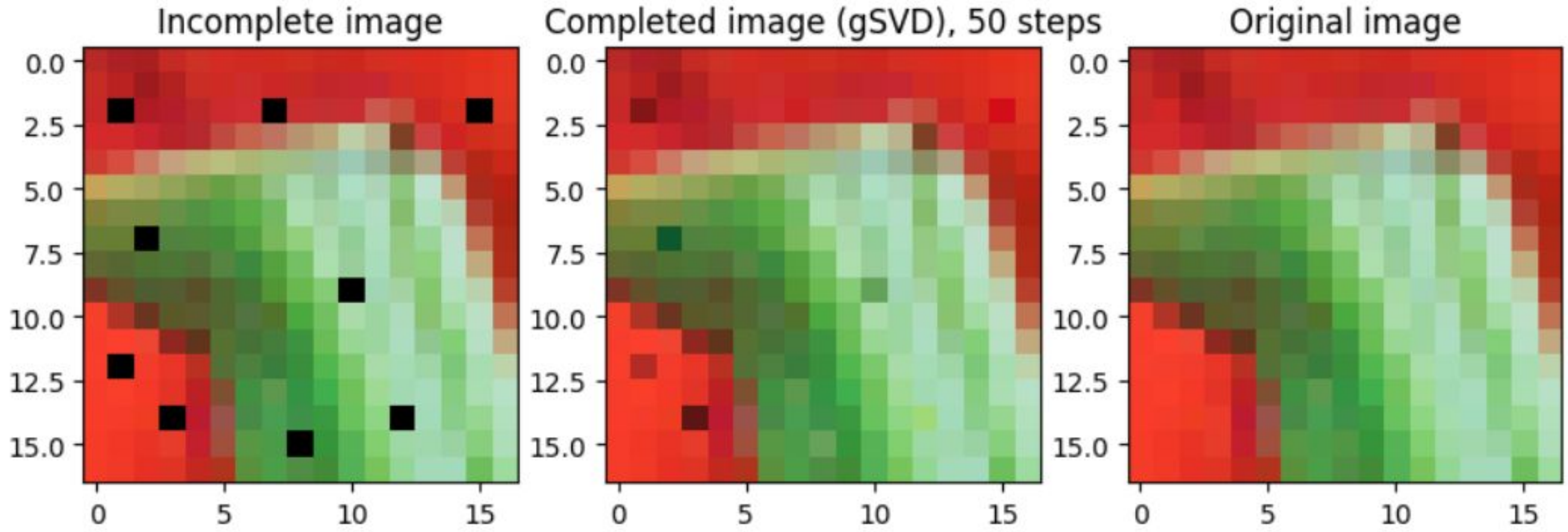


Skoltech

Experiment 3: Image Completion on Peppers

Clifford scores method for CUR
rows = 5, # columns = 5, # steps = 50

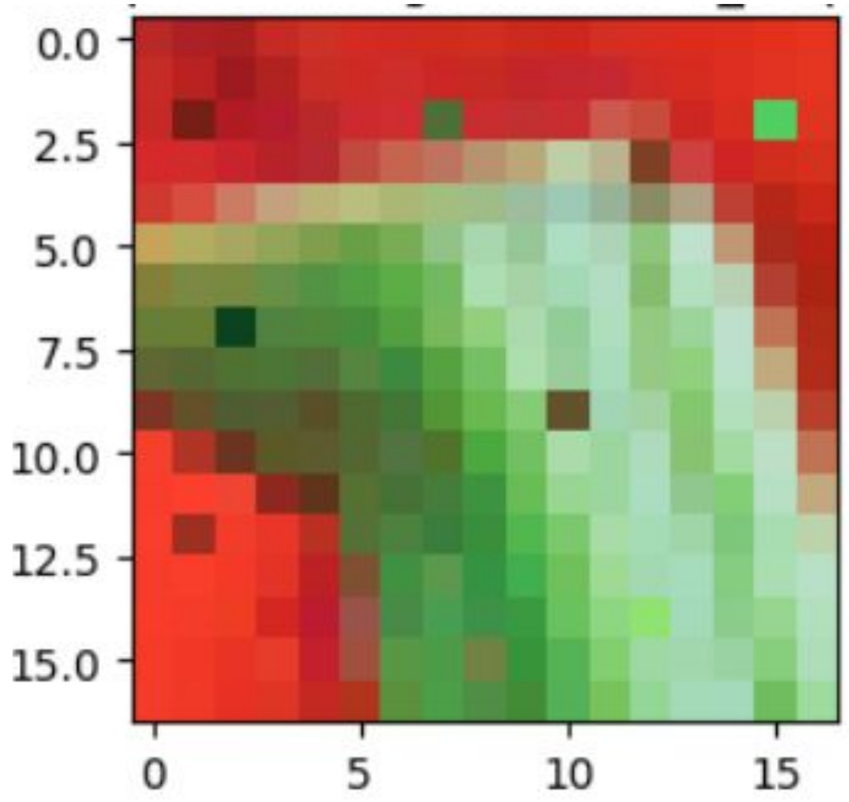
17x17 px



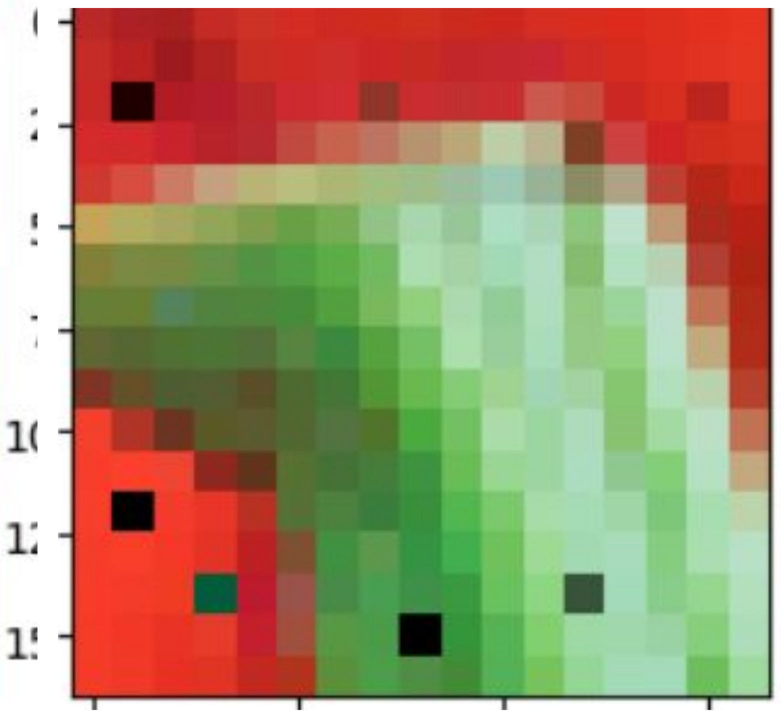
Experiment 3: Image Completion on Peppers

rows = 5, # columns = 5, # steps = 5

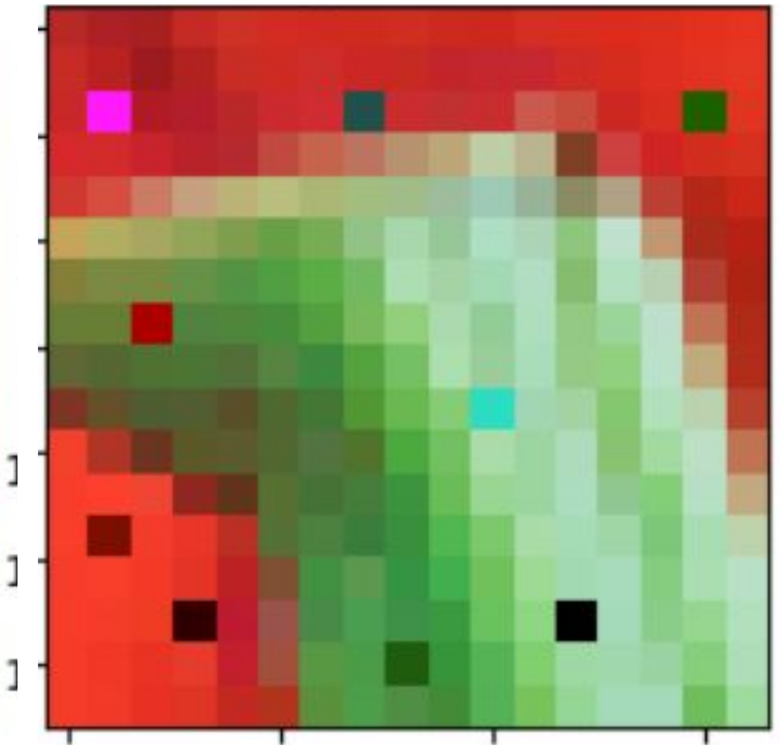
Clifford scores



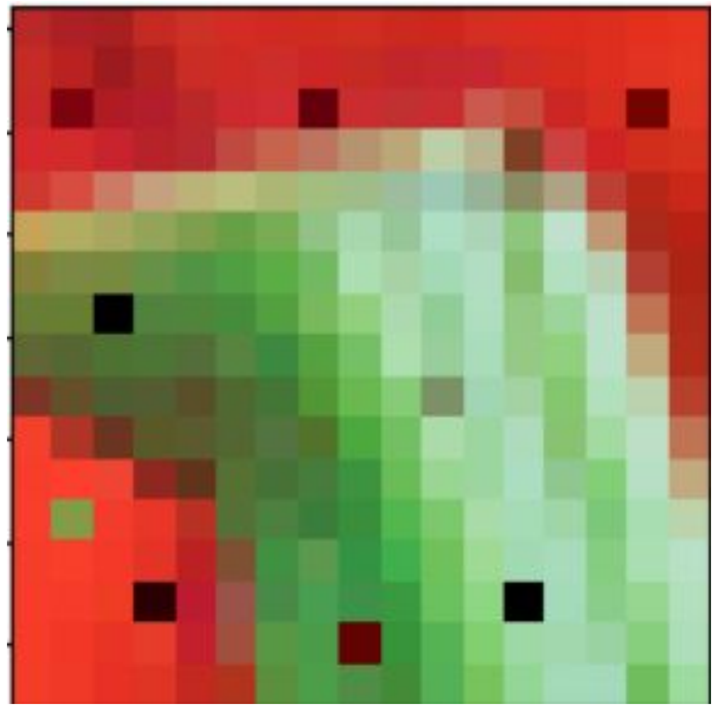
Uniform



Deterministic max norm

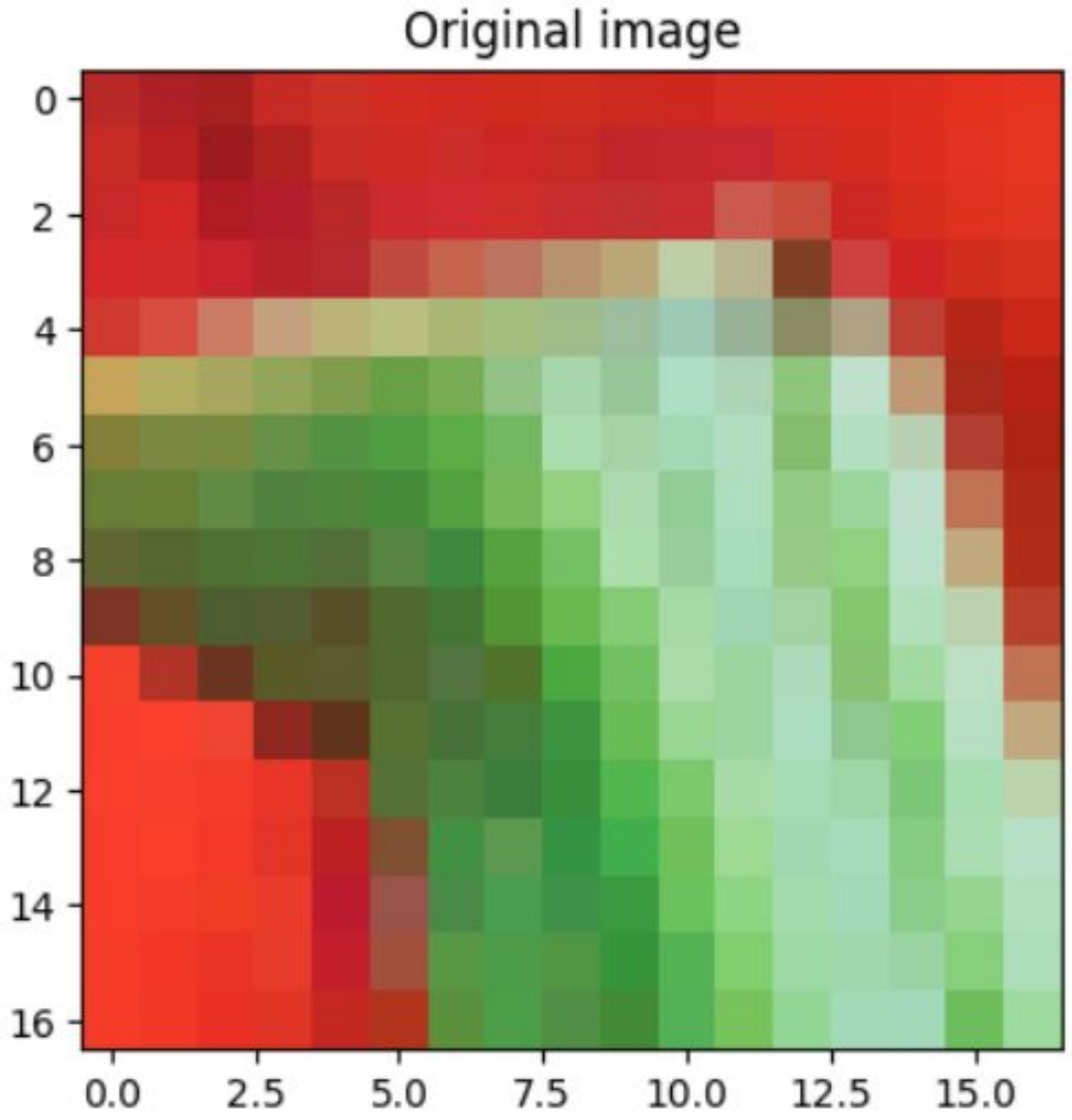


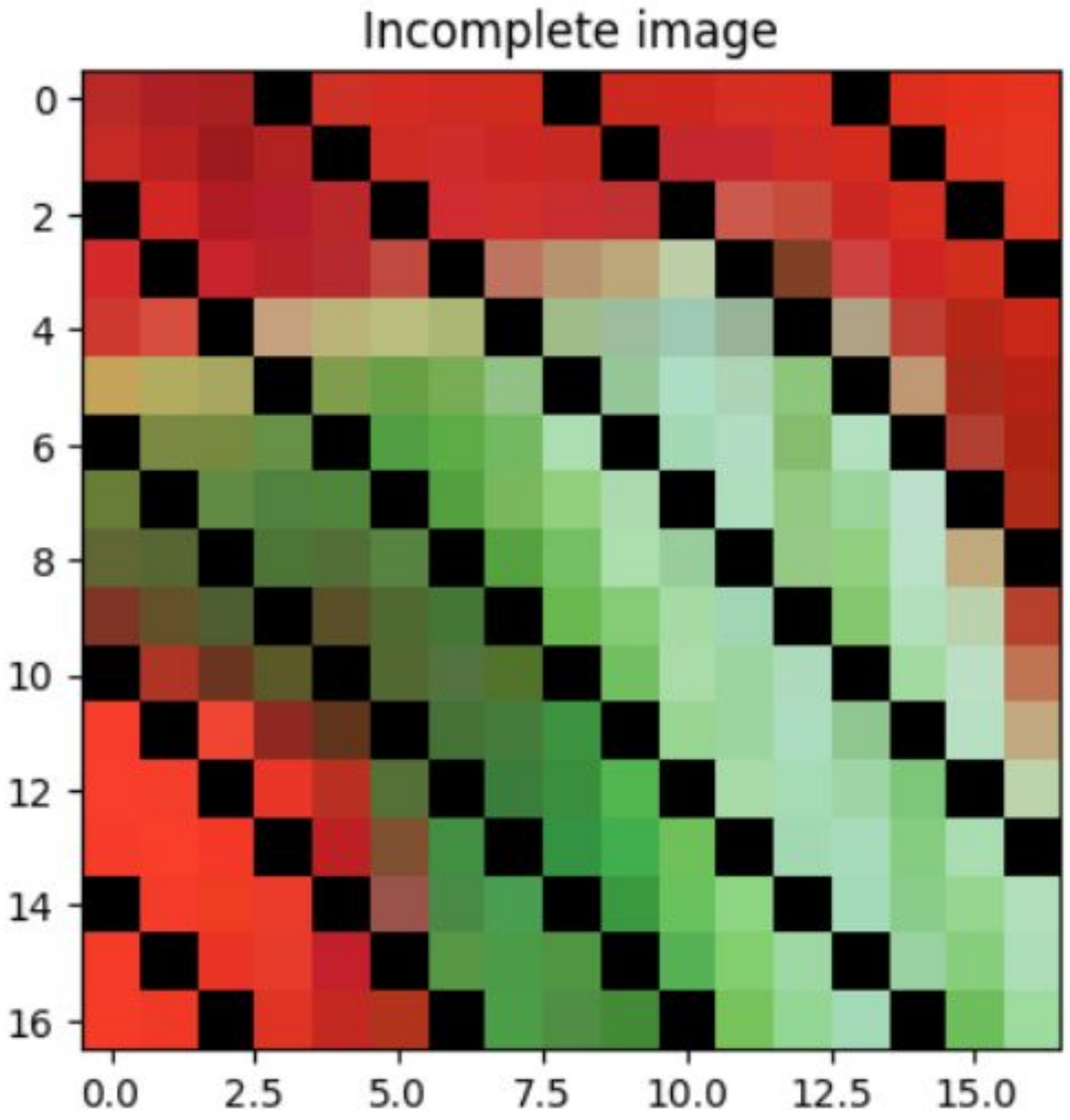
Random max norm



Skoltech

Experiment 4: Image Completion on Peppers





17x17 px

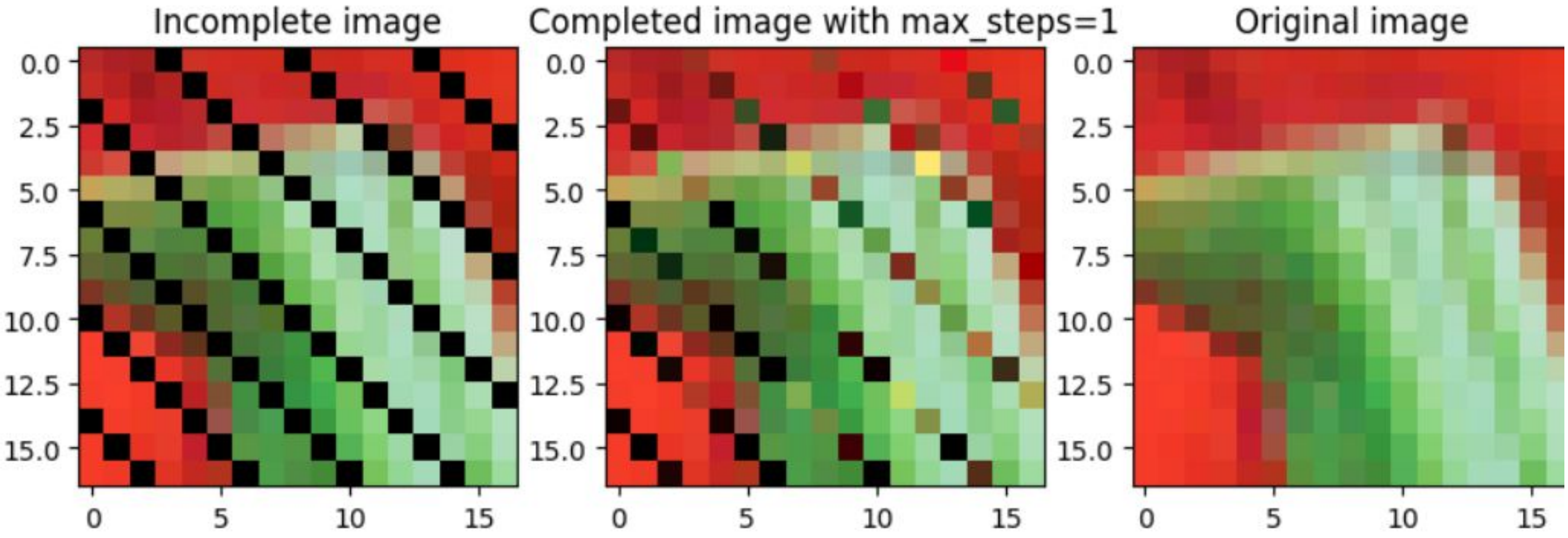
Skoltech

Experiment 4: Image Completion on Peppers

Clifford scores method for CUR

rows = 5, # columns = 5, # steps = 1

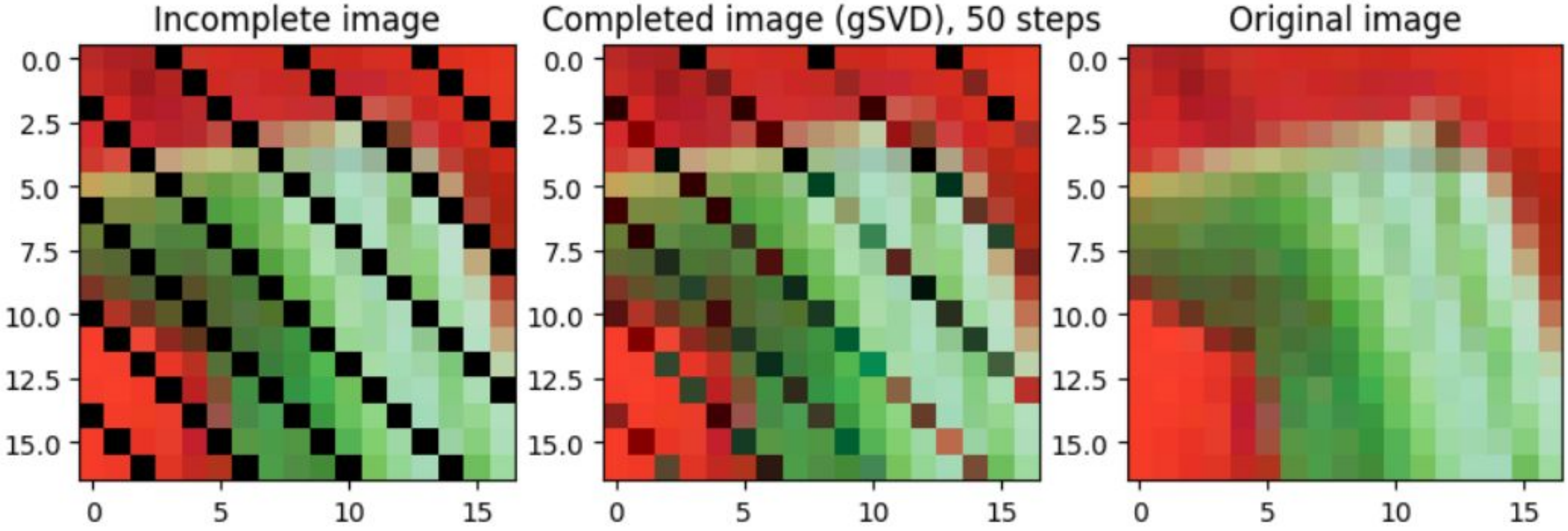
17x17 px



Experiment 4: Image Completion on Peppers

Clifford scores method for CUR
rows = 5, # columns = 5, # steps = 50

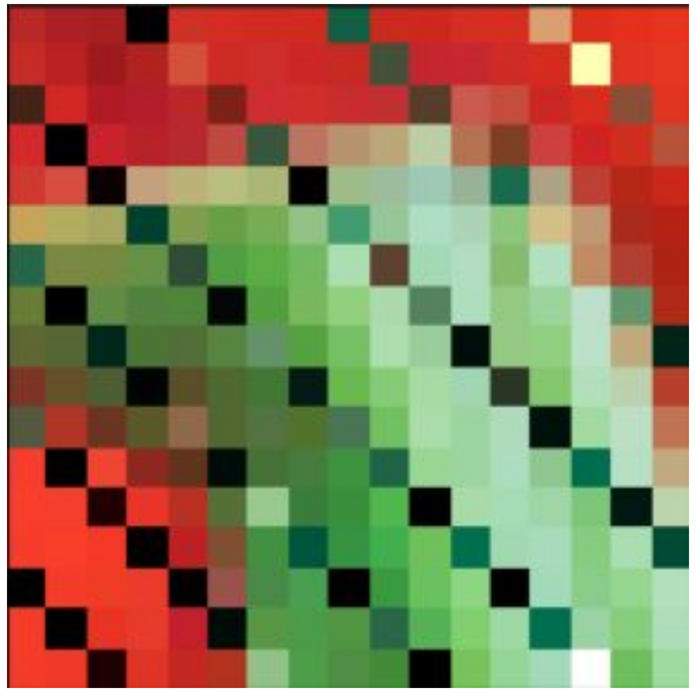
17x17 px



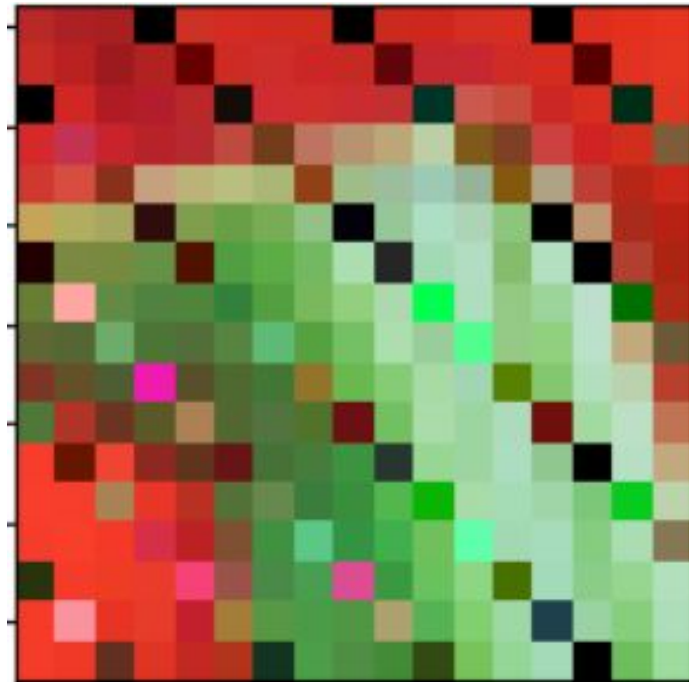
Experiment 4: Image Completion on Peppers

rows = 5, # columns = 5, # steps = 5

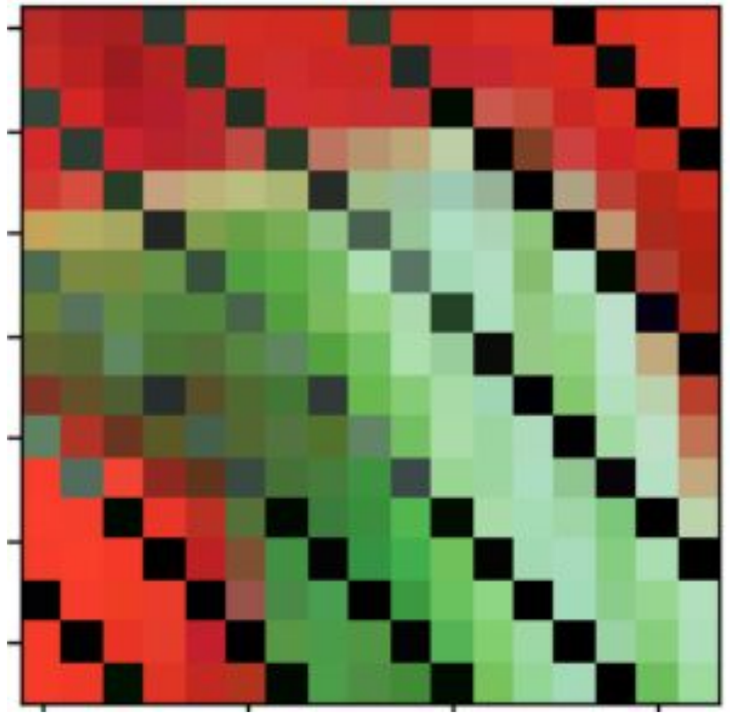
Clifford scores



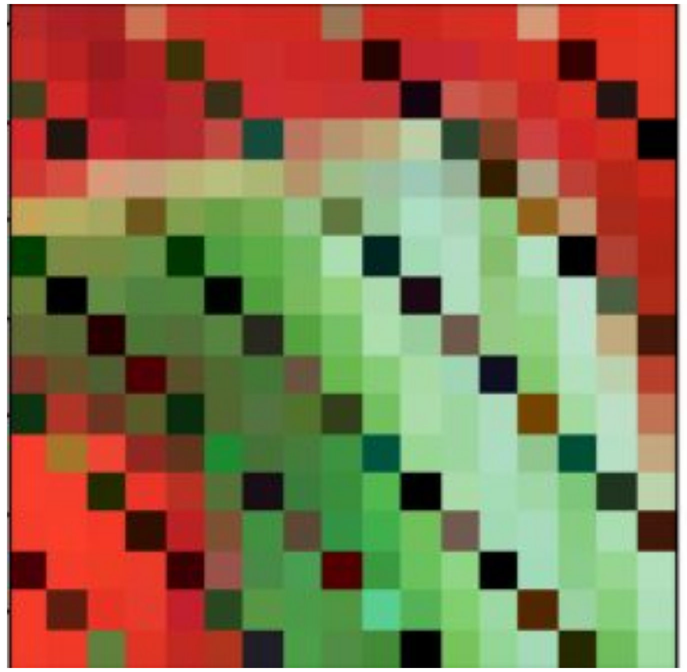
Uniform



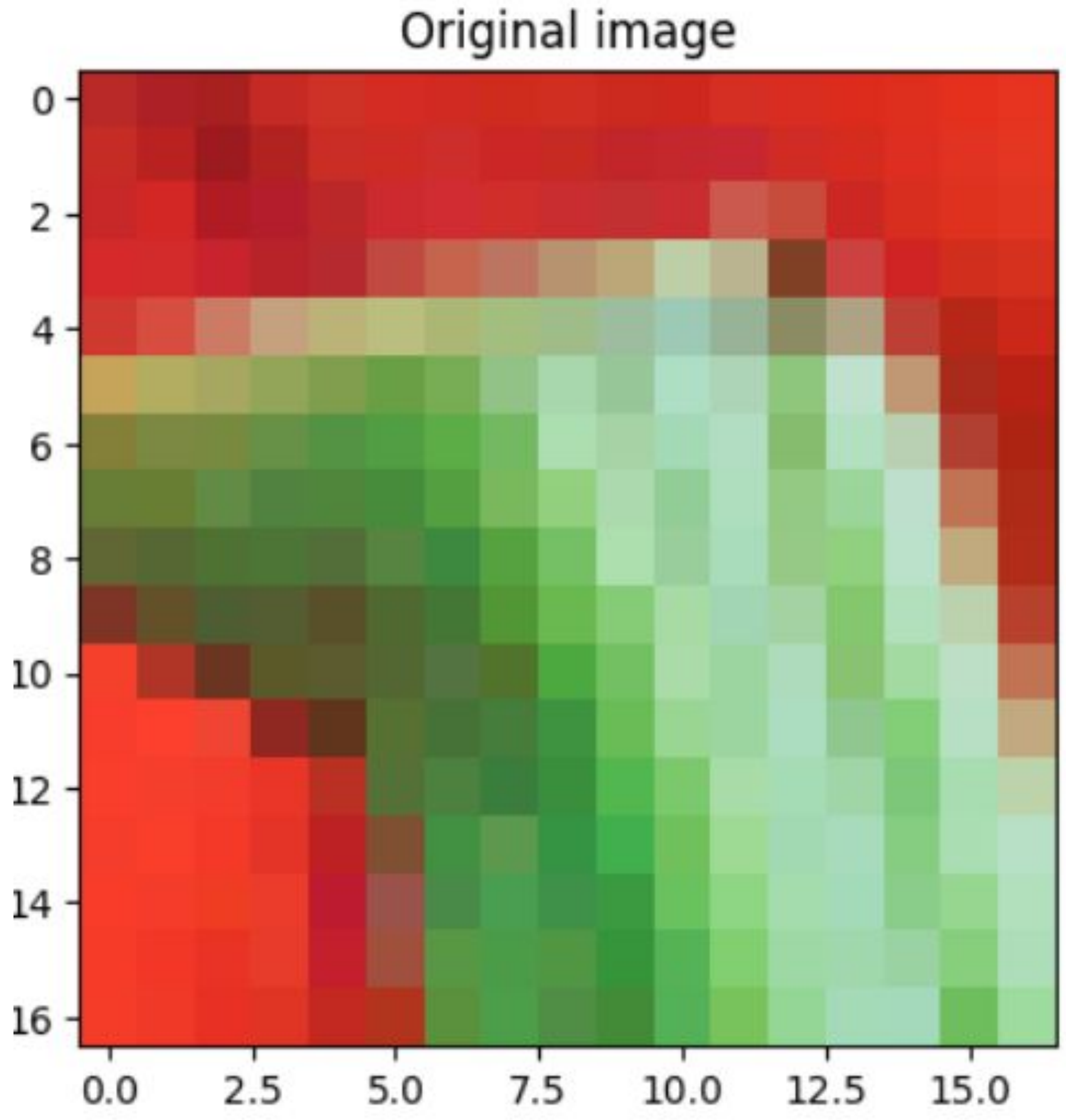
Deterministic max norm

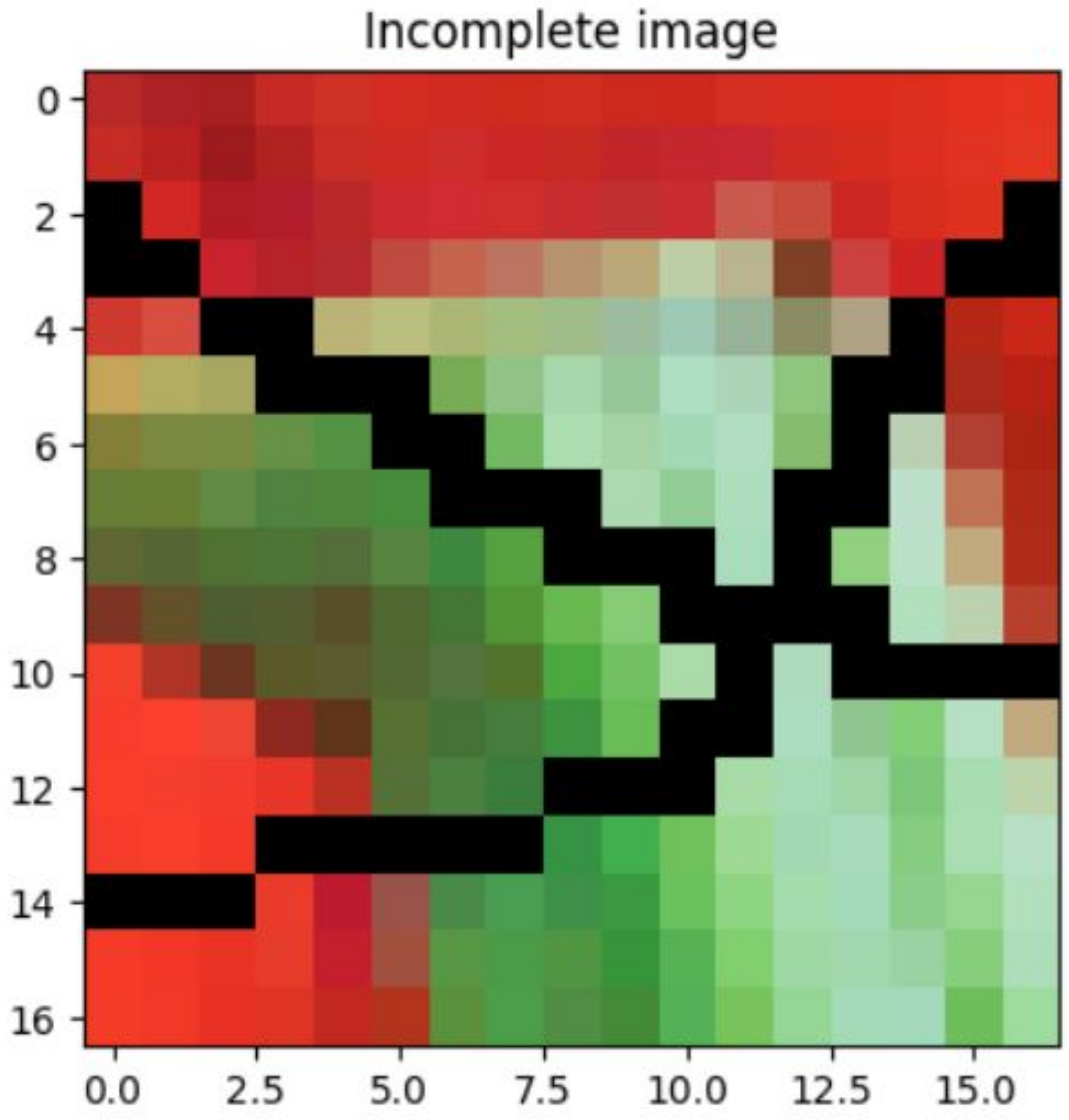


Random max norm



Experiment 5: Image Completion on Peppers





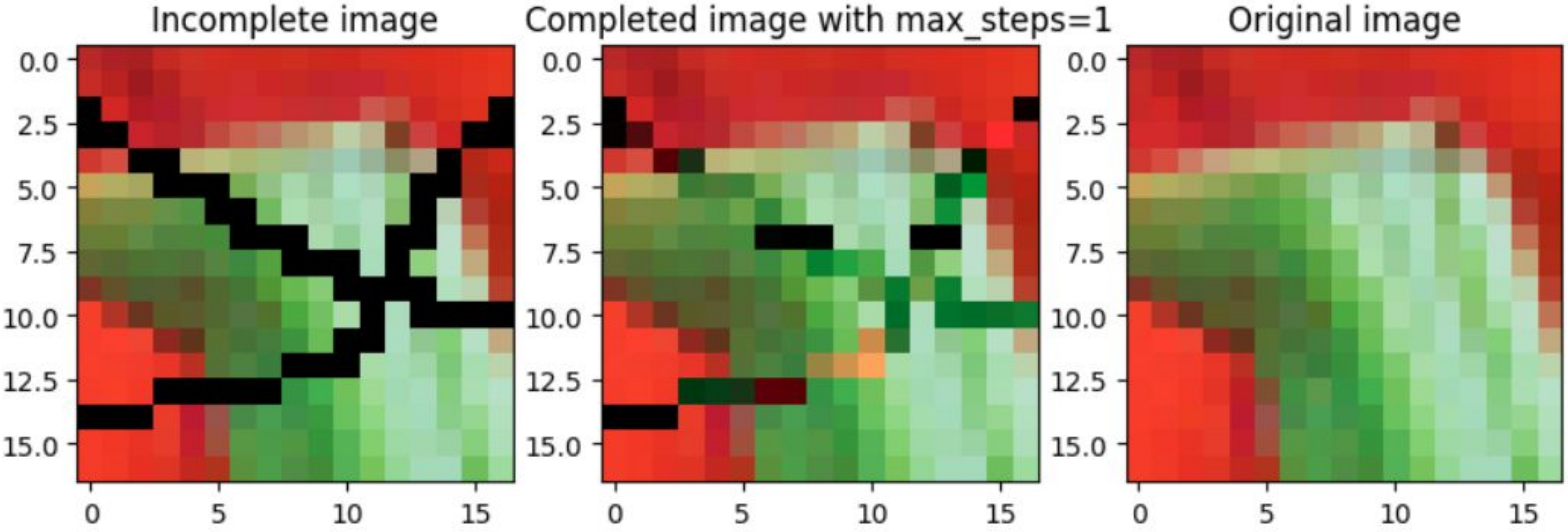
17x17 px

Skoltech

Experiment 5: Image Completion on Peppers

Clifford scores method for CUR
rows = 5, # columns = 5, # steps = 1

17x17 px

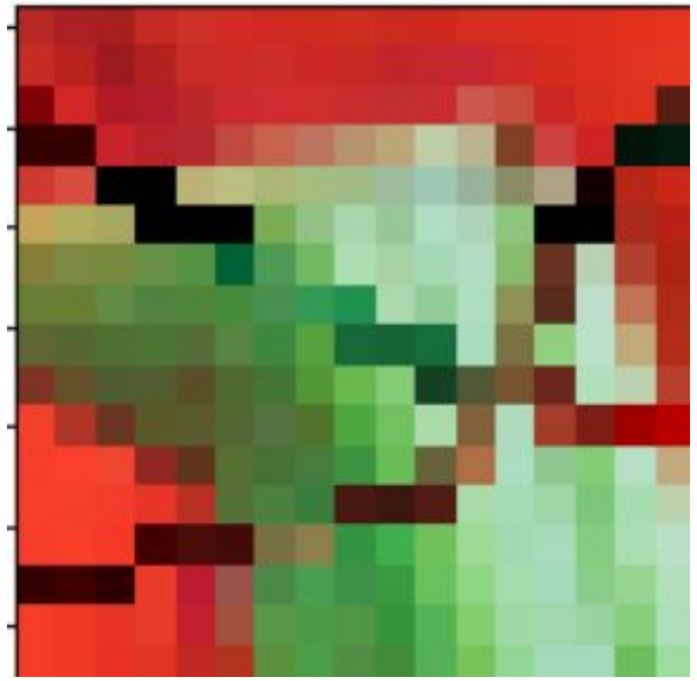


Skoltech

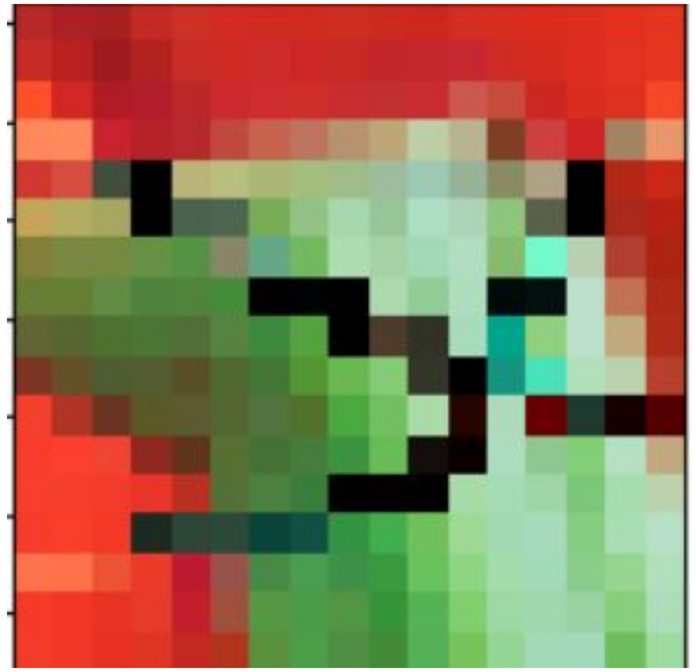
Experiment 5: Image Completion on Peppers

rows = 5, # columns = 5, # steps = 5

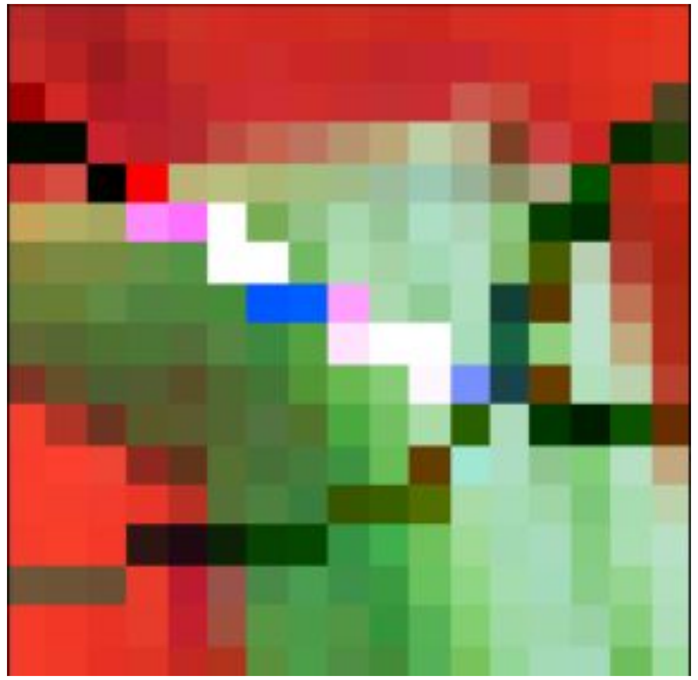
Clifford scores



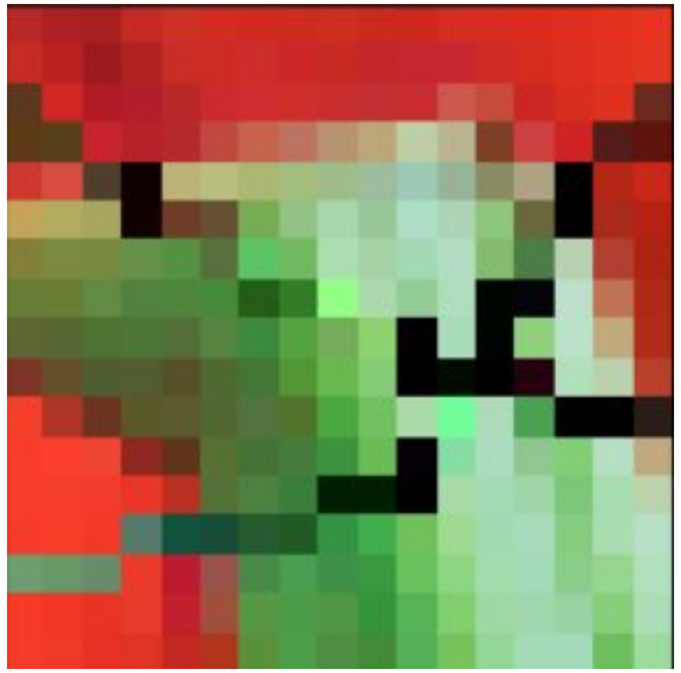
Uniform



Deterministic
max norm



Random
max norm



Thx

Skoltech