vmmlib

Tensor Approximation Classes

Susanne K. Suter

https://github.com/VMML/vmmlib

Outline

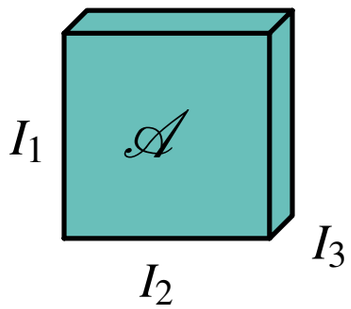
- Part 1: Vmmlib data structures
- Part 2: TA Models
- Part 3: Typical TA algorithms and operations

Downloads and Resources

Test Volumes

- MicroCT test volumes
- http://www.ifi.uzh.ch/vmml/research/datasets.html

Test Dataset: Hazelnut



- A microCT scan of dried hazelnuts
- $I_1 = I_2 = I_3 = 512$
- Values: unsigned char (8bit)
- http://www.ifi.uzh.ch/vmml/research/datasets.html

TA Tutorial

- Tutorial on Tensor Approximation in Visualization and Computer Graphics
  - http://www.ifi.uzh.ch/vmml/links/

Part 1:

Data Structures

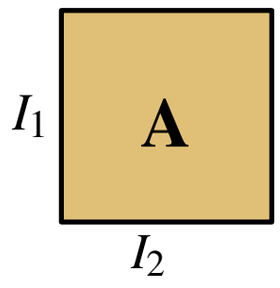
Tensor: A Multidimensional Array

0th-order tensor

1st -order tensor

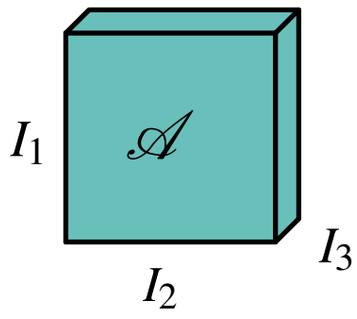
$i_1 = 1, \ldots, I_1$

2nd-order tensor



$i_2 = 1, \ldots, I_2$

3rd-order tensor



$i_3 = 1, \ldots, I_3$

...

A Vector in vmmlib

[vmmlib]

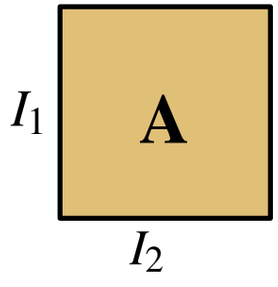
$i_1 = 1, \ldots, I_1$

```
vector< I1, Type >

vector< 4, unsigned char > v;


std::cout << v << std::endl;
```

```
(0, 1, 2)
```

A Matrix in vmmlib

[vmmlib]



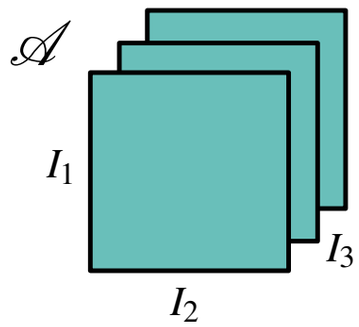
$i_2 = 1, \ldots, I_2$

```
matrix< I1, I2, Type >

matrix< 4, 3, unsigned char > m;

std::cout << m << std::endl;

(0, 1, 2)
(3, 4, 5)
(6, 7, 8)
(9, 10, 11)
```

- I_1 rows
- I_2 columns
- The matrices are per default column-major ordered
- A matrix is an array of I_2 columns, where each column is of size I_1

A Tensor3 in vmmlib

[vmmlib]



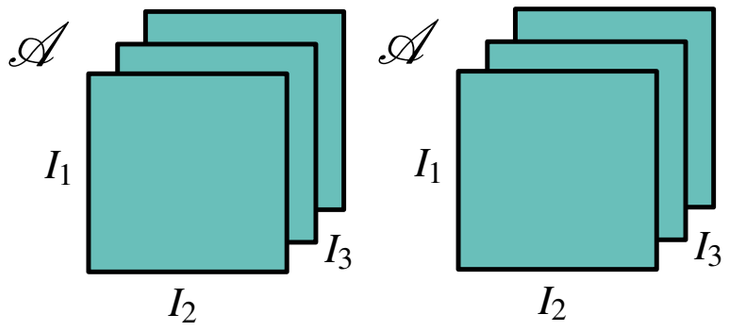

```
tensor3< I1, I2, I3, Type >

tensor3< 4, 3, 2, unsigned char > t3;

std::cout << t3 << std::endl;
```

```
(0, 1, 2)
(3, 4, 5)
(6, 7, 8)
(9, 10, 11)
 ***
(12, 13, 14)
(15, 16, 17)
(18, 19, 20)
(21, 22, 23)
 ***
```

- A tensor3 A in vmmlib is an array of I_3 matrices each of size I_1 times I_2
- A tensor3 is internally allocated and deallocated as pointer

A Tensor4 in vmmlib

[vmmlib]

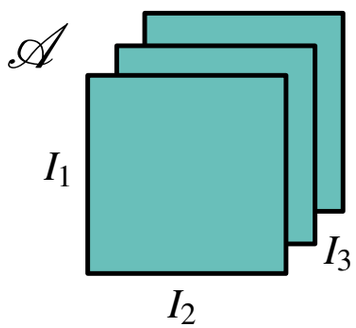



```
tensor4< I1, I2, I3, I4, Type >

tensor4< 4, 3, 2, 2, unsigned char > t4;

 std::cout << t3 << std::endl;
```

```
Example:
(0, 1, 2)
(3, 4, 5)
(6, 7, 8)
(9, 10, 11)
 ***
(12, 13, 14)
(15, 16, 17)
(18, 19, 20)
(21, 22, 23)
 ***
---
(24, 25, 26)
...
```

- A tensor4 in vmmlib is an array of I_4 tensor3s

Large Data Tensors (in vmmlib)



[vmmlib]

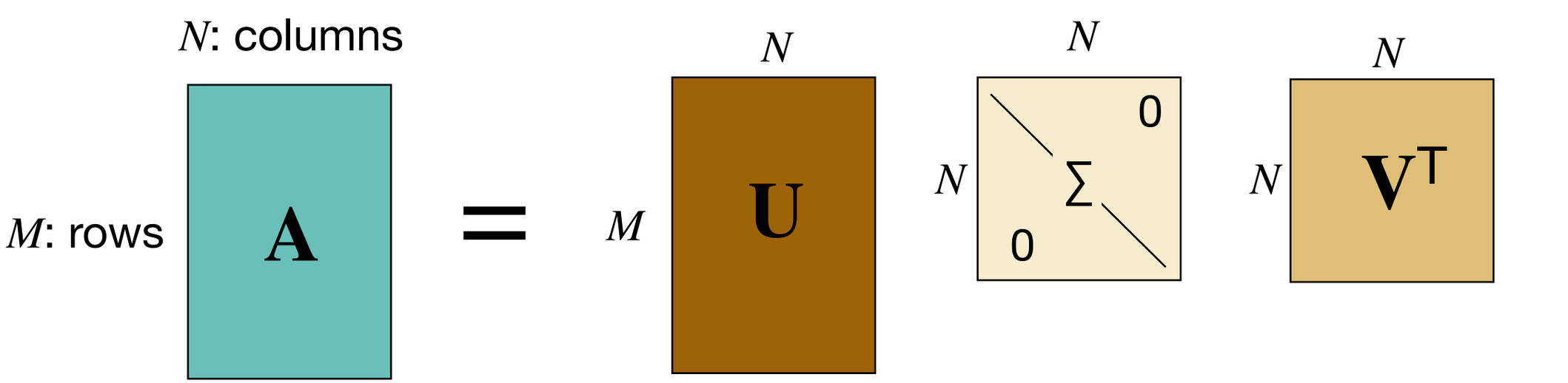
```
const size_t d = 512;
typedef tensor3< d,d,d, unsigned char > t3_512u_t;
typedef t3_converter< d,d,d, unsigned char > t3_conv_t;
typedef tensor_mmapper< t3_512u_t, t3_conv_t > t3map_t;


std::string in_dir = "./dataset";
std::string file_name = "hnut512_uint.raw";
t3_512u_t t3_hazelnut;
t3_conv_t t3_conv;

t3map_t t3_mmap( in_dir, file_name, true, t3_conv ); //true -> read-only
t3_mmap.get_tensor( t3_hazelnut );
```

Part 2:

TA Models

Data Approximation by SVD

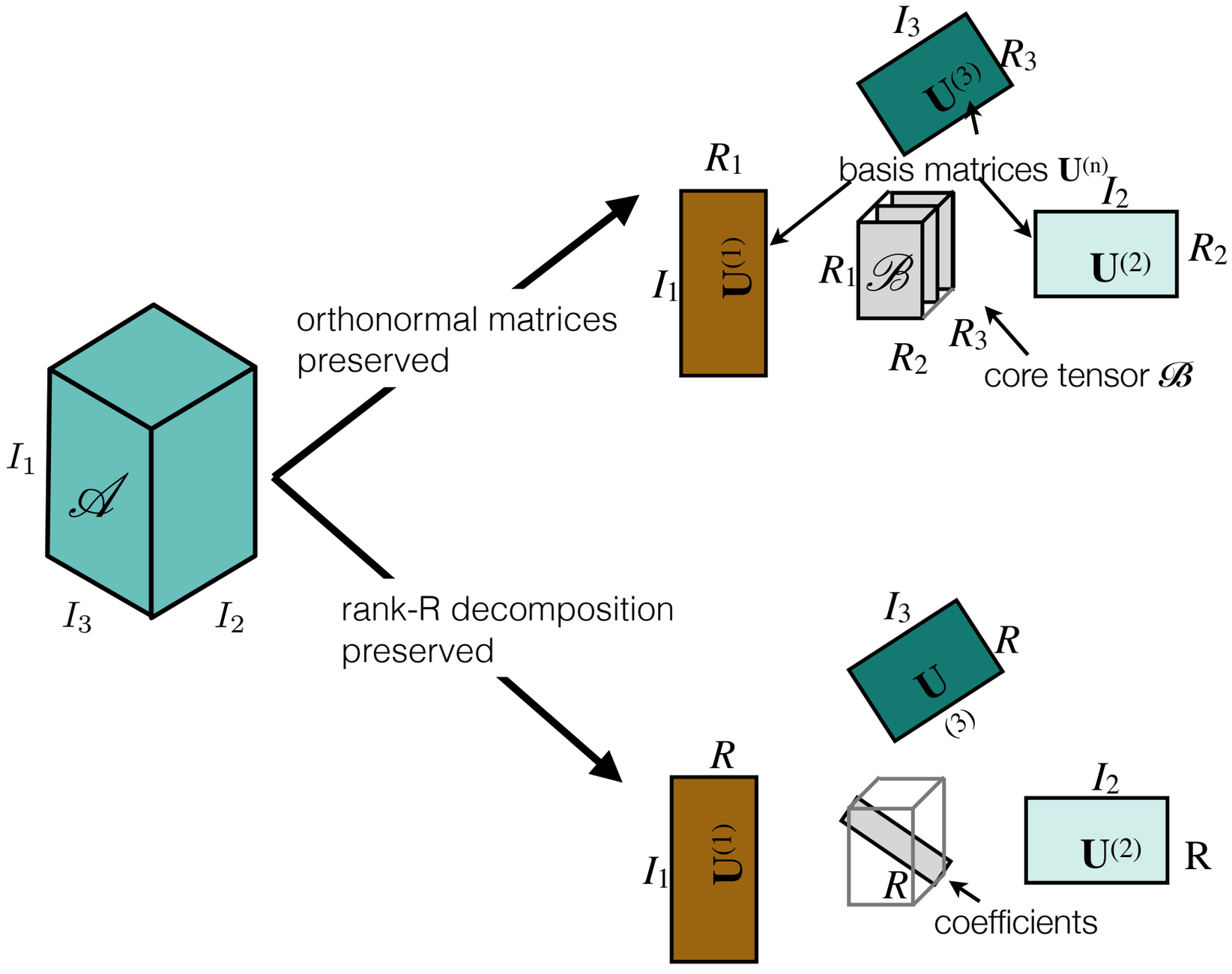
- Singular Value Decomposition (SVD) standard tool for matrices, i.e., 2D input datasets
  - see also principal component analysis (PCA)



Matrix SVD

- Exploit ordered singular values: s1 ≥ s2 ≥ ... ≥ sN
- Select first r singular values (rank reduction)
  - use only bases (singular vectors) of corresponding subspace
- Matrix SVD
  - rank-R decomposition
  - orthonormal row/column matrices

SVD Extension to Higher Orders



Tucker

- Three-mode factor analysis (**3MFA/Tucker3**) [Tucker, 1964+1966]
- Higher-order SVD (**HOSVD**) [De Lathauwer et al., 2000a]

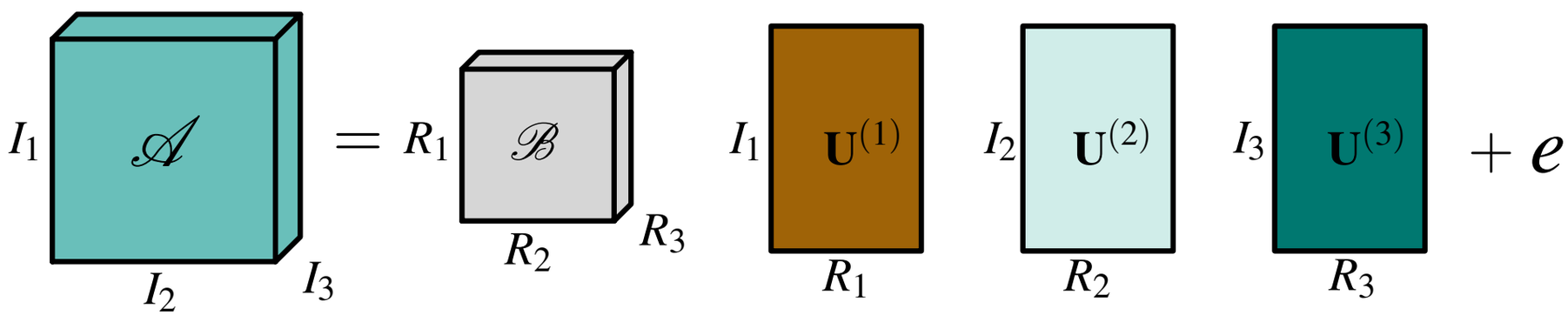
CP

- **PARAFAC** (parallel factors) [Harshman, 1970]
- **CANDECOMP** (CAND) (canonical decomposition) [Caroll & Chang, 1970]

Tucker Model

- Higher order tensor $\mathbf{A} \in \mathbb{R}^{I1 \times \ldots \times I_N}$ represented as a product of a core tensor $\mathscr{B} \in \mathbb{R}^{R1 \times \ldots \times R_N}$ and N factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$
  - using n-mode products $\times_n$

$$\mathscr{A} = \mathscr{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} + \varepsilon$$

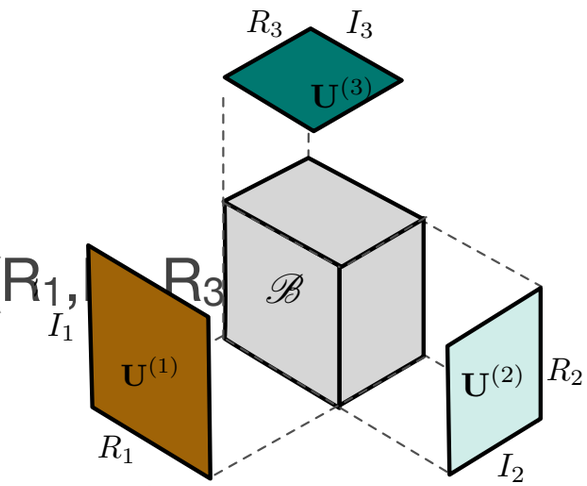


Tucker3 Tensor

[vmmlib]

```
typedef tucker3_tensor< R1, R2, R3, I1, I2, I3, T_value, T_coeff > tucker3_t;
```

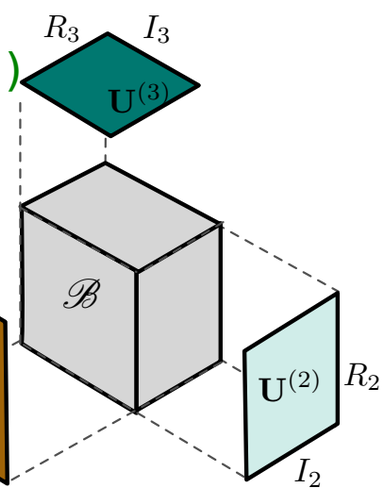
- Define input tensor size (I_1,I_2,I_2)
- Define multilinear rank (R_1,R_2,R_3)
- Define value type and coefficient value type
- Internally always computes with floating point values
- Stores the three factor matrices (I_n x R_n) and the core tensor (R_1, R_3
- ALS (alternating least-squares algorithm):
  - if not converged (fit does not improve anymore, tolerance 1e-04)
  - the ALS stops latest after 10 iteration
- Reconstruction



Example Code Tucker3 Tensor

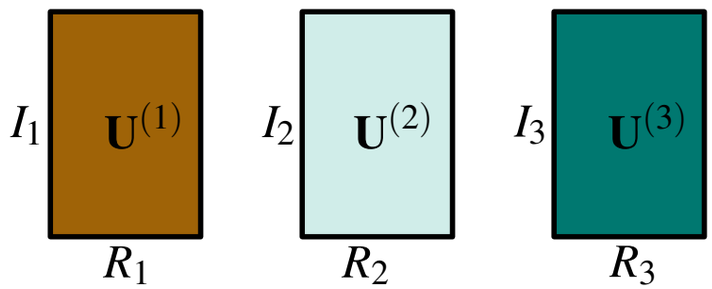
[vmmlib]

```
typedef tensor3< I1, I2, I3, values_t > t3_t;
t3_t t3; //after initializing a tensor3, the tensor is still empty
t3.fill_increasing_values(); //fills the empty tensor with the values 0,1,2,3...

typedef tucker3_tensor< R1, R2, R3, I1, I2, I3, values_t, float > tucker3_t;
tucker3_t tuck3_dec; //empty tucker3 tensor

//choose initialization of Tucker ALS (init_hosvd, init_random, init_dct)
typedef t3_hooi< R1, R2, R3, I1, I2, I3, float > hooi_t;

//Example for initialization with init_rand
tuck3_dec.tucker_als( t3, hooi_t::init_random());

//Example for initialization with init_hosvd
tuck3_dec.tucker_als( t3, hooi_t::init_hosvd());

//Reconstruction
t3_t t3_reco;
tuck3_dec.reconstruct( t3_reco );

//Reconstruction error (RMSE)
double rms_err = = t3.rmse( t3_reco );
```



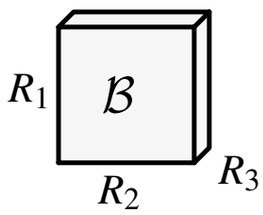
Tucker Tensor-specific Quantization

Suter et al.. Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143, December 2011.



- Factor matrices quantization
  - values between [-1,1]
  - linear quantization

$$\tilde{x}_{\mathbf{U}} = (2^{Q_{\mathbf{U}}} - 1) \cdot \frac{x - x_{min}}{x_{max} - x_{min}}$$



- Core tensor quantization
  - many small values; few large values
  - logarithmic quantization

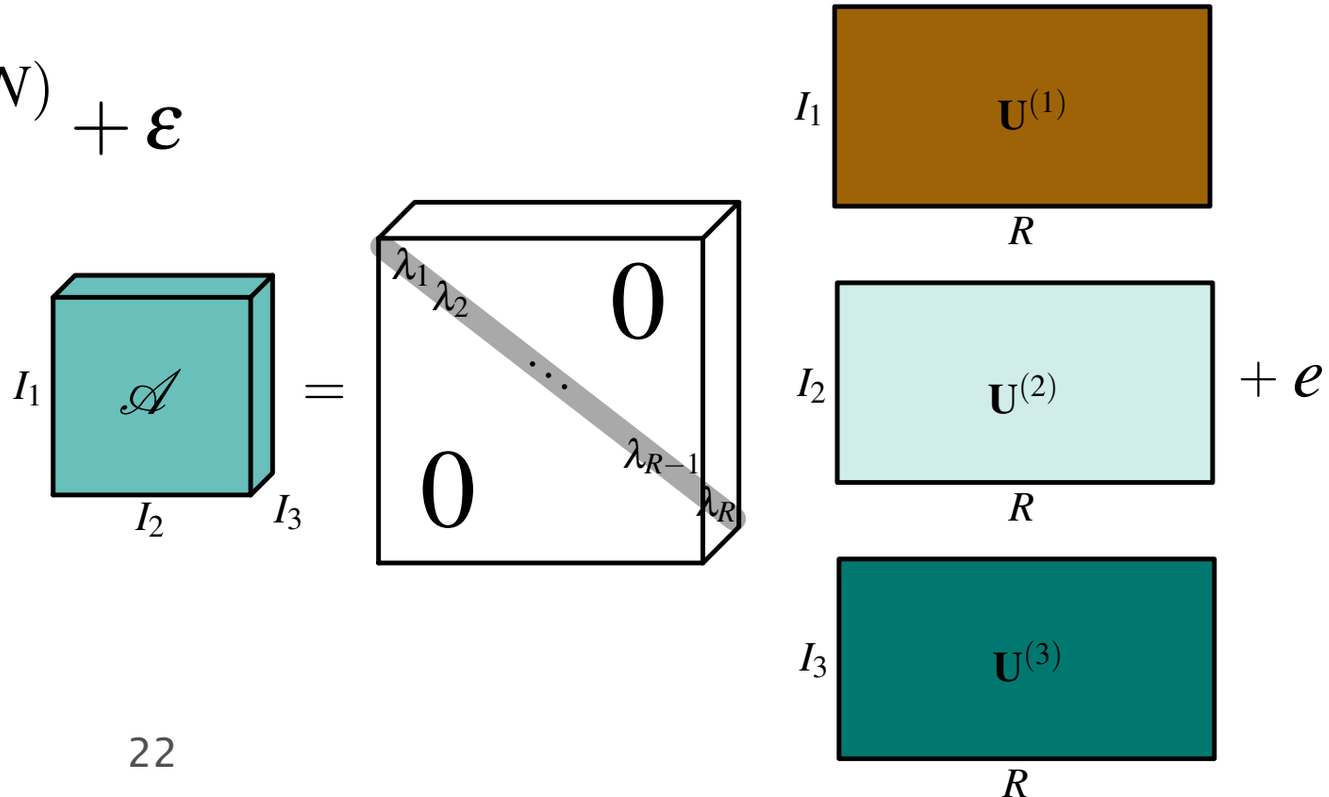
$$|\tilde{x}_{\mathcal{B}}| = (2^{Q_{\mathcal{B}}} - 1) \cdot \frac{\log_2(1 + |x|)}{\log_2(1 + |x_{max}|)}$$

[vmmlib] `typedef qtucker3_tensor< R1, R2, R3, I1, I2, I3, T_value, T_coeff > qtucker3_t;`

CP Model

- *Canonical decomposition* or *parallel factor analysis* model (CP)
- Higher order tensor $\mathbf{A}$ factorized into a sum of rank-one tensors
  - normalized column vectors $\boldsymbol{u}_r^{(n)}$ define factor matrices $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R}$ and weighting factors λ_r

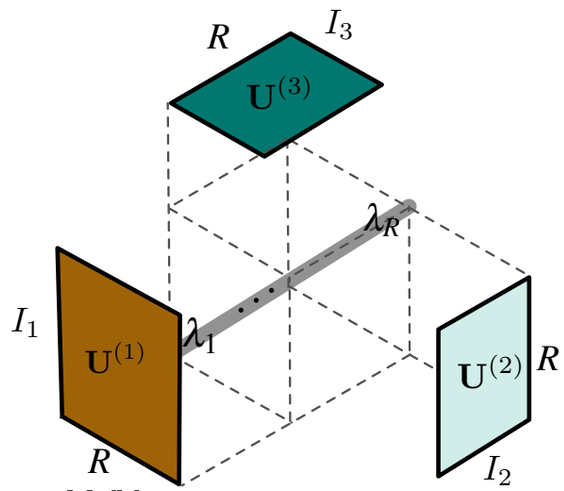
$$\mathscr{A} = \sum_{r=1}^{R} \lambda_r \cdot \boldsymbol{u}_r^{(1)} \circ \boldsymbol{u}_r^{(2)} \circ \ldots \boldsymbol{u}_r^{(N)} + \boldsymbol{\varepsilon}$$



CP3 Tensor

[vmmlib]

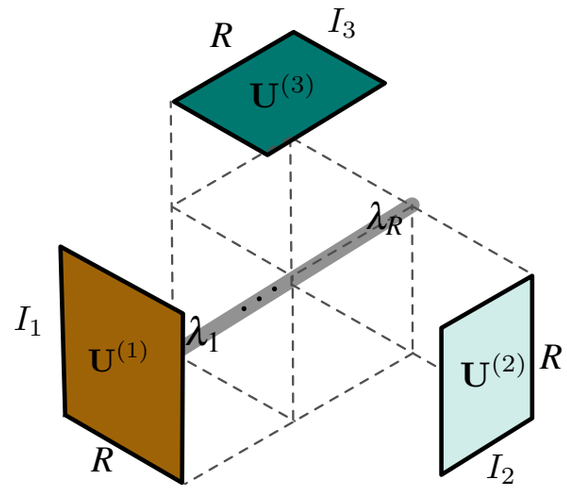
- Define input tensor size (I_1,I_2,I_2)
- Define rank R
- Define value type and coefficient value type
- Internally always computes with floating point values
- Stores three factor matrices each of size (I_n x R) and the lamb
- ALS:
  - if not converged (fit does not improve anymore, tolerance 1e-04)
  - set number of maximum CP ALS iterations
- Reconstruction



Code Example CP3 Tensor

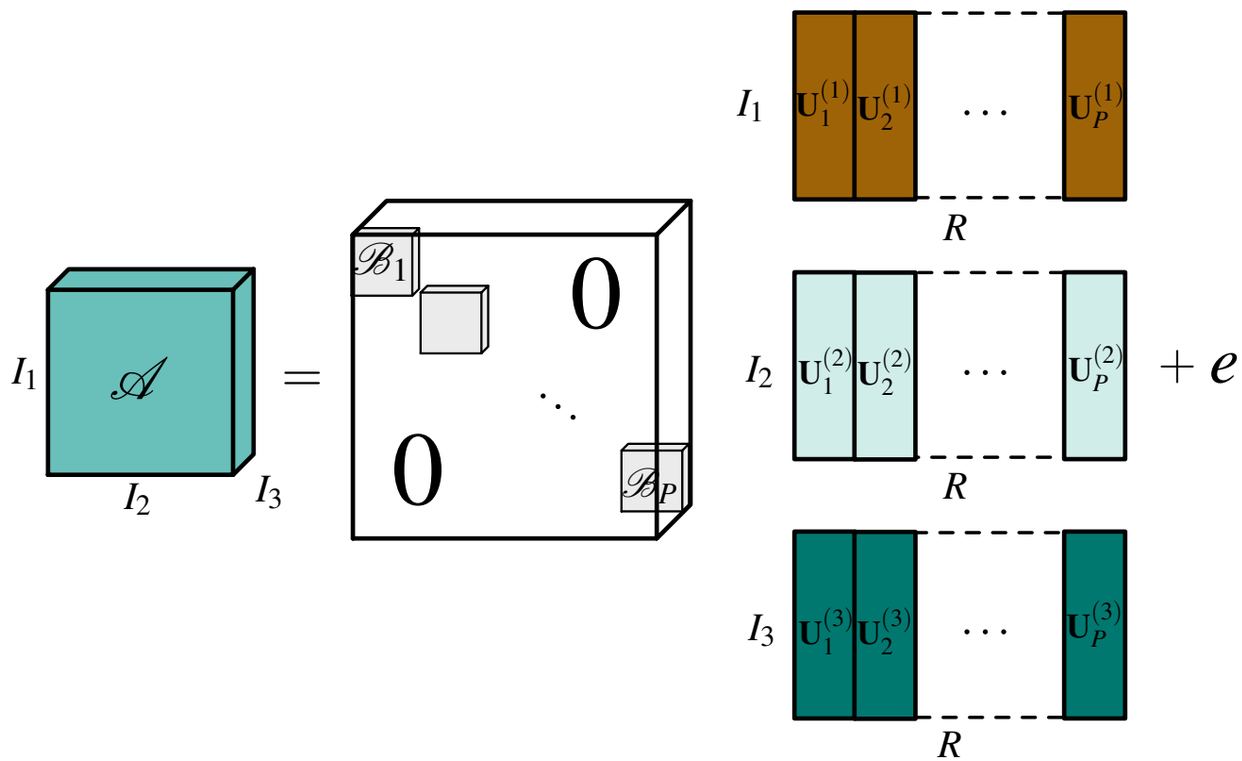
[vmmlib]

```
typedef cp3_tensor< r, a, b, c, values_t, float > cp3_t;
typedef t3_hopm< r, a, b, c, float > t3_hopm_t;

cp3_t cp3_dec;

//Decomposition or CP ALS
//choose initialization of Tucker ALS (init_hosvd, init_random, init_dct)

int max_cp_iter = 20;
cp3_dec.cp_als( t3, t3_hopm_t::init_random(), max_cp_iter );

//Reconstruction
t3_t t3_cp_reco;
cp3_dec.reconstruct( t3_cp_reco );

//Reconstruction error (RMSE)
rms_err = t3.rmse( t3_cp_reco ) ;
```



Generalization

- Any special form of core and corresponding factor matrices
  - e.g. blocks along diagonal
- Incremental methods
  - see: ihopm, ihooi
  - Code examples to be done...

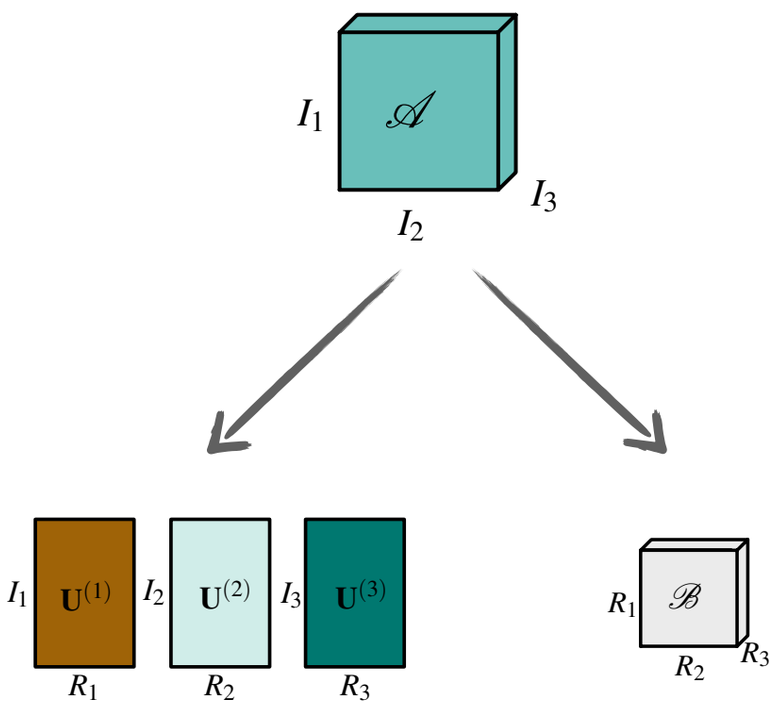


Part 3:
Typical TA Algorithms and Operations

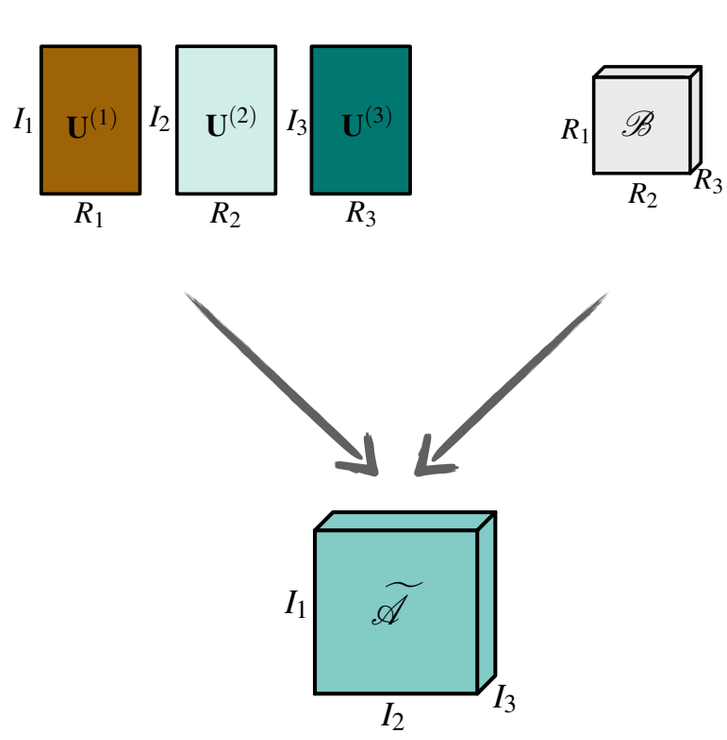
Typical TA Operations

tensor approximation

tensor decomposition

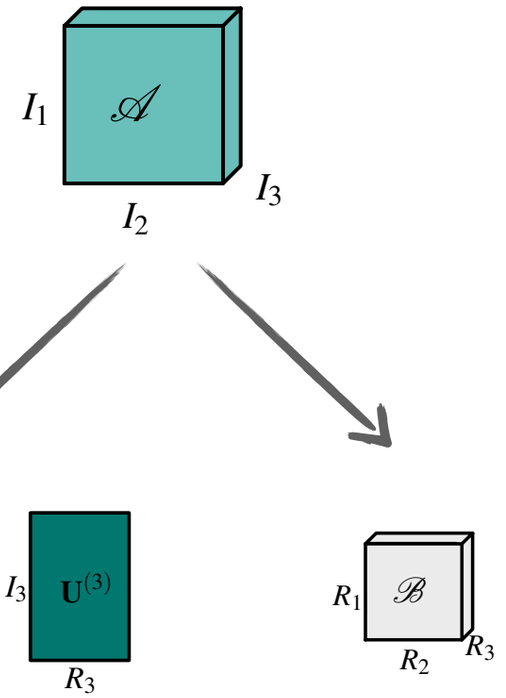


real-time
tensor reconstruction



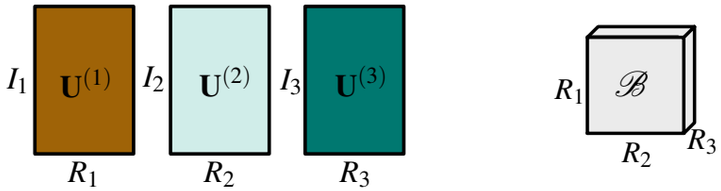
Tensor Decomposition

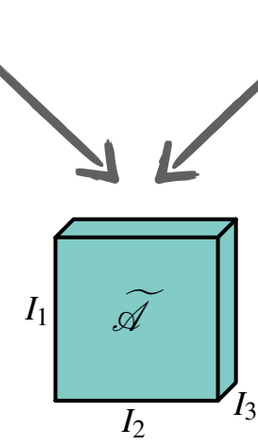
- Create factor matrices
  - Higher-order SVD (HOSVD)
    - Tensor unfolding
  - Alternating least-squares (ALS) algorithms
    - Higher-order orthogonal iteration (HOOI)
    - Higher-order power method (HOPM)
- Generate core tensor
  - Tensor times matrix (TTM) multiplications



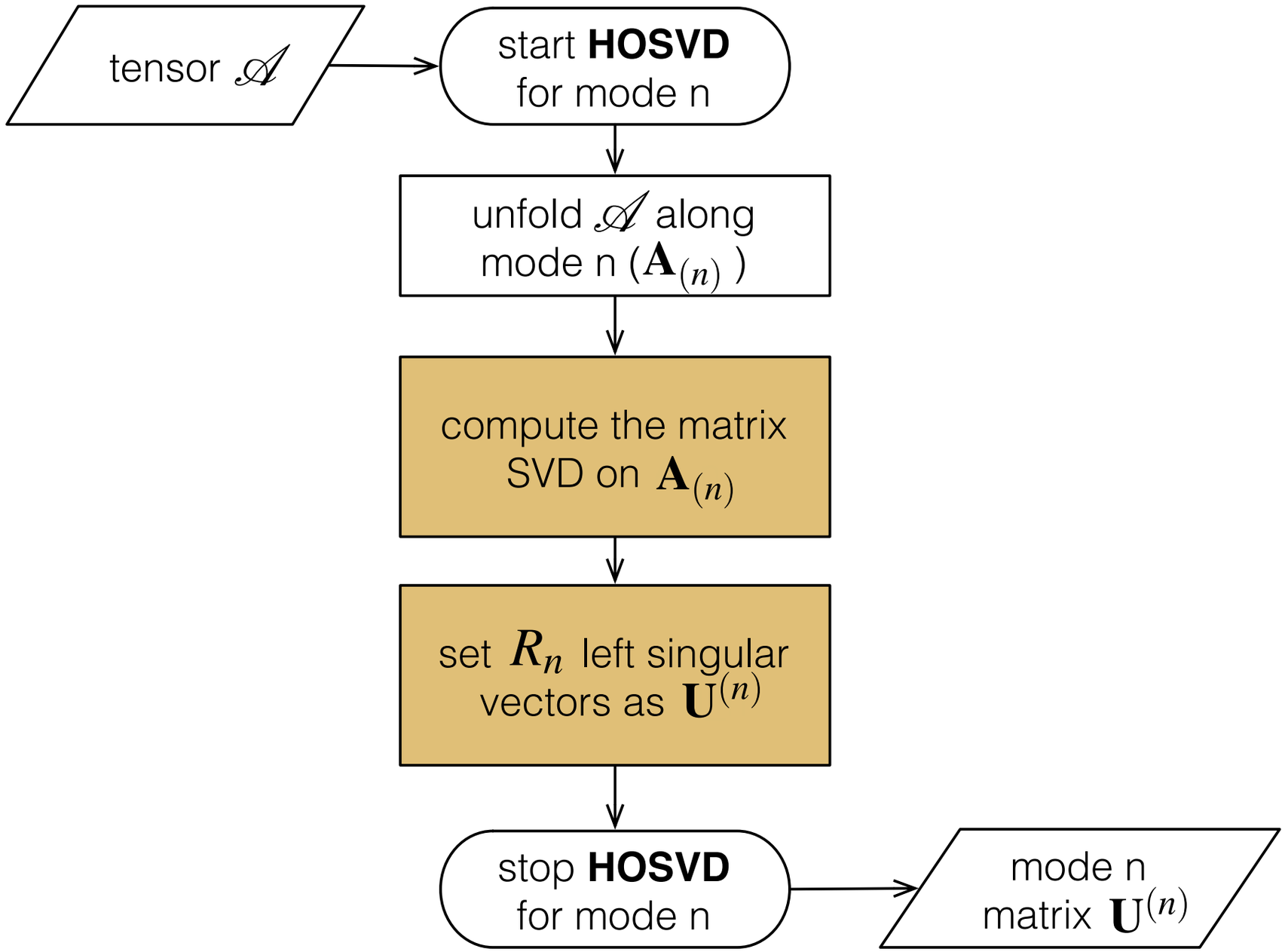
Tensor Reconstruction

- Reconstruction
  - Tensor times matrix (TTM) multiplications





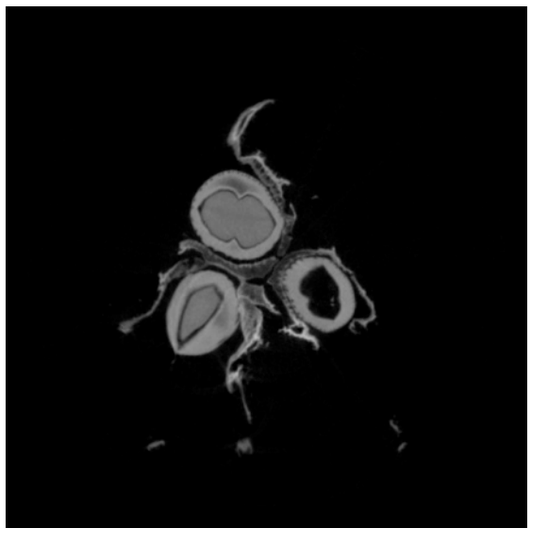
Higher-order SVD (HOSVD)



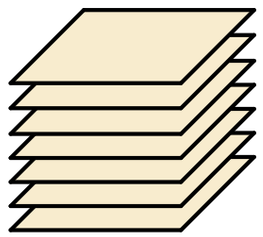
De Lathauwer, de Moor, Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

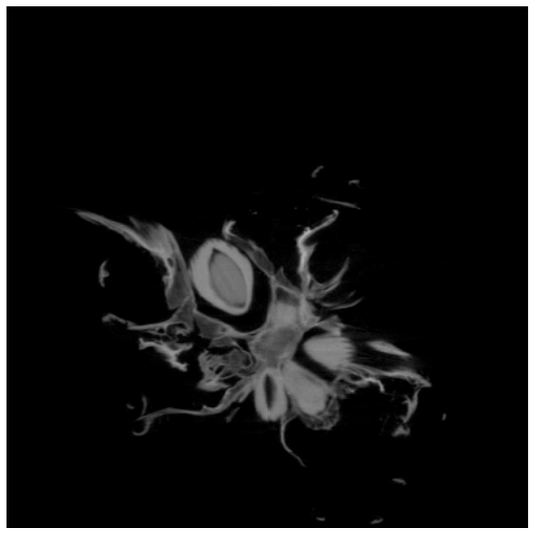
Slices of a Tensor3

frontal slices

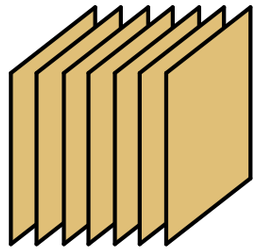


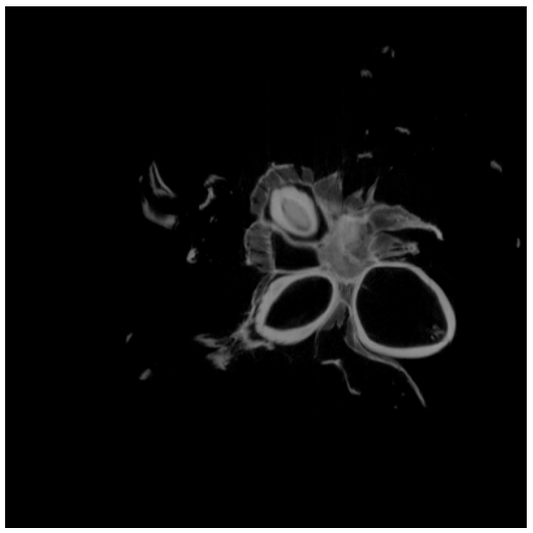
horizontal slices





lateral slices



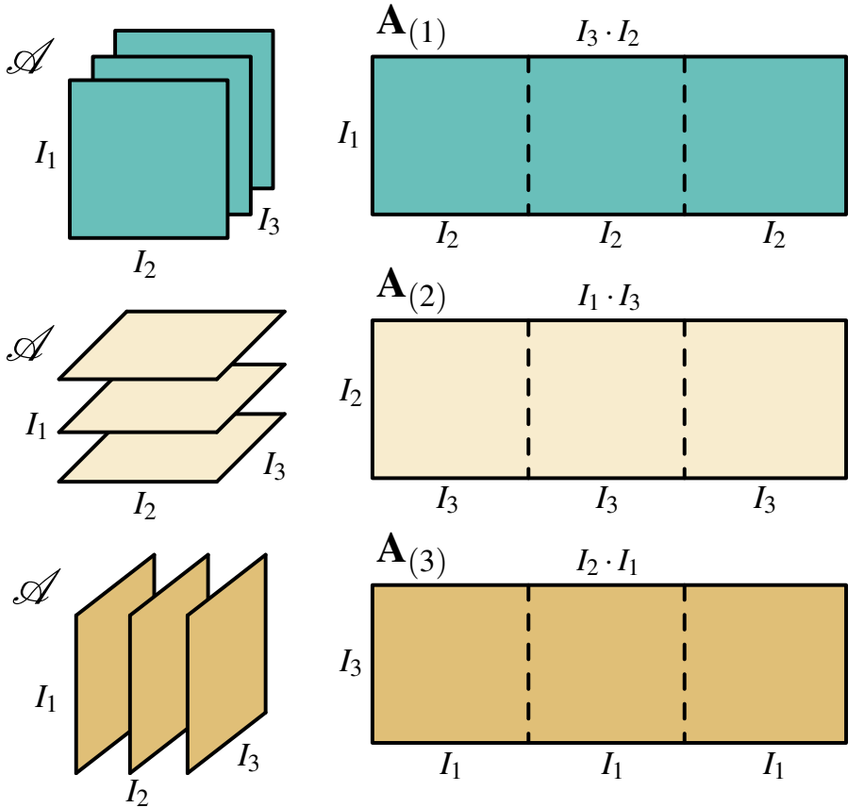


[vmmlib]

```
matrix< 512, 512, values_t > slice;
t3.get_frontal_slice_fwd( 256, slice );
t3.get_horizontal_slice_fwd( 256, slice );
t3.get_lateral_slice_fwd( 256, slice );
```

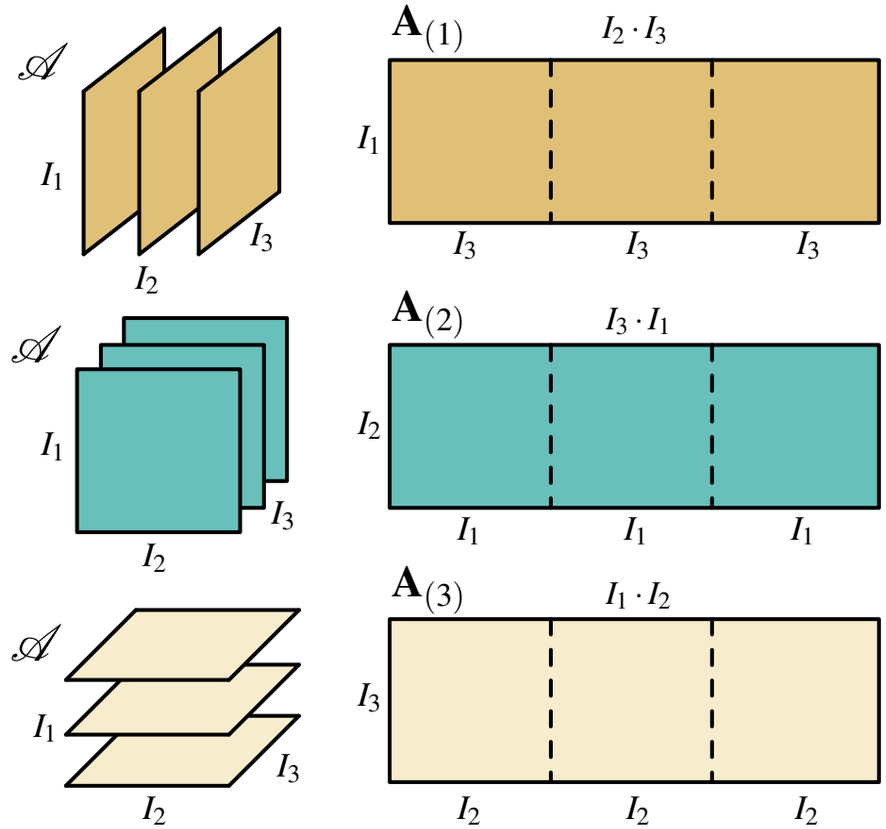
Tensor Unfolding (Matricization)

forward cyclic unfolding



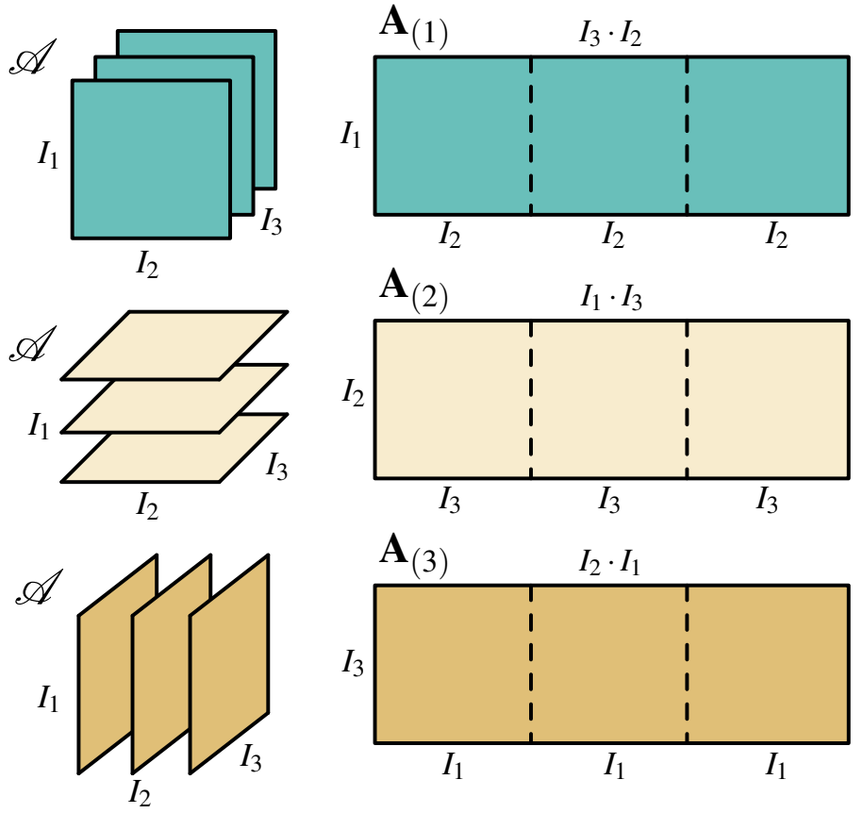
Kiers. Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3):105–122, 2000.

backward cyclic unfolding



De Lathauwer, de Moor, Vandewalle. A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

Forward Cyclic Unfolding




```
tensor3< I1, I2, I3, values_t > t3
matrix< I1, I3*I2, values_t > unf_front_fwd;
t3.frontal_unfolding_fwd( unf_front_fwd );
```

```
forward unfolded tensor (frontal)
(0, 1, 2, 12, 13, 14)
(3, 4, 5, 15, 16, 17)
(6, 7, 8, 18, 19, 20)
(9, 10, 11, 21, 22, 23)
```

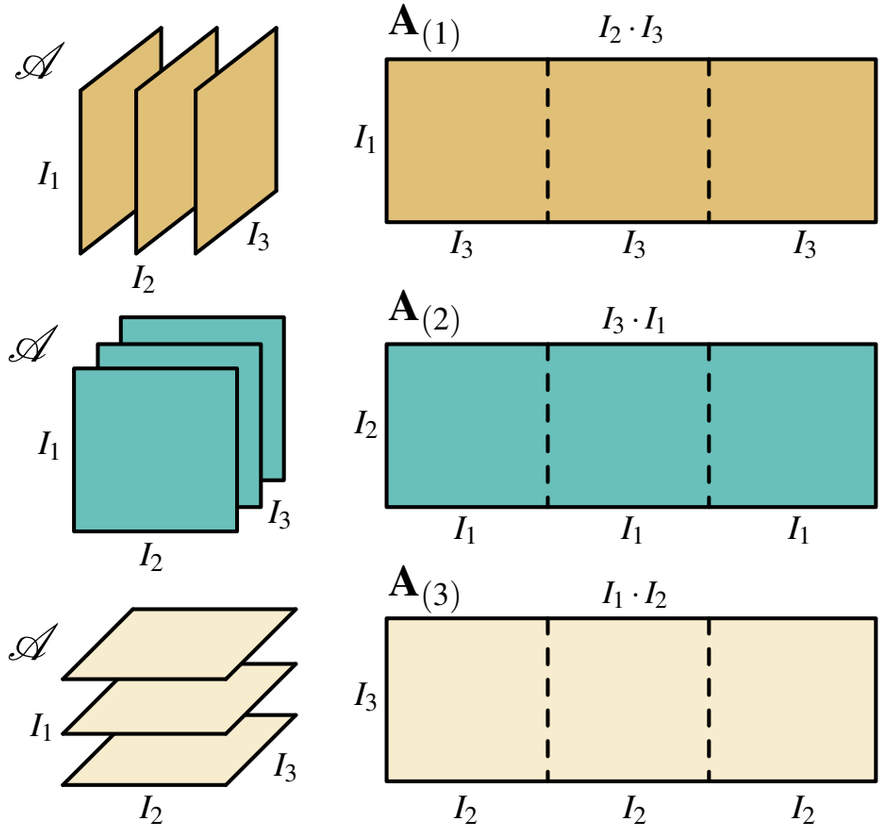
```
matrix< I2, I1*I2, values_t > unf_horiz_fwd;
t3.horizontal_unfolding_fwd( unf_horiz_fwd );
```

```
forward unfolded tensor (horizontal)
(0, 12, 3, 15, 6, 18, 9, 21)
(1, 13, 4, 16, 7, 19, 10, 22)
(2, 14, 5, 17, 8, 20, 11, 23)
```

```
matrix< I3, I2*I1, values_t > unf_lat_fwd;
t3.lateral_unfolding_fwd( unf_lat_fwd );
```

```
forward unfolded tensor (lateral)
(0, 3, 6, 9, 1, 4, 7, 10, 2, 5, 8, 11)
(12, 15, 18, 21, 13, 16, 19, 22, 14, 17, 20, 23)
```

Backward Cyclic Unfolding




```
tensor3< I1, I2, I3, values_t > t3
matrix< I1, I2*I3, values_t > unf_lat_bwd;
t3.lateral_unfolding_bwd( unf_lat_bwd );
```

```
backward unfolded tensor (lateral)
(0, 12, 1, 13, 2, 14)
(3, 15, 4, 16, 5, 17)
(6, 18, 7, 19, 8, 20)
(9, 21, 10, 22, 11, 23)
```

```
matrix< I2, I3*I1, values_t > unf_front_bwd;
t3.frontal_unfolding_bwd( unf_front_bwd );
```

```
backward unfolded tensor (frontal)
(0, 3, 6, 9, 12, 15, 18, 21)
(1, 4, 7, 10, 13, 16, 19, 22)
(2, 5, 8, 11, 14, 17, 20, 23)
```

```
matrix< I3, I1*I2, values_t > unf_horiz_bwd;
t3.horizontal_unfolding_bwd( unf_horiz_bwd );
```

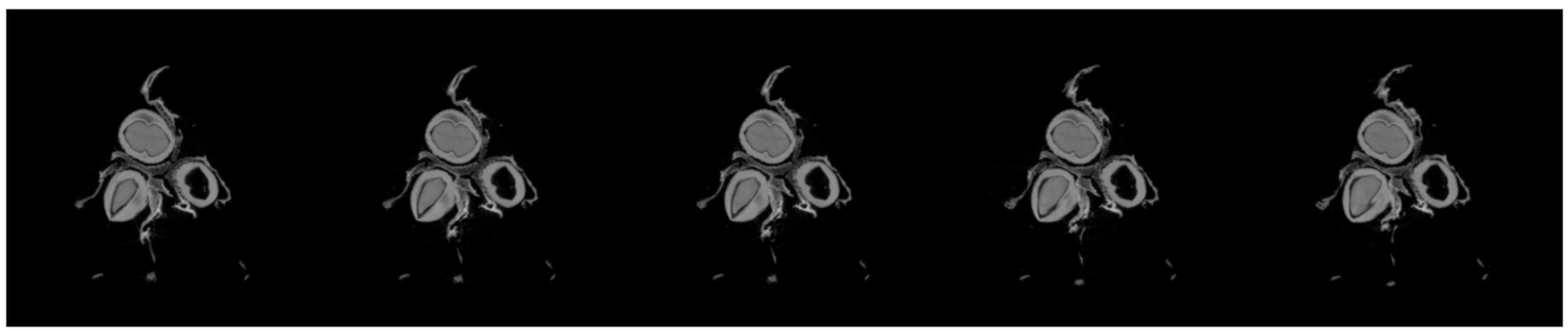
```
backward unfolded tensor (horizontal)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
(12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23)
```

Tensor Unfolding Example

mode-1 unfolding

262’144

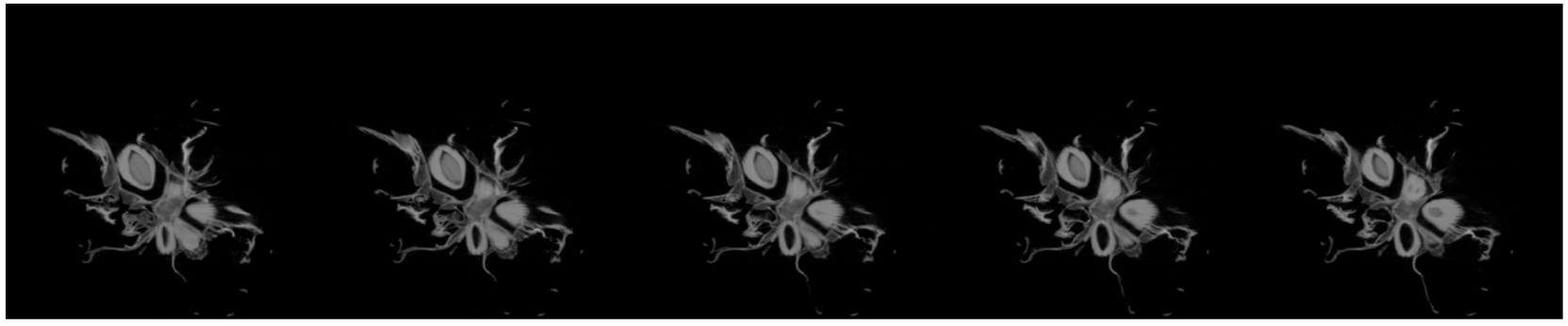
512



262’144

mode-2 unfolding

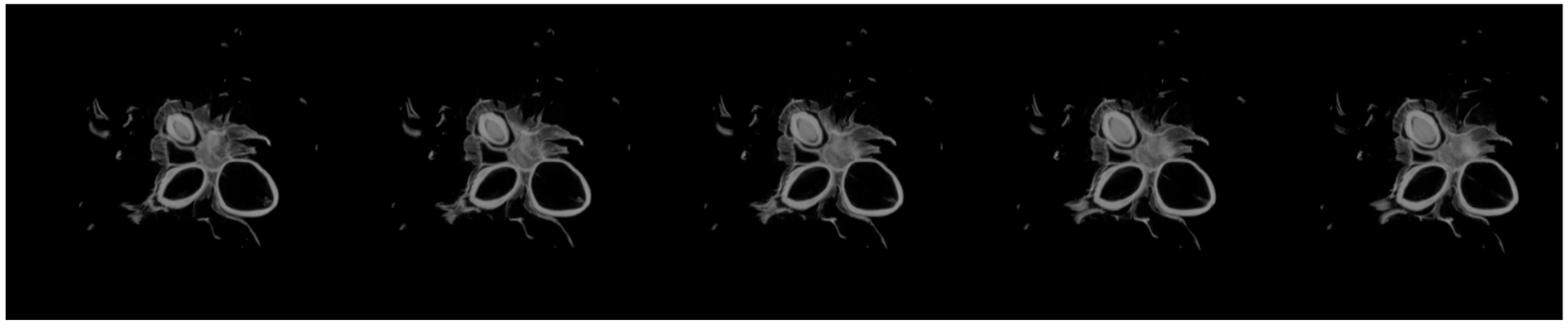
512



262’144

mode-3 unfolding

512

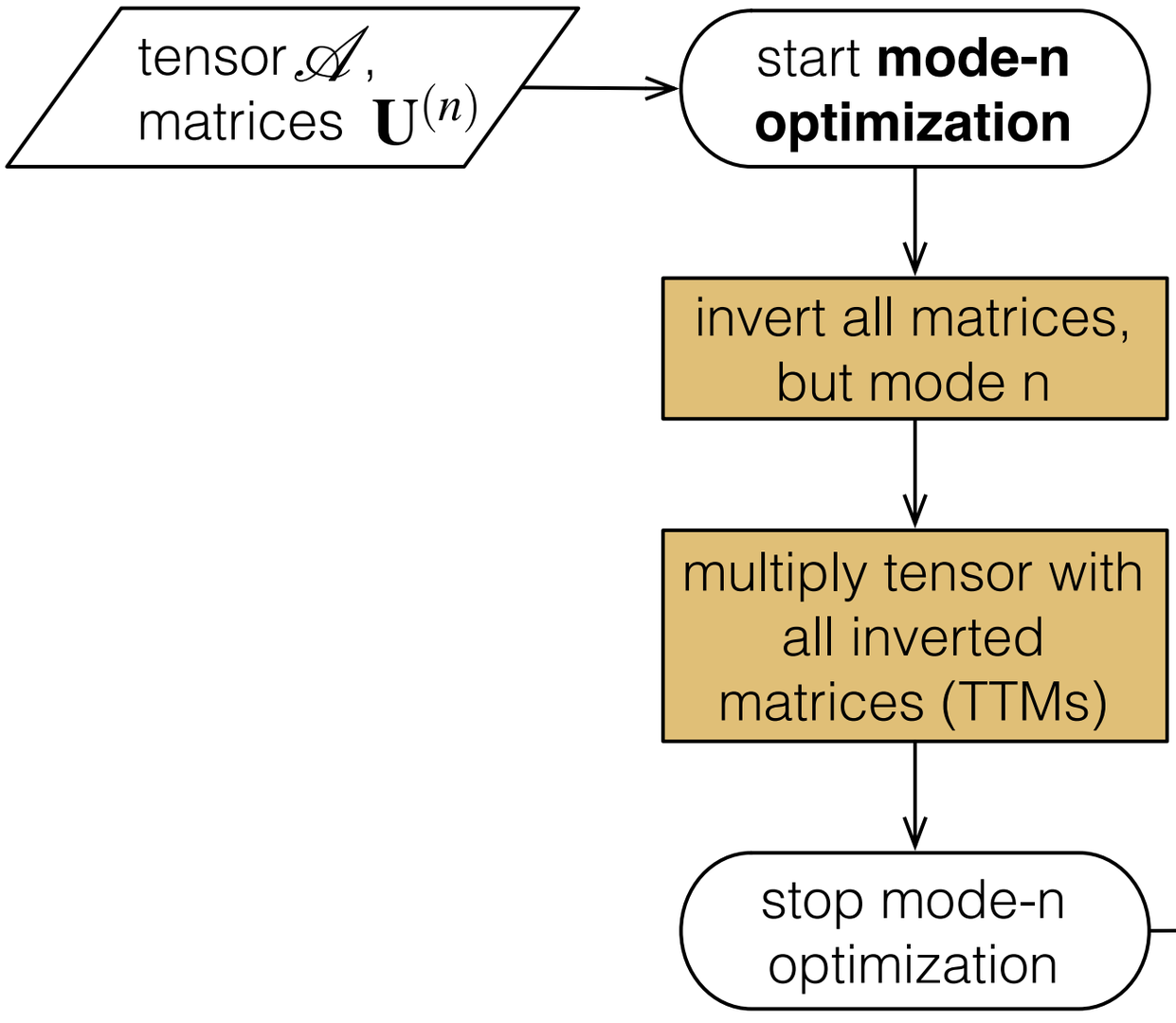


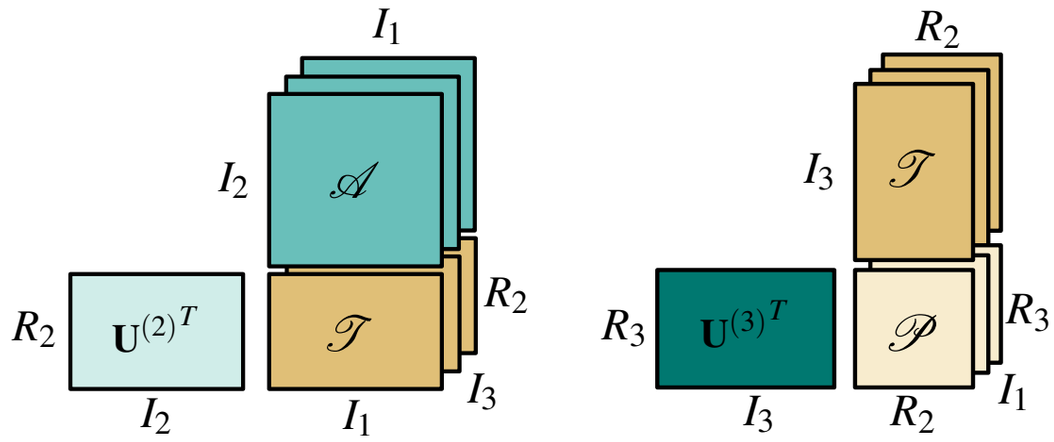
Optimize Factor Matrices

- Higher-order orthogonal iteration
  - Keep factor matrix of mode n fixed
  - Generate optimized data tensor
    - Project original data tensor on the inverted factor matrices of all other modes
  - Receive optimized mode-n factor matrix
    - Apply HOSVD to the optimized tensor

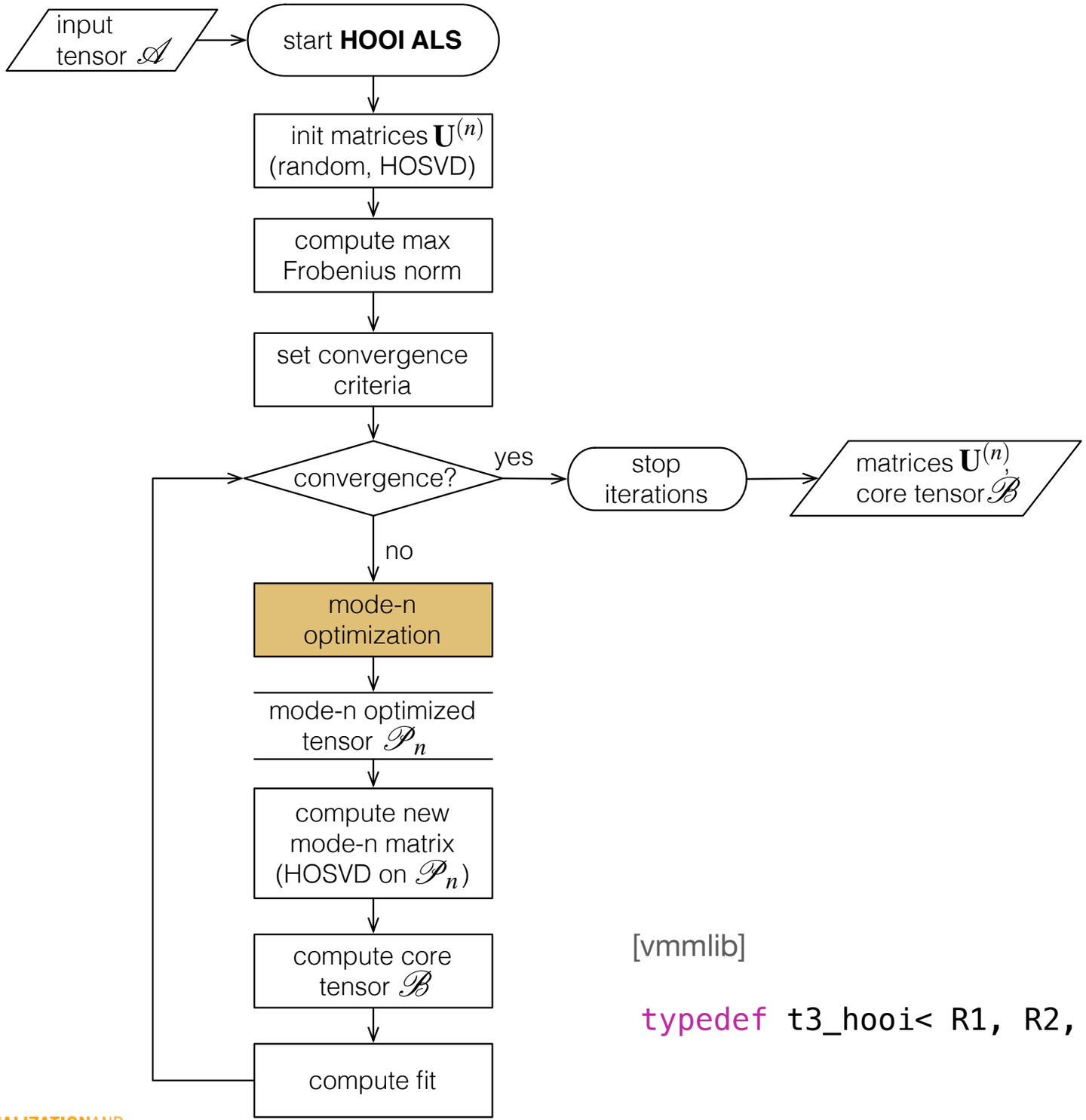
De Lathauwer, de Moor, Vandewalle. On the best rank-1 and rank-(R1,R2,...,RN) approximation of higher-order tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, 2000.

Optimize Mode-n Factor Matrix





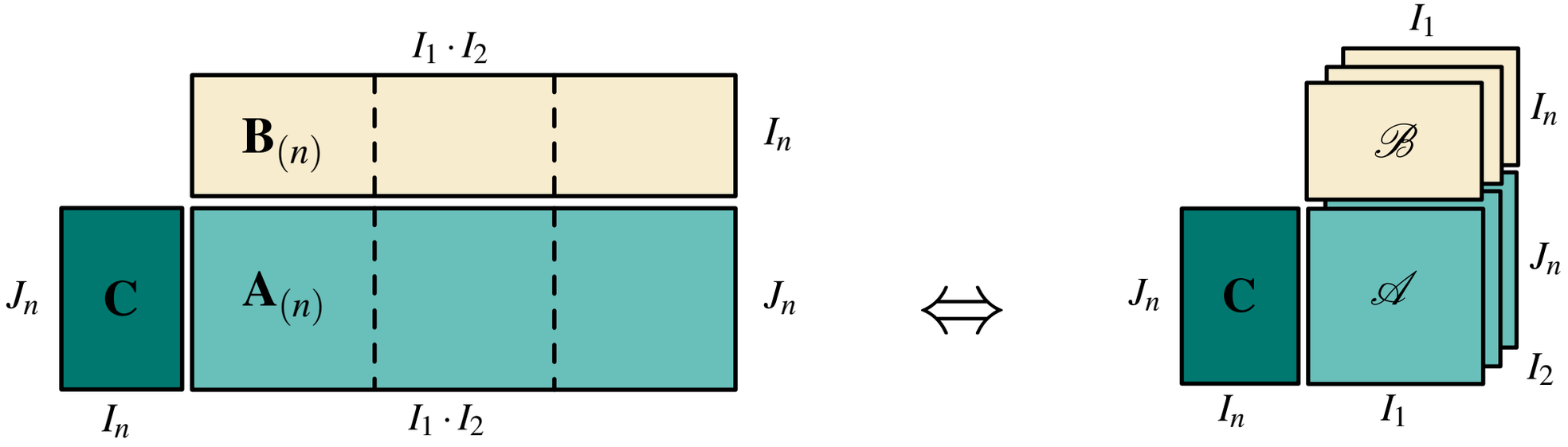
Higher-order Orthogonal Iteration (HOOI)



[vmmlib]

```
typedef t3_hooi< R1, R2, R3, I1, I2, I3 > t3_hooi_t;
```

Tensor Times Matrix Multiplication



$$\mathbf{A}_{(n)} = \mathbf{C}\mathbf{B}_{(n)}$$

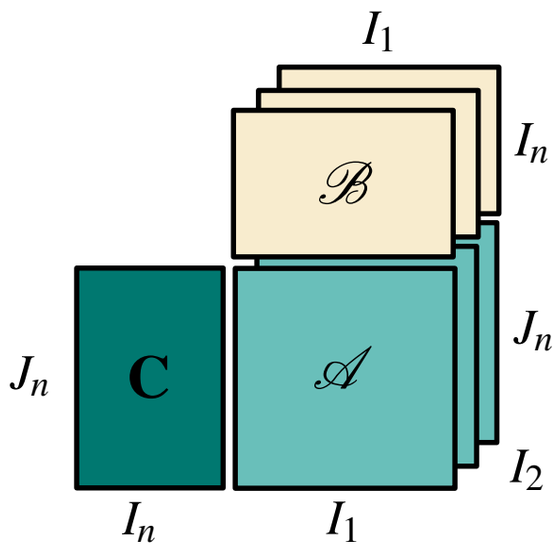
$$\mathscr{A} = \mathscr{B} \times_n \mathbf{C}$$

n-mode product

[De Lathauwer et al., 2000a]

$$(\mathscr{B} \times_n \mathbf{C})_{i_1 \ldots 1_{n-1} j_n i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} b_{i_1 i_2 \ldots i_N} \cdot c_{j_n i_n}$$

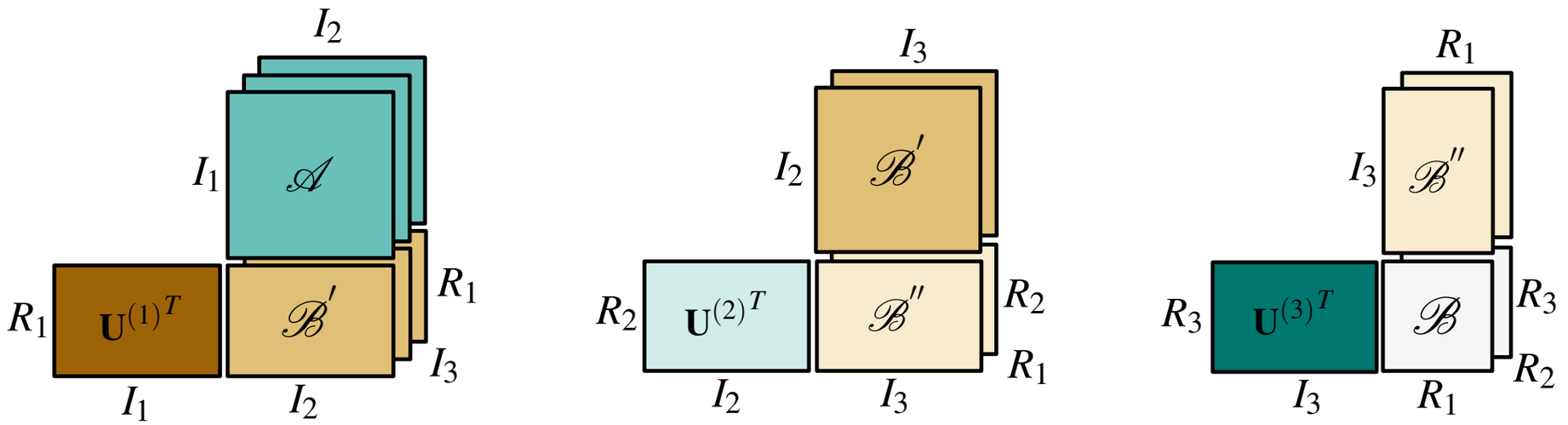
Tensor Times Matrix Multiplications




```
t3_ttm::multiply_frontal_fwd(    tensor3_b, matrix_c1, tensor3_a1 );
t3_ttm::multiply_horizontal_fwd( tensor3_b, matrix_c2, tensor3_a2 );
t3_ttm::multiply_lateral_fwd(    tensor3_b, matrix_c3, tensor3_a3 );

t3_ttm::full_tensor3_matrix_multiplication(
                                    tensor3_b,
                                    matrix_c1,
                                    matrix_c2,
                                    matrix_c3,
                                    tensor3_a
                                    );
```

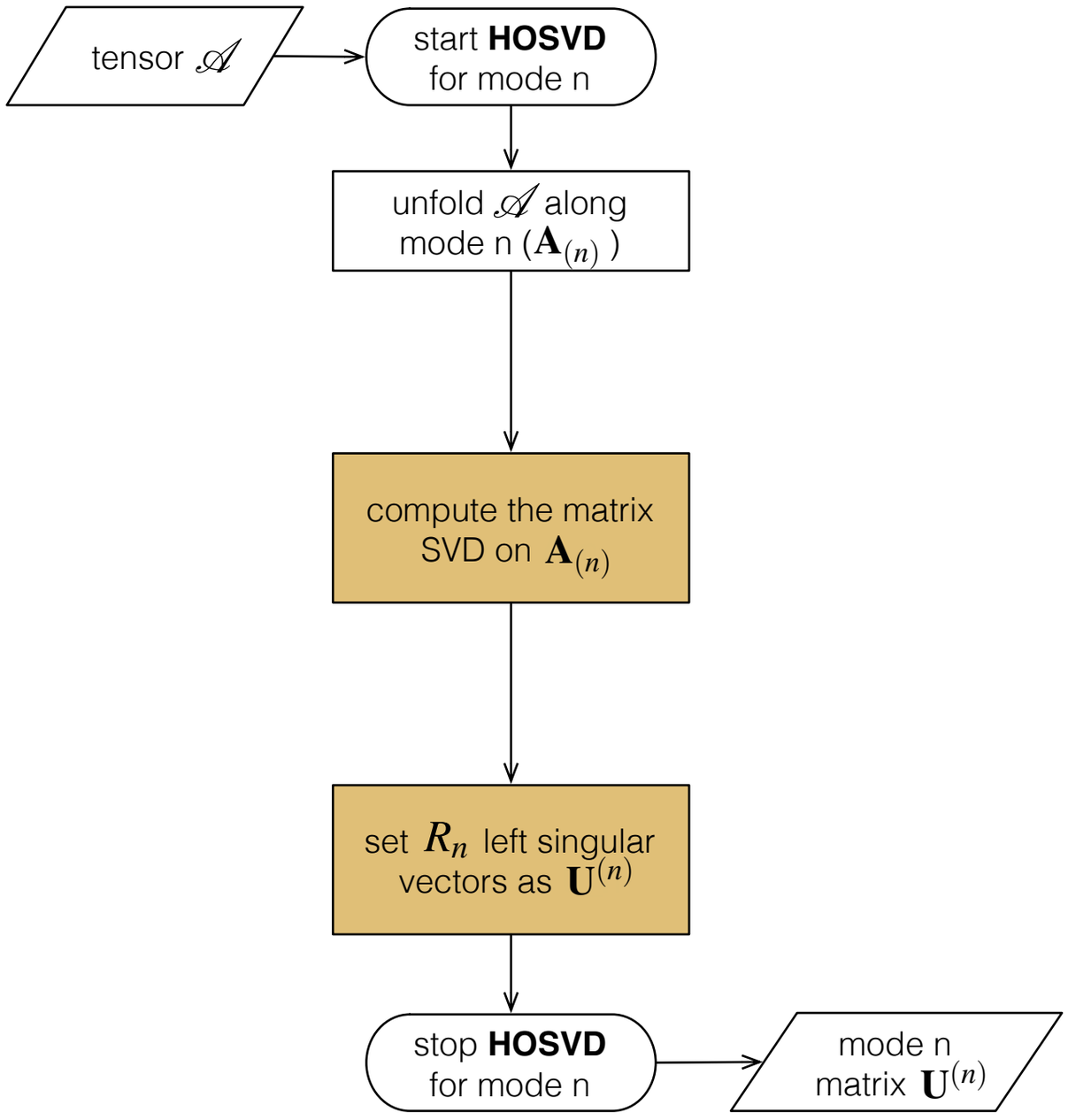
- The T3_TTM is implemented using openMP and BLAS for the parallel matrix-tensor_slice multiplications.
- The full TTM multiplication includes three TTMs: first a TTM along frontal slices, then a TTM along horizontal slices, and finally a TTM along lateral slices.
- Since the tensor3 is an array consisting of frontal slices (matrices), we start first with the frontal slice multiplication. This is optimized for tensors with In > Jn (For example, Tucker core generation). If you have a situation, where Jn > In (for example Tucker reconstruction), you could rearrange the order of the modes of the TTM multiplications such that the most expensive TTM (the one of the largest tensor) is performed along frontal slices.

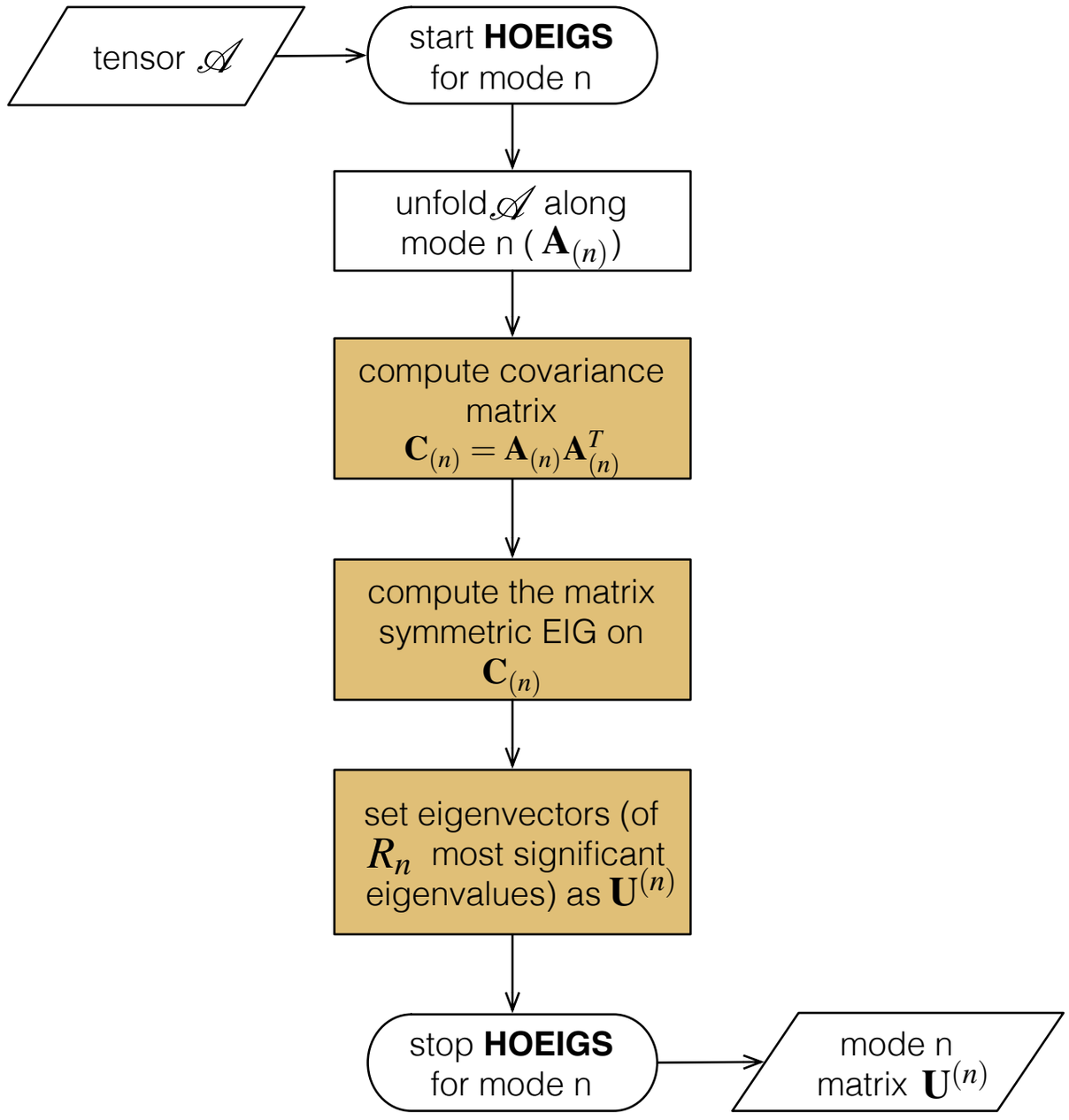
Example TTMs: Core Computation



$$\mathscr{B} = \mathscr{A} \times_1 \mathbf{U}^{(1)(-1)} \times_2 \mathbf{U}^{(2)(-1)} \times_3 \cdots \times_N \mathbf{U}^{(N)(-1)} \xrightarrow[\text{factor matrices}]{\text{orthogonal}} \mathscr{B} = \mathscr{A} \times_1 \mathbf{U}^{(1)T} \times_2 \mathbf{U}^{(2)T} \times_3 \cdots \times_N \mathbf{U}^{(N)T}$$

- Three consecutive TTM multiplications
- For orthogonal matrices, use the transposes of the three factor matrices (otherwise the (pseudo)-inverses)
- `t3_ttm::full_tensor3_matrix_multiplication( A, U1_t, U2_t, U3_t, B );`

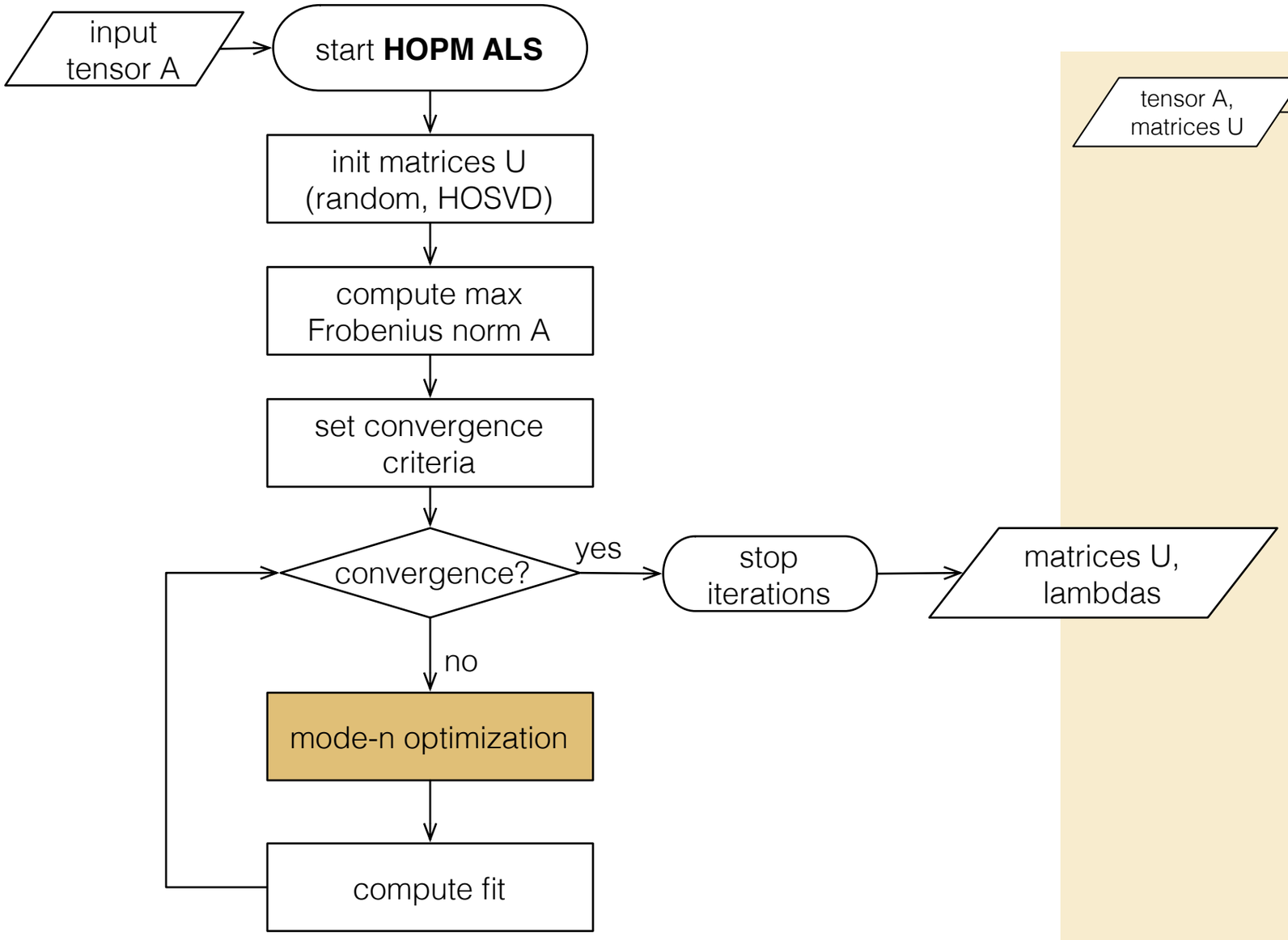


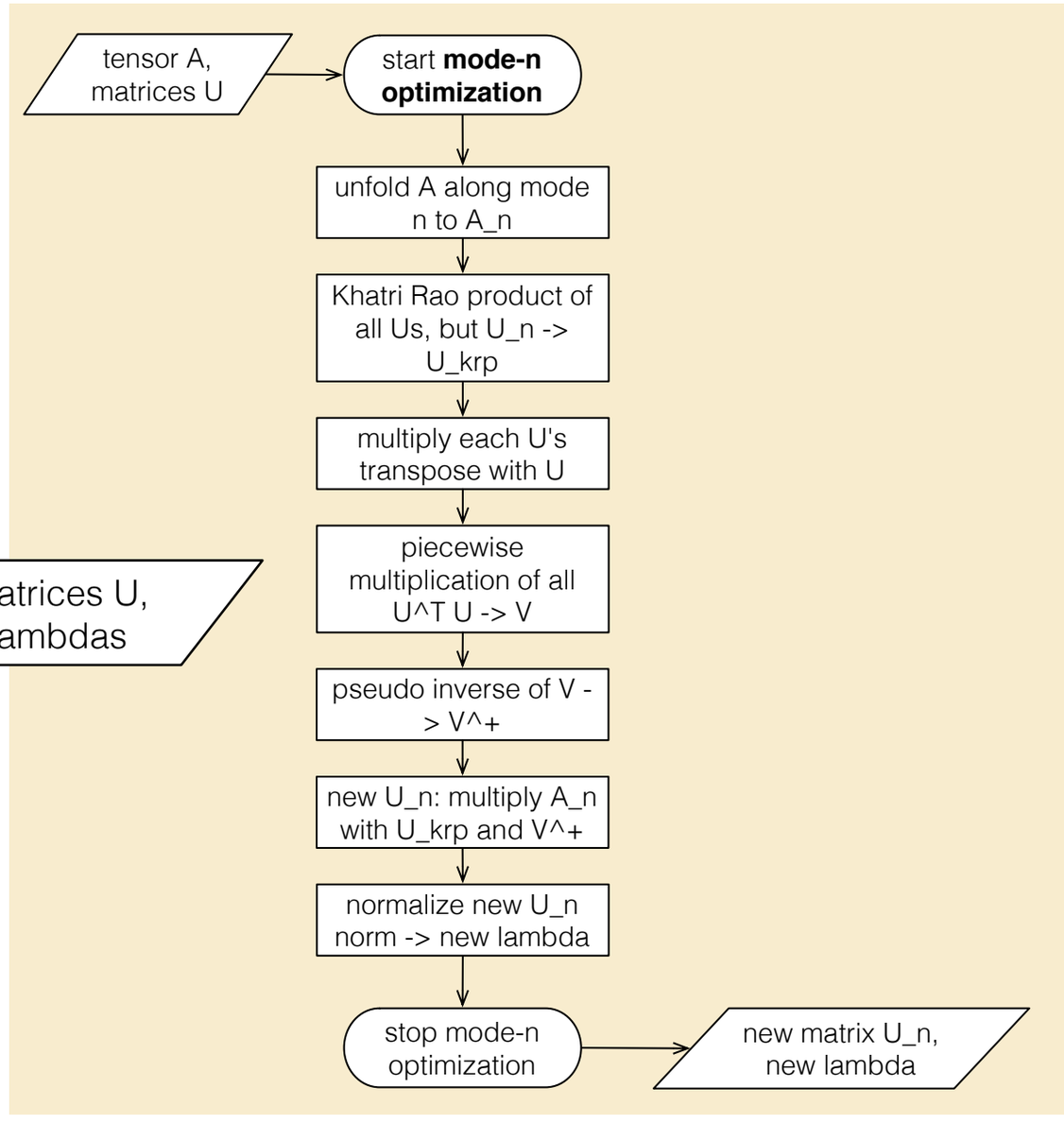


[vmmlib]

```
typedef t3_hosvd< R1, R2, R3, I1, I2, I3 > t3_hosvd_t;
//HOSVD modes: eigs_e or svd_e
```

Higher-order Power Method (HOPM)





[vmmlib]

```
typedef t3_hopm< R, I1, I2, I3 > t3_hopm_t;
```