Journal of Statistical Software

November 2018, Volume 87, Issue 10. doi: 10.18637/jss.v087.i10

rTensor: An R Package for Multidimensional Array (Tensor) Unfolding, Multiplication, and Decomposition

James Li
Facebook

Jacob Bien
Cornell University

Martin T. Wells
Cornell University

Abstract

rTensor is an R package designed to provide a common set of operations and decompositions for multidimensional arrays (tensors). We provide an S4 class that wraps around the base '`array`' class and overloads familiar operations to users of '`array`', and we provide additional functionality for tensor operations that are becoming more relevant in recent literature. We also provide a general unfolding operation, for which the k-mode unfolding and the matrix vectorization are special cases of. Finally, package **rTensor** implements common tensor decompositions such as canonical polyadic decomposition, Tucker decomposition, multilinear principal component analysis, t-singular value decomposition, as well as related matrix-based algorithms such as generalized low rank approximation of matrices and popular value decomposition.

Keywords: tensor, multidimensional arrays, S4, Tucker decomposition, multilinear principal components analysis, generalized low rank approximation of matrices, population valued decomposition, CANDECOMP/PARAFAC, tensor singular value decomposition.

1. Introduction

Advances in medical imaging technology as well as telecommunication data-collection have ushered in massive datasets that make multidimensional data more commonplace. The multilinear structure of such datasets (e.g., individuals × traits × time) gives impetus for statistical techniques that preserve the dimensionality while still tying into the familiar framework of statistical inference and learning.

Flattening of the data (treating one or more of the levels as simply more observations or more variables) and then applying traditional matrix-based methods are often used (Yang,

Zhang, Frangi, and Yang 2004; Zhang and Zhou 2005); however, methods that do not reduce the structural integrity of the data often outperform in both model parsimony and predictive performance (Vasilescu 2009; Sheehan and Saad 2007; Lathauwer, Moor, and Vanderwalle 2000b,a). Hence it is important to extend the statistical framework to datasets that inherently have multi-level structures.

The tensor framework generalizes the familiar notions of vectors and matrices and has been actively used and investigated in chemometrics, image sensing, facial recognition, and psychometrics. We point to Kolda (2006) for a very comprehensive list of references of tensor uses in a variety of fields. Also more recently, there has been a rise of tensor usage in data mining (Acar, Dunlavy, Kolda, and Morup 2011a; Yilmaz, Cemgil, and Simsekli 2011; Acar, Kolda, and Dunlavy 2011b; Morup 2011; Anandkumar, Ge, Hsu, Kakade, and Telgarsky 2014) and computation (Acar *et al.* 2009; Golub and Van Loan 2012; Kang, Papalexakis, Harpale, and Faloutsos 2012). In fact, many of the techniques that have been developed in lieu of a formal tensor setup are later shown to be special cases of models based on the tensor structure (Sheehan and Saad 2007), which further supports the notion that the tensor framework is wide-reaching and important to develop further.

Many papers exist for cataloging and surveying the use of tensor techniques, such as Kolda (2006); Kolda and Bader (2009); Lu, Plataniotis, and Venetsanopoulos (2011); Vasilescu (2009) and Grasedyck, Kressner, and Tobler (2013). Over the past few years, much has changed in the landscape of tensor analysis. This warrants a much more serious consideration of adopting tensor methodology in the statistical community. Our aim in building this R (R Core Team 2018) package is to facilitate tensor manipulation, modeling, and methodological research amongst statisticians. We also believe that with so many different ways to represent a general tensor in matrix form (unfoldings), it is important to have a consistent notation and terminology for such representations.

1.1. Software review

Tensor software is available in multiple platforms. We surveyed and used most of the freely-available ones before deciding to build **rTensor** (Li, Bien, and Wells 2018). In MATLAB (The MathWorks Inc. 2017), Andersson and Bro (2000) created the **N-way Toolbox** to fit the canonical polyadic (CP) decomposition, Tucker decomposition, as well as other multilinear models. This toolbox also handles missing values. Bader and Kolda (2004, 2006) created the **Tensor Toolbox** that provides classes for dense, sparse, and structured tensors. Bader and Kolda (2004, 2006) also provides tensor decompositions such as CP and Tucker. In C++, there are also several libraries designed for tensor operations. **Boost** (Garcia, Siek, and Lumsdaine 2001) and **Blitz++** (Veldhuizen, Cummings, Guio, Stokes, and Shende 2011) both implement multidimensional arrays that promise efficiency and large number of dimensions. However, we found these C++ libraries are mainly lacking both in the decompositions that are supported and in tensor objects where the number of modes that can be dynamically altered after compile-time. While computational efficiency is certainly important, we needed more flexibility in a tensor software that allowed easy tensor analysis and prototyping of models.

In R, the package **tensorA** (Van den Boogaart 2010) provides Einstein and Riemann summing conventions as well as parallel computations for tensors. It does not support any tensor models and decompositions. The package **PTAk** (Leibovici 2010, 2015) does support CP, general Tucker, and two-dimensional principal component analysis (2dPCA), but does not

cover the population value decomposition (PVD), the t-product and the t-singular value decomposition (t-SVD) cases. Package **rTensor** aims to provide a tensor class with general matrix unfolding operations and fundamental tensor operations to support novel development of tensor methods, while package **PTAk** is more specific in its application to spatio-temporal decompositions. Of course, there is also the '`array`' class, which supports multidimensional arrays. Our package aims to extend the functionality of this base '`array`' class by adding k-mode multiplication, t-products, transpose, unfolding, as well as various multidimensional decompositions. Package **rTensor** is available from the Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org/package=rTensor.

Special care is taken to prevent too much slowdown in the code. For instance, consider the `unfold` function, which is central to all of the decompositions and most of the operations featured in our package. There are still significant speed differences between our `unfold` and that of MATLAB's **Tensor Toolbox**, but those are mostly due to speed differences between R's base `aperm` function and those of MATLAB:

```
R> tnsr <- rand_tensor(rep(20, 6))
R> Rprof()
R> mtx <- unfold(tnsr, row_idx = c(4, 1, 3), col_idx = c(2, 5, 6))
R> Rprof(NULL)
R> summaryRprof()

$by.self
                self.time self.pct total.time total.pct
"aperm.default"      0.52      100       0.52       100

$by.total
                total.time total.pct self.time self.pct
"aperm.default"       0.52       100      0.52      100
"aperm"               0.52       100      0.00        0
"unfold"              0.52       100      0.00        0

$sample.interval
[1] 0.02

$sampling.time
[1] 0.52

>> T = randn(20, 20, 20, 20, 20, 20);
>> tic, b = permute(T, [4, 1, 3, 2, 5, 6]); b = reshape(b, 20^3, 20^3); toc,

Elapsed time is 0.182706 seconds.
```

2. rTensor basics

A tensor used in data analysis is a multi-dimensional array (MDA). The modes of a tensor correspond to the dimensions of a MDA. A vector is a 1-tensor, a matrix a 2-tensor, and

Slot name	Type	Description
num_modes	'integer'	The number of modes, or K.
modes	'vector'	The vector of modes/sizes/extents/ dimensions.
data	'vector', 'matrix', or 'array'	The actual data of the tensor.

Table 1: List of slots in the 'Tensor' S4 class.

tensors with 3 or more modes are generally called higher-order tensors. Decompositions of higher-order tensors are often called multi-way analysis or multi-linear models. In this section, we will give an overview of tensor basics as well as how to perform the basic tensor manipulation tasks in package **rTensor**.

2.1. S4 class

Package **rTensor** exports the 'Tensor' S4 class, which extends the base 'array' class that ships with every version of R. The most accurate way to consider the 'Tensor' class is to see it as an API to the default R multidimensional array, allowing the user to easily create, manipulate and model tensors coherent with the set of terminology and algorithms set forth by Kilmer, Braman, Hao, and Hoover (2013); Vasilescu (2009); Kolda (2006); Bro (1997). The 'Tensor' class contains three slots which are given in Table 1.

Let K denote the number of modes for a tensor, and let $n_1 \times n_2 \times \ldots \times n_K$ denote the extents of the modes associated with a K-tensor; n_k specifies the extent of the tensor along mode k. Creation of a 'Tensor' object is done mostly via `as.tensor`, which takes in an 'array', 'matrix', or 'vector' as argument. It is also possible to initialize a 'Tensor' object using the `new("Tensor", num_modes, modes, data)` command.

Addition and subtraction is defined element-wise for tensors of the same modes, while the Frobenius norm extends the matrix case in the usual manner:

$$\|\mathcal{X}\|_F^2 = \sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \ldots \sum_{i_K=1}^{n_K} x_{i_1,\ldots,i_K}^2 .$$

Element-wise operations such as addition, subtraction, element-wise exponentiation, etc. have been overloaded for the 'Tensor' class. For binary element-wise operations, we can provide a 'Tensor' operand along with a 'array' operand as long as the dimensions match up. The same rules apply for 'matrix' objects and 'numeric' vectors.

The Frobenius norm of a K-tensor can be obtained using the method `fnorm`, and we can sum or average across any mode of the tensor to obtain a $(K-1)$-tensor using the methods `modeSum` or `modeMean`. The inner product of two tensors of equal modes can be calculated via the method `innerProd`.

We can subset a 'Tensor' just as we would an 'array' object: simply invoke the subsetting operator `[` and provide the indices of the subset. Any index that is left blank will retrieve the entire range of that mode. To assign the subset of a tensor a value, we would do the expected thing. When a 'Tensor' object is printed to the screen, we only display the first few elements of the data, along with the number of modes and the mode vector. To save space, we omit the data output of printing the tensor. The following code illustrates how to create and operate on a tensor:

```
R> library("rTensor")
R> indices <- c(10, 20, 30, 40)
R> arr <- array(rnorm(prod(indices)), dim = indices)
R> tnsr <- as.tensor(arr)
R> tnsr

Numeric Tensor of 4 Modes
Modes:  10 20 30 40
Data:
[1] -1.1284629  1.0675542  0.8046051 -1.4006100  0.1571699 -1.7975627

R> fnorm(tnsr)

[1] 489.8707

R> modeSum(tnsr, m = 1, drop = FALSE)

Numeric Tensor of 4 Modes
Modes:  1 20 30 40
Data:
[1] -1.8845470 -4.2588321  0.8585407  0.3425072  2.7341223 -2.8549452

R> modeSum(tnsr, m = 1, drop = TRUE)

Numeric Tensor of 3 Modes
Modes:  20 30 40
Data:
[1] -1.8845470 -4.2588321  0.8585407  0.3425072  2.7341223 -2.8549452

R> innerProd(tnsr, tnsr)

[1] 239973.3

R> tnsr[, 1:2, 1, 1]

Numeric Tensor of 2 Modes
Modes:  10 2
Data:
           [,1]       [,2]
[1,] -1.1284629 -1.5121679
[2,]  1.0675542  1.1324152
[3,]  0.8046051 -1.4471833
[4,] -1.4006100  2.3082239
[5,]  0.1571699 -1.7088971
[6,] -1.7975627 -0.5560895
```

2.2. Datasets

We include the AT&T database of faces (Cambridge 1994) in tensor format in this package, which contains images of 40 individuals under 10 different lightings, each image with 92×112 pixels. We structured this dataset as a 4-tensor with modes $92 \times 112 \times 40 \times 10$, where the first two modes correspond to the image pixels, the third mode corresponds to the individual, and the last mode corresponds to the lighting condition. The data object can be accessed using `faces_tnsr`. It is also fairly simple to plot any of the images in this dataset, using the function `plot_orl`. The following snippet of code generates the image corresponding to the 5th individual under the 10th lighting condition:

```
R> plot_orl(subject = 5, condition = 10)
```

2.3. Tensor unfolding

For $K \geq 3$, it is often useful to be able to represent a K-tensor as a matrix or as a vector, especially as a first step in defining a tensor multiplication. This representation is often called unfolding or flattening. To represent a general K-tensor as a matrix, one can choose exactly which modes to map onto the rows and columns. While there are a few conventions that have prevailed in the tensor literature, package **rTensor** provides a general unfolding function that encompasses these conventions as special cases. The folding operations, which invert these unfold operations, are defined through the unfolding themselves. It is important to note that the foldings operate on any arbitrary matrix, so it becomes necessary to specify the exact modes of the resulting tensor.

The general matrix unfolding maps a subset of the modes as indices in the rows and the remaining modes as indices in the columns. As such, it needs to know both which modes are mapped to the rows (`row_idx =`) and which are mapped to the columns (`col_idx =`). The orders of the indices within the rows and columns depend on the order given in these two parameters. Consider the following example. We first use the function `rand_tensor` to generate a 4-tensor consisting of i.i.d. random Normal($\mu = 0, \sigma = 1$) entries, then unfold in two different ways:

```
R> tnsr <- rand_tensor(modes = c(3, 4, 5, 6))
R> unfold(tnsr, row_idx = c(1, 2), col_idx = c(3, 4))

Numeric Tensor of 2 Modes
Modes:  12 30
Data:
           [,1]       [,2]        [,3]       [,4]       [,5]       [,6]
[1,] -0.3598239  0.3199540  1.80046216  0.5264169 -1.1703683  1.6642430
[2,] -0.8469362 -0.2649620 -0.74235168  1.4504419  0.2085634 -0.5773225
[3,] -0.3771786 -0.2427662  0.61483068 -0.6173013 -1.5779438 -0.9869397
[4,]  0.1302490 -0.1866406 -0.04535539  0.5563735 -1.4252314  0.3929646
[5,]  0.9022722 -2.1842911 -0.48736317  0.7888372  1.6272044 -0.3522744
[6,]  1.2760479 -0.2878594  3.96839056  0.9474990  0.1303110 -0.4910413

[...]
```

```
R> unfold(tnsr, row_idx = c(2, 3), col_idx = c(1, 4))

Numeric Tensor of 2 Modes
Modes:  20 18
Data:
             [,1]       [,2]         [,3]       [,4]        [,5]       [,6]
[1,] -0.359823902 -0.8469362 -0.377178615  1.6642430 -0.57732251 -0.9869397
[2,]  0.130249009  0.9022722  1.276047904  0.3929646 -0.35227443 -0.4910413
[3,]  0.001396795  1.5585043  1.289323895  0.4914713 -0.40570634 -0.9339438
[4,] -1.305624607 -0.2760775  0.005373203  0.9010271 -1.37015962  0.8131647
[5,]  0.319953957 -0.2649620 -0.242766237 -0.6987564  0.38338092  0.1212665
[6,] -0.186640559 -2.1842911 -0.287859359 -1.0538028  0.07329372  1.8191891

[...]
```

In the general `fold` method, we would need to specify the full modes of the original '`Tensor`' object as well as `row_idx` and `col_idx`.

```
R> tnsr <- rand_tensor(modes = c(3, 4, 5, 6))
R> unfolded <- unfold(tnsr, row_idx = c(2, 3), col_idx = c(1, 4))
R> folded_back <- fold(unfolded, row_idx = c(2, 3), col_idx = c(1, 4),
+    modes = c(3, 4, 5, 6))
R> identical(folded_back, tnsr)

[1] TRUE
```

Special cases of this general unfolding include the vec$(\cdot)$ operation, which simply stacks the tensor element-wise into a $n_1 n_2 \ldots n_K$ vector. In this case, the row indices is the entire set of modes, while the column indices is the null set. Here we abide by the reverse lexicographical ordering, which is to allow the first index to vary the fastest and the last index to vary the slowest, i.e.,

$$\mathrm{vec}(\mathcal{X}) = \begin{pmatrix} x_{111} \\ x_{211} \\ \vdots \\ x_{121} \\ x_{221} \\ \vdots \\ x_{N_1 N_2 N_3} \end{pmatrix} \in \mathbb{R}^{n_1 n_2 n_3}, \mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}.$$

To invoke the vec$(\cdot)$ operation, the user can call `unfold(tnsr, row_idx = tnsr@modes)`. The default for `col_idx` is `NULL` so it does not need to be specified. We also provide a convenience function `vec`.

Another prevalent unfolding is called the k-mode matricization/unfolding (Kolda 2006). For $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, denote the unfolding in the kth mode as

$$\mathcal{X}_{(k)} \in \mathbb{R}^{n_k \times \prod_{j \neq k} n_j}.$$

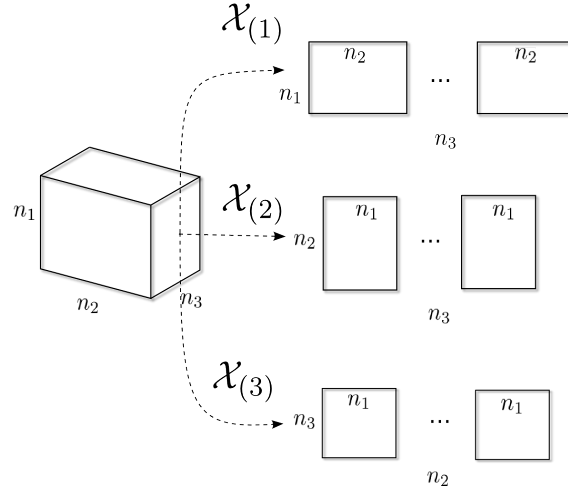


Figure 1: Illustration of the 1-mode, 2-mode, and 3-mode unfoldings for a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

The formal notation for this operation gives a mapping from the $(i_1, i_2, \ldots, i_K)$th element to the (i_k, j)th element of the resulting matrix, where

$$j = 1 + \sum_{p \neq k}^{K} (i_p - 1) J_p, \text{ with } J_p = \prod_{q \neq k}^{p-1} n_q.$$

We stay consistent to the convention in the permutation of the indices $\{n_1, \ldots, n_{k-1}, n_{k+1}, \ldots, n_K\}$. For a 3-tensor, there are three k-mode unfoldings, denoted $\mathcal{X}_{(1)}$, $\mathcal{X}_{(2)}$, and $\mathcal{X}_{(3)}$. Figure 1 shows how the extents of the original tensor map onto the rows and columns of the three unfoldings.

To invoke the k-mode unfolding in the mode k using package **rTensor**, the user can call `unfold` by specifying `row_idx = k` or use the convenience function `k_unfold` with `m = k`.

```
R> tnsr <- rand_tensor(modes = c(2, 3, 4, 5, 6))
R> k_unfold(tnsr, m = 2)

Numeric Tensor of 2 Modes
Modes:  3 240
Data:
           [,1]       [,2]        [,3]       [,4]      [,5]       [,6]
[1,]  0.3578831 -0.8032793 -0.06559354 -0.2990532 -1.232756 -0.6948367
[2,]  1.4607161 -0.7265681  0.65618704  0.1993923 -2.420022 -1.1161419
[3,] -0.3419790 -1.0873609  1.17385498  0.8827398 -1.010616  0.4219558

[...]

R> k_unfold(tnsr, m = 4)

Numeric Tensor of 2 Modes
Modes:  5 144
```

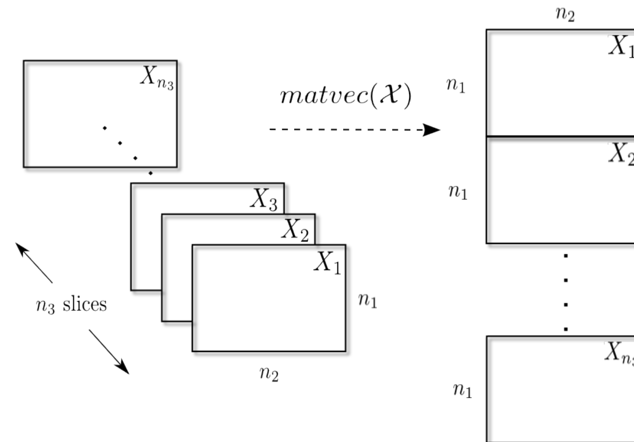


Figure 2: matvec($\mathcal{X}$) for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ which results in a $n_1 n_3 \times n_2$ matrix.

```
Data:
            [,1]        [,2]       [,3]       [,4]       [,5]       [,6]
[1,]  0.35788311 -0.80327927  1.4607161 -0.7265681 -0.3419790 -1.0873609
[2,]  0.09652643 -0.00963707  0.3724675 -1.2845784  0.4300722  0.8459963
[3,] -2.58818372 -0.05481498 -1.8795606  0.7851252 -1.3425023 -0.3089504
[4,]  0.60717572  0.20758000  2.0156336  0.2923068  2.7739223  0.5589189
[5,] -0.22042844 -0.09737430 -0.7436792  0.1000862  0.6692409 -0.6806083

[...]
```

The corresponding `k_fold` method, on the other hand, requires the modes of the original '`Tensor`' object, since all it sees is the matrix as the input.

```
R> tnsr <- rand_tensor(modes = c(2, 3, 4, 5, 6))
R> unfolded <- k_unfold(tnsr, m = 4)
R> k_fold(unfolded, m = 4, modes = c(2, 3, 4, 5, 6))

Numeric Tensor of 5 Modes
Modes:  2 3 4 5 6
Data:
[1] -1.07334966  0.01239117 -0.17909191 -0.67723446 -1.16584575 -1.23881803

R> identical(k_fold(unfolded, m = 4, modes = c(2, 3, 4, 5, 6)), tnsr)

[1] TRUE
```

Kilmer and Martin (2011) proposed another tensor unfolding known as the matvec($\cdot$) operation. For $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, let $X_j := \mathcal{X}[:, :, j]$, $j = 1, \ldots, n_3$, then the matrix vectorization of $\mathcal{X}$, denoted matvec($\mathcal{X}$), is

$$\text{matvec}(\mathcal{X}) = \begin{bmatrix} X_1 \\ \vdots \\ X_{n_3} \end{bmatrix} \in \mathbb{R}^{n_1 n_3 \times n_2}.$$

The matvec$(\cdot)$ is more explicitly illustrated in Figure 2. Observe that matvec$(\mathcal{X}) = \mathcal{X}_{(2)}^\top$. The user could easily perform the matvec$(\cdot)$ operation using the transpose of `k_unfold(..., m = 2)`. Nevertheless, we provide the function `matvec` that avoids the matrix transpose and maps the second mode directly to the row indices. We also provide the inverse folding function `unmatvec`.

2.4. Tensor multiplication

The k-mode product specifies multiplication between a K-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ and a matrix $M \in \mathbb{R}^{J \times n_k}$, where n_k is the k-mode for $\mathcal{X}$ (Kolda 2006). The result is a K-tensor in $\mathbb{R}^{n_1 \times \ldots \times n_{k-1} \times J \times n_{k+1} \times \ldots \times n_K}$. This is defined element-wise to be

$$(\mathcal{X} \times_k M)_{i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_K} = \sum_{i_k=1}^{n_k} \mathcal{X}_{i_1,\ldots,i_K} \cdot M_{j,i_k}.$$

As the name suggests, this product definition is closely related to the k-mode unfolding. In fact

$$\mathcal{Y} = \mathcal{X} \times_k M \quad \Leftrightarrow \quad \mathcal{Y}_{(k)} = M \cdot \mathcal{X}_{(k)},$$

where "$\cdot$" denotes the usual matrix multiplication.

In other words, we can think about the k-mode product as a left matrix multiplication onto the k-mode vectors: each k-mode vector of the resulting tensor $\mathcal{Y}$ is a result of a matrix-vector multiplication between M and the corresponding k-mode vector of $\mathcal{X}$. Note that if M is a vector (i.e., $J = 1$), then each k-mode vector of $\mathcal{Y}$ is the result of an inner product between two vectors, and $\mathcal{Y}$ will have the kth mode being 1 and is essentially a $(K-1)$-tensor.

Also note that if $X \in \mathbb{R}^{n_1 \times n_2}$ were a matrix, then the k-mode product between X and $M_1 \in \mathbb{R}^{J_1 \times n_1}$, $M_2 \in \mathbb{R}^{J_2 \times n_2}$ is equivalent to the following matrix products

$$X \times_1 M_1 = M_1^\top \cdot X \in \mathbb{R}^{J_1 \times n_2}$$
$$X \times_2 M_2 = X \cdot M_2 \in \mathbb{R}^{n_1 \times J_2}.$$

The function to perform k-mode multiplication is `ttm` – short for "tensor times matrix", the name of a function with similar usage in the MATLAB **Tensor Toolbox** (Bader and Kolda 2004). `ttm` takes in a '`Tensor`' object, a '`matrix`' object, and the mode for multiplication. It then proceeds to unfold the '`Tensor`' object in the mode specified, to perform matrix multiplication with the '`matrix`' object on the left, and to fold the resulting matrix back into a '`Tensor`'. Naturally, the number of columns of the '`matrix`' object must match the mode specified for the original '`Tensor`' object.

```
R> tnsr <- rand_tensor(modes = c(4, 6, 8, 10))
R> mat <- matrix(rnorm(12), ncol = 6)
R> ttm(tnsr = tnsr, mat = mat, m = 2)

Numeric Tensor of 4 Modes
Modes:  4 2 8 10
Data:
[1] -1.4457715 -0.1249305  2.3950996 -0.7543450 -0.1727204  0.7295488
```

The k-mode product serves as the basis for many tensor decompositions and regression models, including the Tucker decomposition and the CP decomposition. It also bears mentioning that the Kronecker product permits a matrix view of the product between a general K-tensor and a list of matrices (Kolda 2006)

$$\mathcal{Y} = \mathcal{X} \times_1 M_1 \times_2 M_2 \ldots \times_K M_K$$
$$\Leftrightarrow \mathcal{Y}_{(k)} = M_k \cdot \mathcal{X}_{(k)} \cdot (M_K \otimes \ldots \otimes M_{k+1} \otimes M_{k-1} \ldots \otimes M_1)^\top.$$

For many tensor decompositions, there is frequently a need to perform a series of k-mode multiplications using multiple factor matrices. To this end, `ttl` is a function that takes in a single tensor $\mathcal{X}$, a '`list`' of '`matrix`' objects $\{M_1, M_2, \ldots, M_n\}$, and a vector of modes $(i_1, i_2, \ldots, i_n)$, and then returns the output $\mathcal{X} \times_{i_1} M_1 \times_{i_1} M_1 \times_{i_3} \ldots \times_{i_n} M_n$. The number of columns of each matrix must match the corresponding modes of $\mathcal{X}$.

```
R> mat2 <- matrix(rnorm(24), ncol = 8)
R> ttl(tnsr = tnsr, list_mat = list(mat, mat2), ms = c(2, 3))

Numeric Tensor of 4 Modes
Modes:  4 2 3 10
Data:
[1] -8.385004 -7.194387  5.908805  1.689403 -2.678040 -3.381004
```

To illustrate the k-mode multiplication further, let us consider a more visual example using a single subject from `faces_tnsr`. We can swap the positions of the images in the subject 3-tensor ($92 \times 112 \times 10$) by multiplying it with a 10×10 matrix with 1 in the coordinates of the swap, and 0 everywhere else. Consider the following example:

```
R> subject <- faces_tnsr[, , 1, ]
R> new_positions <- c(7, 10, 1, 6, 3, 2, 5, 8, 9, 4)
R> mat <- matrix(0, 10, 10)
R> for (i in 1:10L) {
+    mat[new_positions[i], i] <- 1
+ }
R> subject_new <- ttm(subject, mat, m = 3)
R> all(identical(subject_new[, , new_positions], subject[, , seq(1, 10)]))

TRUE
```

For more properties of the k-mode product, see Kolda (2006). While the k-mode product defines multiplication between a tensor and a matrix, it does not provide a natural way to multiply two 3-tensors. To this end, the t-product has recently been proposed by Kilmer and Martin (2011). First we must illustrate the block circulant matrix generated from the matvec$(\cdot)$ of a tensor. Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then

$$\text{circ}(\text{matvec}(\mathcal{X})) = \begin{bmatrix} X_1 & X_{n_3} & X_{n_3-1} & \ldots & X_2 \\ X_2 & X_1 & X_{n_3} & \ldots & X_3 \\ X_3 & X_2 & X_1 & \ldots & X_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_{n_3} & X_{n_3-1} & X_{n_3-2} & \ldots & X_1 \end{bmatrix},$$

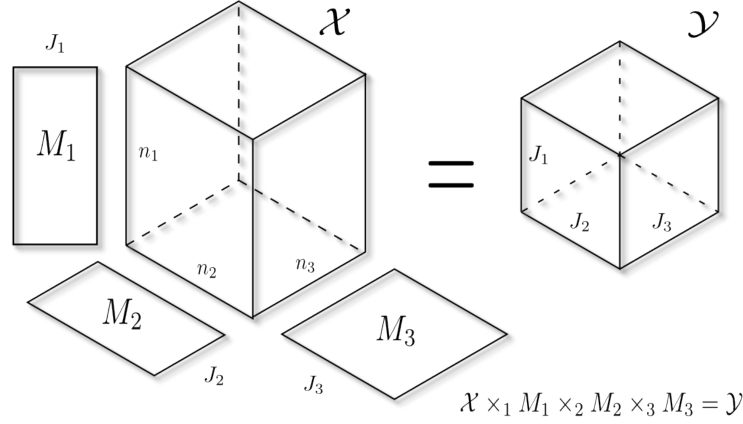


Figure 3: k-mode product of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and matrices $M_k \in \mathbb{R}^{J_k \times n_k}$. The result is $\mathcal{Y} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$.

where $X_j = \mathcal{X}[:,:,j]$ is defined to be the jth slice of $\mathcal{X}$ along mode 3.

The t-product is defined via the block circulant structure and the matvec$(\cdot)$ operator and allows for a direct multiplication of two 3-tensors. For $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and $\mathcal{B} \in \mathbb{R}^{n_2 \times L \times n_3}$, the t-product is $\mathcal{A} * \mathcal{B} \in \mathbb{R}^{n_1 \times L \times n_3}$, where the matvec$(\mathcal{A} * \mathcal{B})$ is a result of matrix multiplication

$$\begin{aligned}\text{matvec}(\mathcal{A} * \mathcal{B}) &= \text{circ}(\text{matvec}(\mathcal{A})) \cdot \text{matvec}(\mathcal{B}) \\ &= \begin{bmatrix} A_1 & A_{n_3} & A_{n_3-1} & \dots & A_2 \\ A_2 & A_1 & A_{n_3} & \dots & A_3 \\ A_3 & A_2 & A_1 & \dots & A_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n_3} & A_{n_3-1} & A_{n_3-2} & \dots & A_1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_{n_3} \end{bmatrix} \in \mathbb{R}^{n_1 n_3 \times L}.\end{aligned}$$

Here $A_j = \mathcal{A}[:,:,j], B_j = \mathcal{B}[:,:,j]$ are the jth slices along mode 3 of $\mathcal{A}$ and $\mathcal{B}$ respectively. To get the tensor product $\mathcal{A} * \mathcal{B}$, we simply have to fold matvec$(\mathcal{A} * \mathcal{B})$ using the inverse folding for matvec$(\cdot)$, denoted unmatvec$(\cdot)$. Hence $\mathcal{A} * \mathcal{B} = \text{unmatvec}(\text{circ}(\text{matvec}(\mathcal{A})) \cdot \text{matvec}(\mathcal{B}))$.

From the definition of the t-product, we can see that each mode-3 slice of the resulting tensor $\mathcal{A} * \mathcal{B}$ is given by a sum of products of the mode-3 slices of $\mathcal{A}$ and $\mathcal{B}$. In fact, the t-product $\mathcal{A} * \mathcal{B}$ defines a linear map that takes $\mathcal{B} \in \mathbb{R}^{n_2 \times L \times n_3}$ to $\mathbb{R}^{n_1 \times L \times n_3}$ (Martin, Shafer, and LaRue 2013). It also allows the extension of familiar linear algebra concepts such as the transpose, orthogonality, nullspace, and range. For detailed accounts of these properties, refer to Kilmer and Martin (2011). When $n_3 = 1$, we get back the usual matrix multiplication.

The t-product is implemented via the function `t_mult`. It takes in two '`Tensor`' objects, $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}, \mathcal{Y} \in \mathbb{R}^{n_2 \times L \times n_3}$, and returns $\mathcal{X} * \mathcal{Y} \in \mathbb{R}^{n_1 \times L \times n_3}$ (Algorithm 1). Note that we take the fast Fourier transform based algorithm approach as found in Kilmer *et al.* (2013) in our implementation. This implementation avoids creating the prohibitively large block circulant matrix that is involved in the definition of the t-product.

```
R> tnsr1 <- rand_tensor(modes = c(3, 4, 5))
R> tnsr2 <- rand_tensor(modes = c(4, 6, 5))
R> t_mult(tnsr1, tnsr2)
```

```
Numeric Tensor of 3 Modes
Modes:  3 6 5
Data:
[1] -0.08515525  1.65217442  0.86002207  3.33730964  1.87913580 -1.59224659
```

3. rTensor decompositions

In this section, we describe tensor decompositions that are implemented in package **rTensor**. These decomposition models represent the bulk of the tensor methodology used in facial recognition, data mining, and statistical analysis of images. Throughout this section, we illustrate the various decomposition functions in package **rTensor** using the AT&T face dataset (Cambridge 1994), which is included in the package as `faces_tnsr`.

3.1. CP, HOSVD, and Tucker

The CP decomposition stems independently from psychometrics (Carroll and Chang 1970) and chemometrics (Bro 1997), where the same method was separately named canonical decomposition (CANDECOMP) and parallel factors (PARAFAC).

A K-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ is called rank-1 if it can be expressed as an outer product of K vectors. More precisely, the rank of a K-tensor is defined as the minimal value of r for which the tensor can be expressed as as sum of r rank-1 tensors:

$$\mathcal{X} = \sum_{\ell=1}^{r} v_{1\ell} \circ v_{2\ell} \ldots \circ v_{K\ell}, \text{ where } v_{k\ell} \in \mathbb{R}^{n_k}, 1 \leq \ell \leq r, 1 \leq k \leq K.$$

For matrices, the well-known Eckhart Young theorem provides the existence and form of an optimal lower-rank approximation. However, this type of result has been shown not to generalize to K-tensors for $K \geq 3$ (Kolda 2003).

The CP decomposition provides an approximation of $\mathcal{X}$ using a rank-r tensor $\hat{\mathcal{X}}$, where r is given *a priori*. The goal is then to construct a rank-r tensor that minimizes the Frobenius norm of the difference between $\mathcal{X}$ and $\hat{\mathcal{X}}$

$$\min_{\text{all } v_{k\ell}} \|\mathcal{X} - \hat{\mathcal{X}}\|_F,$$

where

$$\begin{aligned}
\hat{\mathcal{X}} &= \sum_{\ell=1}^{r} v_{1\ell} \circ v_{2\ell} \ldots \circ v_{K\ell} && (1)\\
&= \sum_{\ell=1}^{r} \lambda_\ell \cdot u_{1\ell} \circ u_{2\ell} \ldots \circ u_{K\ell},\ u_{k\ell} = \frac{v_{k\ell}}{\|v_{k\ell}\|} && (2)\\
&= \Lambda \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K, && (3)
\end{aligned}$$

with $U_k = \begin{bmatrix} u_{k1} & u_{k2} & \ldots & u_{kr} \end{bmatrix} \in \mathbb{R}^{n_k \times r}$ and $\Lambda \in \mathbb{R}^{r \times r \times r}$ a 3-tensor that contains the λ's on the super-diagonal and 0 elsewhere, as seen in Figure 4.

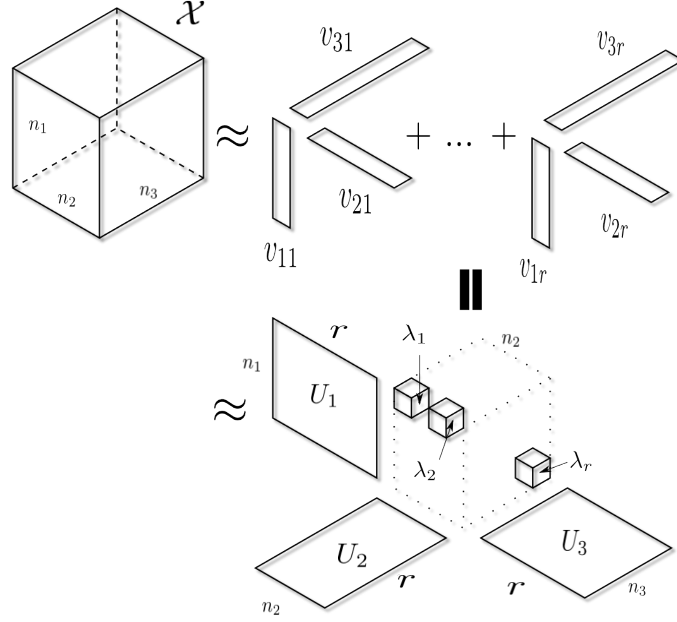


Figure 4: CP decomposition for a $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$. The first part shows a representation using a sum of rank-1 tensors. The second part shows a representation using factor matrices.

The equivalence of lines (1) and (2) above are due to the fact that each $u_{k\ell}$ vector is $v_{k\ell}$ normalized by its norm with the norm information stored in the λ_r's. Furthermore, we can store the $u_{k\ell}$ vectors as a factor matrix U_k for each $k = 1, \ldots, K$ (Kroonenberg 2012), leading to the form in line (3). Note that here the U_k matrices are not orthogonal. This relationship is illustrated for a 3-tensor below in Figure 4.

The `cp` function implements the classical alternating least squares method to compute the CP decomposition of a general K-tensor.[1] Note that this algorithm is not guaranteed to converge to the global minimum (Kolda 2006). The function returns `lambdas` and `U_list`. `lambdas` is a vector containing the elements in the super-diagonal core tensor λ, while `U_list` is the list of factor matrices $U_1, \ldots, U_K$. We demonstrate the CP decomposition on one of the subjects (#14) in the AT&T face database.

```
R> subject14 <- faces_tnsr[, , 14, ]
R> cp1 <- cp(subject14, num_components = 50)
R> cp2 <- cp(subject14, num_components = 10)
R> cp1$norm_percent

[1] 87.4869

R> cp2$norm_percent

[1] 81.16162
```

[1] Acceleration techniques for CP decompositions have recently been proposed in Phan, Tichavsky, and Cichocki (2013).

Figure 5: CP decomposition with 50 components and 10 components on subject 14 in the AT&T face dataset. Picture 1 of 10 shown. *Left:* Original. *Middle:* 50 components. *Right:* 10 components.

The Tucker decomposition (Tucker 1966; Kroonenberg 2012; Lathauwer *et al.* 2000b) is still based on the idea of obtaining the best approximation of $\mathcal{X}$, but relaxes the constraint that $\hat{\mathcal{X}}$ must be expressed as a sum of r rank-1 tensors. Instead, the Tucker decomposition constructs $\hat{\mathcal{X}}$ to approximate $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ using a reduced core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$ and K factor matrices, each of rank $r_k \leq n_k$, $k = 1, \ldots, K$,

$$\hat{\mathcal{X}} = \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K.$$

If this looks similar to the CP, it is because the CP decomposition is a specialized version of the Tucker decomposition with all the ranks equal, (i.e., $r = r_1 = \ldots = r_K$) (Kolda 2006). One way to compute the Tucker decomposition is via the higher order orthogonal iteration (HOOI) (Lathauwer *et al.* 2000b), which constrains the factor matrices to be orthogonal.

Before we demonstrate HOOI, we first discuss the very much related higher-order singular value decomposition (HOSVD) provided by the seminal paper by Lathauwer *et al.* (2000a). HOSVD decomposes a K-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ as follows

$$\mathcal{X} = \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K,$$

where each square matrix $U_k \in \mathbb{R}^{n_k \times n_k}$ is orthogonal and the core tensor $\mathcal{G} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$ has the special property that for any $k, 1 \leq k \leq K$, it is

- all-orthogonal: $\langle \mathcal{G}_{i_k=\alpha}, \mathcal{G}_{i_k=\beta} \rangle = 0$ for any $\alpha \neq \beta$, and
- ordered: $\|\mathcal{G}_{i_k=1}\|_F \geq \|\mathcal{G}_{i_k=2}\|_F \geq \ldots \|\mathcal{G}_{i_k=n_k}\|_F$.

All-orthogonality of $\mathcal{G}$ means that for any of the K modes, any subtensor of size $K-1$ with different indices (i.e., α and β) along the same mode has an inner-product of 0. For example, if $K = 3$, then any two matrix slices with different indices along that mode has an inner-product of 0. It is important to note that the resulting $\mathcal{G}$ is a diagonal tensor under the CP decomposition, but not necessarily so under Tucker.

Algorithm 1: Illustration of the higher-order singular value decomposition (HOSVD).

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$.
for $k = 1, \ldots, K$ **do**
  $U_k \leftarrow$ left orthogonal matrix of the SVD of $\mathcal{X}_{(k)}$
end
$\mathcal{G} \leftarrow \mathcal{X} \times_1 U_1^\top \times_2 U_2^\top \times_3 \ldots \times_K U_K^\top$
output : core tensor $\mathcal{G} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, orthogonal factor matrices $U_1, \ldots, U_K$, each $U_k \in \mathbb{R}^{n_k \times n_k}$

The corresponding algorithm to compute the HOSVD illustrates its crucial connection between the k-mode unfolding: for each k-mode unfolding, perform a matrix SVD for $\mathcal{X}_{(k)}$ so that $\mathcal{X}_{(k)} = U_k \Sigma_k V_k^\top$. Now we can simply define $\mathcal{G} := \mathcal{X} \times_1 U_1^\top \times_2 U_2^\top \ldots \times_K U_K^\top$, then

$$\begin{aligned} \mathcal{G}_{(k)} &= U_k^\top \cdot \mathcal{X}_{(k)} \cdot (U_K^\top \otimes \ldots \otimes U_{k+1}^\top \otimes U_{k-1}^\top \otimes \ldots \otimes U_1^\top)^\top \\ \Rightarrow \mathcal{X}_{(k)} &= U_k \cdot \mathcal{G}_{(k)} \cdot (U_K \otimes \ldots \otimes U_{k+1} \otimes U_{k-1} \otimes \ldots \otimes U_1)^\top \\ \Rightarrow \mathcal{X} &= \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 \ldots \times_K U_K, \end{aligned}$$

giving us the HOSVD.[2] We demonstrate below how to do this in package **rTensor**.

```
R> tnsr <- rand_tensor(modes = c(2, 4, 6, 8))
R> hosvd1 <- hosvd(tnsr)
R> hosvd1$U[[1]] %*% t(hosvd1$U[[1]])
```

```
             [,1]         [,2]
[1,] 1.000000e+00 1.110223e-16
[2,] 1.110223e-16 1.000000e+00
```

Note that we can also allow the orthogonal factor matrices to be truncated in a HOSVD (i.e., truncate each U_k to its first r_k columns for each $1 \leq k \leq K$), thereby compressing $\mathcal{X}$ and forming an approximation. This procedure is called the truncated HOSVD (Lathauwer *et al.* 2000b), and it is done simply by specifying the reduced ranks in the `hosvd` function. The truncated HOSVD forms the conceptual basis for HOOI.

```
R> hosvd2 <- hosvd(tnsr, ranks = c(1, 2, 3, 4))
R> 1 - hosvd2$fnorm_resid / fnorm(tnsr)
```

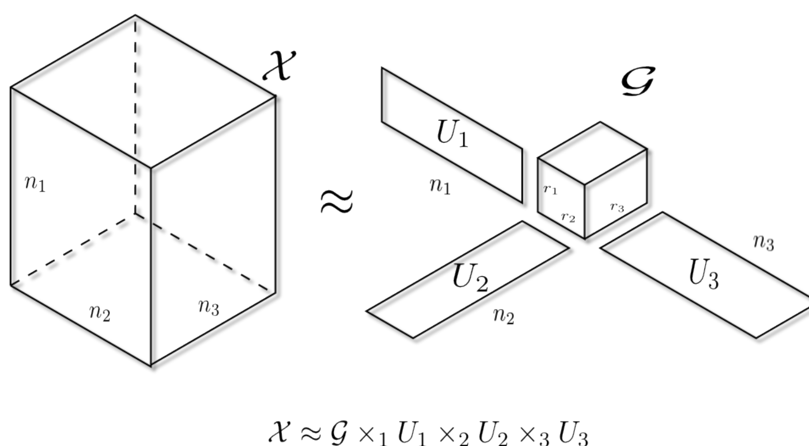
```
[1] 0.0944159
```

What makes HOOI different from HOSVD is that HOOI essentially involves *multiple iterations* of this alternating truncation and SVD for all the modes to give us a locally optimized approximation $\hat{\mathcal{X}}$. The full algorithm for HOOI is given in Algorithm 2 and illustrated in Figure 6.

[2] A more efficient strategy for performing (truncated-)HOSVD, which leads to almost the same quasi-optimal accuracy, has been described in Vannieuwenhoven, Vandebril, and Meerbergen (2012).

Algorithm 2: Illustrations of the orthogonal Tucker alternating least squares (HOOI).

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_K}$, and desired ranks $r_1, \ldots, r_K$.
initialize: $U_1, \ldots, U_K$ via HOSVD
while *Not converged* **do**
  for $k = 1, \ldots, K$ **do**
    $\mathcal{Y} \leftarrow \mathcal{X} \times_1 U_1^\top \times_2 \ldots \times_{k-1} U_{k-1}^\top \times_{k+1} U_{k+1}^\top \ldots \times_K U_K^\top$
    $U_k \leftarrow r_k$ leading singular vectors of $\mathcal{Y}_{(k)}$
  end
end
$\mathcal{G} \leftarrow \mathcal{X} \times_1 U_1^\top \times_2 \ldots \times_K U_K^\top$
output : core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times \ldots \times r_K}$, factor matrices with orthogonal columns $U_1, \ldots, U_K$, each $U_k \in \mathbb{R}^{n_k \times r_k}$



$$\mathcal{X} \approx \mathcal{G} \times_1 U_1 \times_2 U_2 \times_3 U_3$$

Figure 6: The Tucker decomposition for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ results in factor matrices with orthogonal columns $U_k \in \mathbb{R}^{n_k \times r_k}, k = 1, 2, 3$ and an all-orthogonal core tensor $\mathcal{G} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$.

For HOOI and the remaining iterative algorithms we describe in this paper, we use the same convergence criterion. At each iteration of the algorithm, we check for convergence by first forming a current estimate of the original tensor based on the estimated U_k's, then measure the difference between the estimation error of the current iteration and the estimation error of the previous iteration. If this difference is smaller than the difference tolerance set by the user, then we stop iterating. By default there is also a maximum number of iterations set in place, which can also be adjusted by the user.

We apply the more general HOOI on the entire face dataset, compressing on all 4 modes (including the mode running across the individuals), as demonstrated in Figure 7.

```
R> tucker1 <- tucker(faces_tnsr, ranks = c(46, 56, 35, 8))
R> tucker1$norm_percent

[1] 88.78375

R> tucker2 <- tucker(faces_tnsr, ranks = c(23, 28, 10, 3))
```

Figure 7: Illustration of the HOOI with various ranks on the entire AT&T face dataset. Subject 11, picture 6 shown. *Left:* Original. *Middle:* $r_1 = 46, r_2 = 56, r_3 = 35, r_4 = 8$. *Right:* $r_1 = 23, r_2 = 28, r_3 = 10, r_4 = 3$.

```
R> tucker2$norm_percent

[1] 77.86446
```

3.2. GLRAM, MPCA, and PVD

In this section, we discuss several statistical models that are unified under the Tucker framework. While some of these models explicitly use tensor notation and methodology, others use a series of matrices as inputs.

The generalized low rank approximation of matrices (GLRAM; Ye 2005) belongs in the latter category. For a series of matrices of the same size, $M_1, \ldots, M_{n_3} \in \mathbb{R}^{n_1 \times n_2}$, GLRAM constructs orthogonal matrices $L \in \mathbb{R}^{n_1 \times r_1}, R \in \mathbb{R}^{n_2 \times r_2}$ and a series of core matrices $G_j \in \mathbb{R}^{r_1 \times r_2}$ to minimize the quantity $\sum_{j=1}^{n_3} \|M_j - L \cdot G_j \cdot R^\top\|_F^2$. The parameters r_1 and r_2 would also need to be given *a priori*.

The series of images GLRAM takes as input can be restructured into a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, where $\mathcal{X}[:,:,j] = M_j$. This has been done in Sheehan and Saad (2007), where it is shown that when structured in this way, GLRAM becomes a special case of the HOOI. During GLRAM, we are essentially performing compression over only the first two modes, leaving the third mode uncompressed. We present the GLRAM algorithm using tensor notation in Algorithm 3 and demonstrate the function `glram` on subject 21 in the AT&T face dataset in Figure 8.

```
R> subject21 <- faces_tnsr[, , 21, ]
R> glram1 <- tucker(subject21, ranks = c(46, 56, 10))
R> glram1$norm_percent

[1] 95.65463
```

Algorithm 3: GLRAM using tensor notation.

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, ranks r_1, r_2.
initialize: U_1, U_2 via HOSVD, $U_3 = I_{n_3}$
while *Not converged* **do**
  $\mathcal{Y} \leftarrow \mathcal{X} \times_1 U_2^\top$
  $U_1 \leftarrow r_1$ leading singular vectors of $\mathcal{Y}_{(1)}$
  $\mathcal{Y} \leftarrow \mathcal{X} \times_1 U_1^\top$
  $U_2 \leftarrow r_1$ leading singular vectors of $\mathcal{Y}_{(2)}$
end
$\mathcal{G} \leftarrow \mathcal{X} \times_1 U_1^\top \times_2 U_2^\top$
output : $L = U_1$; $R = U_2$; $(G_1, \ldots, G_{n_3}) = \text{matvec}(\mathcal{G})$

Figure 8: GLRAM with various ranks on subject 21 in the AT&T face dataset. Picture 2 shown. *Left:* Original. *Middle:* $r_1 = 46, r_2 = 56$. *Right:* $r_1 = 23, r_2 = 28$.

```
R> glram2 <- tucker(subject21, ranks = c(23, 28, 10))
R> glram2$norm_percent

[1] 91.88975
```

Multilinear principal component analysis (MPCA; Lu, Plataniotis, and Venetsanopoulos 2008) is a special case of the general Tucker decomposition for K-tensors, compressing on $K-1$ modes and leaving one mode uncompressed. Hence GLRAM can be viewed as a special case of MPCA as GLRAM is designed specifically for 3-tensors. Notationally, MPCA is equivalent to HOOI with $U_K = I_{n_k}$.

We turn next to the entire face database of 40 individuals and run MPCA on the 4-tensor, compressing on the first three modes (e.g., image row, image column, and pictures), using the same r_1 and r_2 as in GLRAM. We can also examine the Frobenius norm recovered using MPCA, and it seems as though having more individuals (and hence having a 4-tensor) did not help in recovery. This is also noticeable from the estimated images.

```
R> mpca1 <- tucker(faces_tnsr, ranks = c(46, 56, 20, 10))
```

Figure 9: MPCA with various ranks on the entire AT&T face dataset. Subject 35, picture 8 shown. *Left:* Original. *Middle:* $r_1 = 46, r_2 = 56, r_3 = 20$. *Right:* $r_1 = 46, r_2 = 56, r_3 = 10$.

```
R> mpca1$norm_percent
```

```
[1] 84.54957
```

```
R> mpca2 <- tucker(faces_tnsr, ranks = c(46, 56, 10, 10))
R> mpca2$norm_percent
```

```
[1] 79.65943
```

PVD was recently proposed by Crainiceanu, Caffo, Luo, Zipunnikov, and Punjabi (2013) and provides a framework to construct population-level factor matrices for a series of images. We show in this section that PVD can be viewed as a variant of GLRAM. This point was first made by Lock, Nobel, and Marron (2011) in the rejoinder of the original PVD paper Crainiceanu *et al.* (2013), and Crainiceanu *et al.* replied that PVD differs from Tucker (or specifically, GLRAM) in many ways. Most notably, the matrices P and D do not have to be orthogonal and the default PVD has a closed form solution. We first present PVD and the default algorithm suggested by the authors to construct the population matrices P and D, then examine the differences between PVD and GLRAM. Finally, we discuss how PVD might be cast into the tensor framework.

Like GLRAM, PVD is a model designed for a series of matrices instead of a 3-tensor. Given a sample of images $M_1, \ldots, M_{n_3} \in \mathbb{R}^{n_1 \times n_2}$, and $2n_3+2$ parameters, PVD constructs population level matrices $P \in \mathbb{R}^{n_1 \times r_1}$ and $D \in \mathbb{R}^{n_2 \times r_2}$ such that $X_j = P \cdot V_j \cdot D + E_j$, where the $V_j \in \mathbb{R}^{r_1 \times r_2}$, $j = 1, \ldots, n_3$, are called the core matrices. In addition to the 2 parameters $r_i \leq n_i$, $i = 1, 2$, we also would need to choose $2n_3$ compression parameters, $l_1, \ldots, l_{n_3}, h_1, \ldots, h_{n_3}$, that will determine how much left and right truncation will occur for each of the n_3 matrices.

The PVD procedure starts with a separate SVD of each image, $M_j = U_j \Sigma_j W_j^\top$, truncating (possibly differently for each image) the left and right eigenvectors to form $\tilde{U}_j$ and $\tilde{W}_j$. Then one stacks the $\tilde{U}_j$'s column wise to form a big matrix U and does the same for $\tilde{W}_j$'s to form

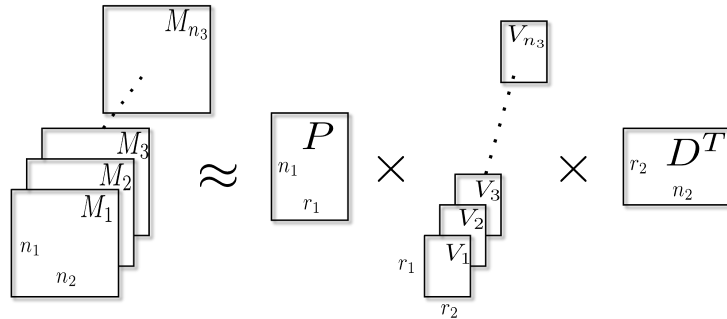


Figure 10: PVD of a series of images $M_1, \ldots, M_{n_3}$. Each M_j is approximated by $P \cdot V_j \cdot D^\top$.

W. The final step is to conduct an eigenvalue decomposition of $U \cdot U^\top$ and $W \cdot W^\top$ to form the population level matrices P and D. In the end, each M_j has a projection

$$\hat{M}_j = P \cdot \underbrace{\{(P^\top \cdot \tilde{U}_j) \cdot \Sigma_j^{(l_j,h_j)} \cdot (\tilde{W}_j \cdot D^\top)\}}_{V_j} \cdot D,$$

where $\Sigma_j^{(l_j,h_j)}$ is the $l_j \times h_j$ left upper block of Σ_j.

Unlike the usual algorithm needed to solve GLRAM, the algorithm to solve the default PVD is not iterative, although the computational cost of the model does scale up with the number of images, since each image requires a separate SVD. Furthermore, with a large n_3, the $UU^\top$ and $WW^\top$ matrices may be intractable for a full eigenvalue decomposition. We present the matrix version of the default PVD algorithm below.

We decided that the input into a PVD model should be a 3-tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, with n_3 being the number of images in the series, and each $n_1 \times n_2$ $\mathcal{X}[:,:,j]$ being an image, $1 \leq j \leq n_3$.

As with the ranks of HOOI and CP, optimizing the parameters required by the PVD model is currently mostly ad-hoc. Recall that for $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, there are $2n_3 + 2$ parameters. At each of the n_3 SVDs of individual images, ℓ_j and h_j are the truncation indices for the left and right eigenvectors, respectively. These are the `uranks` and `wranks`. At the end, we also have r_1 and r_2, which are the truncation indices for the two final eigenvalue decompositions of the two large covariance matrices. These are the parameters `a` and `b` required by the `pvd` function. Empirically, we have found that having $\ell_j > r_1$ or having $h_j > r_2$ resulted in poor fits of the data. To illustrate `pvd`, we return to the AT&T face dataset, choosing subject # 8 subject this time.

```
R> subject8 <- faces_tnsr[, , 8, ]
R> pvd1 <- pvd(subject8, uranks = rep(46, 10), wranks = rep(56, 10),
+     a = 46, b = 56)
R> pvd2 <- pvd(subject8, uranks = rep(46, 10), wranks = rep(56, 10),
+     a = 23, b = 28)
R> pvd1$norm_percent

[1] 96.67298
```

Algorithm 4: Default PVD.

input : Matrices $M_1, \ldots, M_{n_3}$, matrix-wise ranks $l_1, \ldots, l_{n_3}, h_1, \ldots, h_{n_3}$, final ranks r_1, r_2.

for $j = 1, \ldots, n_3$ **do**

  Perform SVD to obtain $M_j = U_j \cdot \Sigma_j \cdot W_j^\top$

  $\tilde{U}_j \leftarrow$ left l_j columns of U_j

  $\tilde{W}_j \leftarrow$ left h_j columns of W_j

end

Stack the $\tilde{U}_j$ column-wise to construct $U = \begin{bmatrix} \tilde{U}_1 & \ldots & \tilde{U}_{n_3} \end{bmatrix}$

and similarly stack the $\tilde{W}_j$ to form $W = \begin{bmatrix} \tilde{W}_1 & \ldots & \tilde{W}_{n_3} \end{bmatrix}$.

$P \leftarrow$ eigenvectors corresponding to the r_1 leading eigenvalues of $U \cdot U^\top$

$D \leftarrow$ eigenvectors corresponding to the r_2 leading eigenvalues of $W \cdot W^\top$

for $j = 1, \ldots, n_3$ **do**

  $V_j \leftarrow P^\top \cdot \tilde{U}_j \cdot \Sigma_j^{(l_j, r_j)} \cdot \tilde{W}_j^\top D^\top$

end

output: $P, D, V_1, \ldots, V_{n_3}$

Figure 11: PVD model with various ranks on subject 8 in the AT&T face dataset. Picture 4 shown. *Left:* Original. *Middle:* $l_1 = \ldots = l_{n_3} = r_1 = 46, h_1 = \ldots = h_{n_3} = r_2 = 56$. *Right:* $l_1 = \ldots = l_{n_3} = 46, h_1 = \ldots = h_{n_3} = 56, r_1 = 23, r_2 = 28$.

```
R> pvd2$norm_percent
```

```
[1] 92.8208
```

3.3. t-SVD

The decomposition t-SVD is based on the t-product (recall its definition from Section 2.4) (Kilmer *et al.* 2013). Before we discuss the t-SVD, we first introduce the notion of the tensor

transpose based on the t-product. Let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then

$$\mathcal{X}^\top := \text{unmatvec}(\begin{bmatrix} X_1^\top \\ X_{n_3}^\top \\ \vdots \\ X_2^\top \end{bmatrix}), \text{ where } X_j = \mathcal{X}[:,:,j].$$

It is easily verified that $(\mathcal{X}^\top)^\top = \mathcal{X}$. Furthermore, let the identity tensor $\mathcal{I} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ be defined with $\mathcal{I}[:,:,1] = I_{n_1}$, the matrix identity of size n_1, and the rest of $\mathcal{I}$ is set to 0. These two definitions then facilitate the notion of tensor orthogonality via the t-product

$$\mathcal{Q} \in \mathbb{R}^{n_1 \times n_1 \times n_3} \text{ is orthogonal if and only if } \mathcal{Q} * \mathcal{Q}^\top = \mathcal{Q}^\top * \mathcal{Q} = \mathcal{I} \in \mathbb{R}^{n_1 \times n_1 \times n_3}.$$

We have overloaded the transpose function `t` for the '`Tensor`' class. Note, however, that this is only currently defined for 3-tensors and not for general K-tensors.

```
R> tnsr <- rand_tensor(c(4, 5, 6))
R> tnsr

Numeric Tensor of 3 Modes
Modes:  4 5 6
Data:
[1]  0.1721087 -0.5029503 -1.4179078 -0.3187059  0.2417989 -0.6050234

R> t_tnsr <- t(tnsr)
R> t_tnsr

Numeric Tensor of 3 Modes
Modes:  5 4 6
Data:
[1]  0.1721087  0.2417989  1.2417775 -2.1195366 -0.8991089 -0.5029503

R> identical(t(t_tnsr), tnsr)

[1] TRUE
```

As shown in Kilmer *et al.* (2013), an orthogonal tensor preserves the Frobenius norm under the t-product. In other words, if $\mathcal{Q} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ is orthogonal and $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then $\|\mathcal{Q} * \mathcal{X}\|_F = \|\mathcal{X}\|_F$. We can now describe the t-SVD: let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, then $\mathcal{X}$ admits a decomposition

$$\mathcal{X} = \mathcal{U} * \mathcal{S} * \mathcal{V}^\top,$$

where $\mathcal{U}, \mathcal{V}$ are orthogonal tensors of sizes $n_1 \times n_1 \times n_3$ and $n_2 \times n_2 \times n_3$ respectively, and $\mathcal{S}$ is of size $n_1 \times n_2 \times n_3$ and consists of diagonal matrices along the third mode. When $n_3 = 1$, then t-SVD reduces to the matrix SVD of $X \in \mathbb{R}^{n_1 \times n_2}$ (Kilmer *et al.* 2013). This is a consequence of the fact that the t-product reduces to matrix multiplication when $n_3 = 1$.

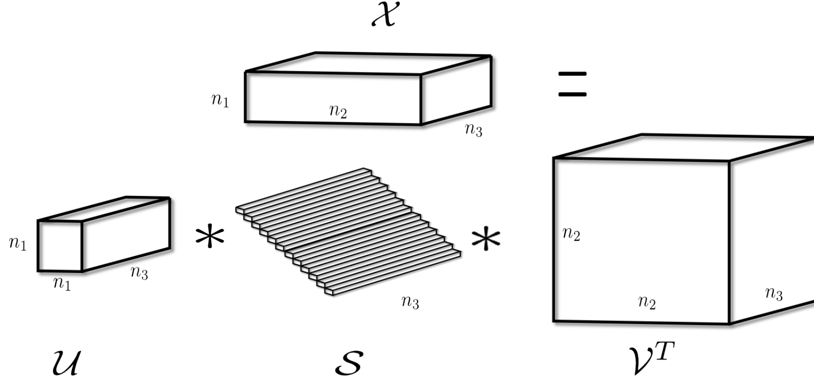


Figure 12: The t-SVD for a $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ results in two orthogonal tensors – $\mathcal{U} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$ and $\mathcal{V} \in \mathbb{R}^{n_2 \times n_2 \times n_3}$ – and $\mathcal{S} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ has diagonal faces along n_3.

Algorithm 5: An illustration of the t-singular value decomposition (t-SVD).

---

input : $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, n_2$ **do**
    $\mathcal{D}[i_1, i_2, :] = \text{fft}(\mathcal{X}[i_1, i_2, :])$
  end
end
for $j = 1, \ldots, n_3$ **do**
  Compute the SVD of the complex $\mathcal{D}[:, :, j]$ to yield $\mathcal{D}[:, :, j] = U_j \cdot \Sigma_j \cdot V_j^\top$
  $\mathcal{U}[:, :, j] \leftarrow U_j$
  $\mathcal{V}[:, :, j] \leftarrow V_j$
  $\mathcal{S}[:, :, j] \leftarrow \Sigma_j$
end
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, n_1$ **do**
    $\mathcal{U}[i_1, i_2, :] = \text{ifft}(\mathcal{U}[i_1, i_2, :])$
  end
end
for $i_1 = 1, \ldots, n_2$ **do**
  for $i_2 = 1, \ldots, n_2$ **do**
    $\mathcal{V}[i_1, i_2, :] = \text{ifft}(\mathcal{V}[i_1, i_2, :])$
  end
end
for $i_1 = 1, \ldots, n_1$ **do**
  for $i_2 = 1, \ldots, n_2$ **do**
    $\mathcal{S}[i_1, i_2, :] = \text{ifft}(\mathcal{S}[i_1, i_2, :])$
  end
end
output: Orthogonal tensors $\mathcal{U} \in \mathbb{R}^{n_1 \times n_1 \times n_3}$, $\mathcal{V} \in \mathbb{R}^{n_2 \times n_2 \times n_3}$ and $\mathcal{S} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ with diagonal slices along the third mode

---

The $\mathcal{S}$ tensor contains the eigentubes $\mathcal{S}[i,i,:], 1 \leq i \leq \tilde{n} := \min(n_1, n_2)$, each of which is a vector of length n_3. Similar to the matrix eigenvalue counterparts, these eigentubes are ordered by the Frobenius norm

$$\|\mathcal{S}[1,1,:]\|_F \geq \|\mathcal{S}[2,2,:]\|_F \geq \ldots \geq \|\mathcal{S}[\tilde{n},\tilde{n},:]\|_F.$$

Computations involving the t-product is not carried out via the block circulant matrices but rather via the fast Fourier transform based algorithms given by Kilmer and Martin (2011). FFT facilitates the computation since the matrix formed by unfolding the first tensor is block circulant, and the transform gives rise to a block diagonal structure in Fourier space, which allows for significant reduction in computation of the product. While complex values are involved in the computation, if $\mathcal{X}$ consists of real values, then $\mathcal{U}, \mathcal{V}$ and $\mathcal{S}$ are all real as well. We demonstrate how to compute the t-SVD using package **rTensor** below. Unfortunately, by using the `fft` function in R, there is some round-off error that occurs when we transform a series using FFT, then transform it back using the inverse FFT. Our calculations in the t-SVD inherit these round-off errors, as we observe in the example below.

```
R> tnsr <- rand_tensor(c(3, 4, 5))
R> decomp <- t_svd(tnsr)
R> decomp

$U
Numeric Tensor of 3 Modes
Modes:  3 3 5
Data:
[1] -0.10611584  0.02219984  0.08776625 -0.02380716  0.22520605  0.05009474

$V
Numeric Tensor of 3 Modes
Modes:  4 4 5
Data:
[1] -0.041270397 -0.057252769  0.536529617  0.009139057  0.052476990
[6] -0.006963590

$S
Numeric Tensor of 3 Modes
Modes:  3 4 5
Data:
[1] 5.911667 0.000000 0.000000 0.000000 3.251907 0.000000

R> decomp$S@data

, , 1

         [,1]     [,2]     [,3] [,4]
[1,] 5.911667 0.000000 0.000000    0
[2,] 0.000000 3.251907 0.000000    0
```

```
[3,] 0.000000 0.000000 1.772416    0

, , 2

          [,1]     [,2]      [,3] [,4]
[1,] 0.4903723 0.000000 0.0000000    0
[2,] 0.0000000 0.833781 0.0000000    0
[3,] 0.0000000 0.000000 0.6733481    0

, , 3

          [,1]      [,2]      [,3] [,4]
[1,] 0.6469147 0.0000000 0.0000000    0
[2,] 0.0000000 0.3426009 0.0000000    0
[3,] 0.0000000 0.0000000 0.4197653    0

, , 4

          [,1]      [,2]      [,3] [,4]
[1,] 0.6469147 0.0000000 0.0000000    0
[2,] 0.0000000 0.3426009 0.0000000    0
[3,] 0.0000000 0.0000000 0.4197653    0

, , 5

          [,1]     [,2]      [,3] [,4]
[1,] 0.4903723 0.000000 0.0000000    0
[2,] 0.0000000 0.833781 0.0000000    0
[3,] 0.0000000 0.000000 0.6733481    0
```

```
R> recovered <- t_mult(t_mult(decomp$U, decomp$S), t(decomp$V))
R> mean(recovered@data - tnsr@data)
```

```
[1] -1.60115e-16
```

4. Summary

We provide an R package that implements prevalent tensor unfolding/refolding, multiplication, and decompositions. This article is meant to guide users of our package as well as connect several themes in the tensor literature. Tensor methodology is still under active development, and the landscape of tensor computation can change very rapidly due to surging interests from the field of statistics and machine learning.

The following table summarizes all of the decompositions and the inputs. These decompositions represent the bulk of the package, and we try to be consistent in the outputs of each decomposition. The output to every function is a standard list containing objects that are relevant to that decomposition.

Function	Tensor size	Other parameters
`cp`	$n_1 \times n_2 \times \ldots \times n_K$	number of components r, maximum number of iterations, convergence criterion
`mpca`	$n_1 \times n_2 \times \ldots \times n_K$	vector of ranks $\mathbf{r} = (r_1, \ldots, r_{K-1})$, maximum number of iterations, convergence criterion
`tucker`	$n_1 \times n_2 \times \ldots \times n_K$	vector of ranks $\mathbf{r} = (r_1, \ldots, r_K)$, maximum number of iterations, convergence criterion
`pvd`	$n_1 \times n_2 \times n_3$	vector of left ranks $\ell = (\ell_1, \ldots, \ell_{n_3})$, vector of right ranks $\mathbf{h} = (h_1, \ldots, h_{n_3})$, final left rank r_1, final right rank r_2
`hosvd`	$n_1 \times n_2 \times \ldots \times n_K$	*optional:* vector of ranks $\mathbf{r} = (r_1, \ldots, r_{K-1})$
`t_svd`	$n_1 \times n_2 \times n_3$	none

Table 2: List of tensor decompositions in package **rTensor**.

For decompositions that allow for compression – `cp`, `mpca`, `tucker`, `pvd`, and `t_compress`, the output for each function will all be a list containing:

- `est` – the compressed estimate of the original '`Tensor`' object.
- `fnorm_resid` – the Frobenius norm of the difference between `est` and the original '`Tensor`' object.
- `norm_percent` – the percent of the Frobenius norm "recovered" by the compressed estimated. This is calculated as 1 − `fnorm_resid/fnorm(tnsr)`.
- `conv` – whether or not the algorithm converged by the maximum iteration (only for the iterative algorithms such as `cp`, `mpca`, and `tucker`).
- `all_resids` – a vector of residuals at each iteration (only for the iterative algorithms).

References

Acar E, Dunlavy D, Kolda T, Morup M (2011a). "Scalable Tensor Factorizations for Incomplete Data." *Chemometrics and Intelligent Laboratory Systems*, **106**(1), 41–56. doi:10.1016/j.chemolab.2010.08.004.

Acar E, Kolda T, Dunlavy D (2011b). "All-at-Once Optimization for Coupled Matrix and Tensor Factorizations." arXiv:1105.3422 [math.NA], URL https://arxiv.org/abs/1105.3422.

Acar E, *et al.* (2009). "Workshop Report: Future Directions in Tensor-Based Computation and Modeling." URL http://www.cs.cornell.edu/cv/TenWork/FinalReport.pdf.

Anandkumar A, Ge R, Hsu D, Kakade SM, Telgarsky M (2014). "Tensor Decompositions for Learning Latent Variable Models." *Journal of Machine Learning Research*, **15**, 2773–2832. URL http://jmlr.org/papers/v15/anandkumar14b.html.

Andersson CA, Bro R (2000). "The **N-Way Toolbox** for MATLAB." *Chemometrics and Intelligent Laboratory Systems*, **52**(1), 1–4. doi:10.1016/s0169-7439(00)00071-x.

Bader B, Kolda T (2004). "MATLAB Tensor Classes for Fast Algorithm Prototyping." *Technical report*, Sandia National Laboratories.

Bader BW, Kolda TG (2006). "Algorithm 862: MATLAB Tensor Classes for Fast Algorithm Prototyping." *ACM Transactions on Mathematical Software*, **32**(4), 635–653. doi:10.1145/1186785.1186794.

Bro R (1997). "PARAFAC. Tutorial and Applications." *Chemometrics and Intelligent Laboratory Systems*, **38**(2), 149–171. doi:10.1016/s0169-7439(97)00032-4.

Cambridge AL (1994). *AT&T "The Database of Faces" (Formerly "The ORL Database of Faces"*. URL http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

Carroll JD, Chang JJ (1970). "Analysis of Individual Differences in Multidimensional Scaling via an N-Way Generalization of Eckhart-Young Decomposition." *Psychometrika*, **35**(3), 283–319. doi:10.1007/bf02310791.

Crainiceanu C, Caffo B, Luo S, Zipunnikov V, Punjabi N (2013). "Population Value Decomposition: A Framework for the Analysis of Image Populations." *Journal of the American Statistical Association*, **106**(495), 775–790. doi:10.1198/jasa.2011.ap10089.

Garcia R, Siek J, Lumsdaine A (2001). *Boost.MultiArray: The* ***Boost*** *Multidimensional Array Library*. URL https://www.boost.org/doc/libs/1_55_0/libs/multi_array/doc/.

Golub GH, Van Loan C (2012). *Matrix Computations*. 4th edition. John Hopkins University Press.

Grasedyck L, Kressner D, Tobler C (2013). "A Literature Survey of Low-Rank Tensor Approximation Techniques." *GAMM-Mitteilungen*, pp. 53–78.

Kang U, Papalexakis E, Harpale A, Faloutsos C (2012). "GigaTensor: Scaling Tensor Analysis Up by 100 Times – Algorithms and Discoveries." In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pp. 316–324. ACM, New York. doi:10.1145/2339530.2339583.

Kilmer M, Braman K, Hao N, Hoover R (2013). "Third-Order Tensors as Operators on Matrices: A Theoretical and Computational Framework with Applications in Imaging." *SIAM Journal on Matrix Analysis and Applications*, **34**(1), 148–172. doi:10.1137/110837711.

Kilmer M, Martin C (2011). "Facial Recognition Using Tensor-Tensor Decomposition." *SIAM Linear Algebra and Its Applications*, **435**(3), 641–658. doi:10.1016/j.laa.2010.09.020.

Kolda T (2006). "Multilinear Operators for Higher-Order Decompositions." *Technical report*, Sandia National Laboratories.

Kolda TG (2003). "A Counterexample to the Possibility of an Extension of the Eckart-Young Low-Rank Approximation Theorem for the Orthogonal Rank Tensor Decomposition." *SIAM Journal of Matrix Analysis and Applications*, **24**(3), 762–767. doi:10.1137/s0895479801394465.

Kolda TG, Bader BW (2009). "Tensor Decomposition and Applications." *SIAM Review*, **51**(3), 455–500. doi:10.1137/07070111x.

Kroonenberg PM (2012). *Applied Multiway Data Analysis*. John Wiley & Sons.

Lathauwer L, Moor BD, Vanderwalle J (2000a). "A Multilinear Singular Value Decomposition." *Journal of Matrix Analysis and Applications*, **21**(4), 1253–1278. doi:10.1137/s0895479896305696.

Lathauwer L, Moor BD, Vanderwalle J (2000b). "On the Best Rank-1 and Rank(R_1, R_2, ..., R_N) Approximation of Higher-Order Tensors." *Journal of Matrix Analysis and Applications*, **21**(4), 1324–1342. doi:10.1137/s0895479898346995.

Leibovici D (2015). ***PTAk**: Principal Tensor Analysis on k Modes*. R package version 1.2-12, URL https://CRAN.R-project.org/package=PTAk.

Leibovici DG (2010). "Spatio-Temporal Multiway Data Decomposition Using Principal Tensor Analysis on k-Modes: The R Package **PTAk**." *Journal of Statistical Software*, **34**(10), 1–34. doi:10.18637/jss.v034.i10.

Li J, Bien J, Wells M (2018). ***rTensor**: Tools for Tensor Analysis and Decomposition*. R package version 1.4, URL https://CRAN.R-project.org/package=rTensor.

Lock EF, Nobel AB, Marron JS (2011). "Comment on "Population Value Decomposition, a Framework for the Analysis of Image Populations" by Crainiceanu et al." *Journal of the American Statistical Association*, **106**(495), 798–802. doi:10.1198/jasa.2011.ap11236.

Lu H, Plataniotis K, Venetsanopoulos A (2008). "MPCA: Multilinear Principal Component Analysis of Tensor Objects." *IEEE Transactions on Neural Networks*, **19**(1), 18–39. doi:10.1109/tnn.2007.901277.

Lu H, Plataniotis K, Venetsanopoulos A (2011). "A Survey of Multilinear Subspace Learning for Tensor Data." *Pattern Recognition*, **44**(7), 1540–1551. doi:10.1016/j.patcog.2011.01.004.

Martin C, Shafer R, LaRue B (2013). "An Order-p Tensor Factorization with Applications in Imaging." *SIAM Journal on Scientific Computing*, **35**(1), A474–A490. doi:10.1137/110841229.

Morup M (2011). "Applications of Tensor (Multiway Array) Factorizations and Decompositions in Data Mining." *WIREs Data Mining Knowledge Discovery*, **1**(1), 24–40. doi:10.1002/widm.1.

Phan AH, Tichavsky P, Cichocki A (2013). "Fast Alternating LS Algorithms for High Order CANDECOMP/PARAFAC Tensor Factorizations." *IEEE Transactions on Signal Processing*, **61**(19), 4834–4846. doi:10.1109/tsp.2013.2269903.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Sheehan BN, Saad Y (2007). "Higher Order Orthogonal Iteration of Tensors (HOOI) and Its Relation to PCA and GLRAM." In *SDM*, pp. 355–365. SIAM.

The MathWorks Inc (2017). *MATLAB – The Language of Technical Computing, Version R2017b*. Natick. URL http://www.mathworks.com/products/matlab/.

Tucker LR (1966). "Some Mathematical Notes on Three-Mode Factor Analysis." *Psychometrika*, **31**(3), 279–311. doi:10.1007/bf02289464.

Van den Boogaart KG (2010). ***tensorA**: Advanced Tensors Arithmetic with Named Indices*. R package version 0.36, URL https://CRAN.R-project.org/package=tensorA.

Vannieuwenhoven N, Vandebril R, Meerbergen K (2012). "A New Truncation Strategy for the Higher-Order Decomposition." *SIAM Journal on Scientific Computing*, **34**(2), A1027–A1052. doi:10.1137/110836067.

Vasilescu M (2009). *A Multilinear (Tensor) Algebraic Framework for Computer Graphics, Computer Vision, and Machine Learning*. Ph.D. thesis, Department of Computer Science, University of Toronto.

Veldhuizen T, Cummings J, Guio P, Stokes A, Shende S (2011). *The **Blitz++** Library*. URL http://blitz.sourceforge.net/.

Yang J, Zhang D, Frangi A, Yang J (2004). "Two-Dimensional PCA: A New Approach to Appearance-Based Face Representation and Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **28**(1), 131–137. doi:10.1109/tpami.2004.1261097.

Ye J (2005). "Generalized Low Rank Approximations of Matrices." *Machine Learning*, **61**(1–3), 167–191. doi:10.1007/s10994-005-3561-6.

Yilmaz KY, Cemgil AT, Simsekli U (2011). "Generalised Coupled Tensor Factorisation." In J Shawe-Taylor, RS Zemel, P Bartlett, FCN Pereira, KQ Weinberger (eds.), *Advances in Neural Information Processing Systems 24*, pp. 2151–2159. NIPS.

Zhang D, Zhou Z (2005). "$(2D)^2$ PCA: Two-Directional Two-Dimensional PCA for Efficient Face Representation and Recognition." *Neurocomputing*, **69**(1–3), 224–231. doi:10.1016/j.neucom.2005.06.004.

Affiliation:

James Li
Facebook
E-mail: jamesyili@gmail.com
URL: http://jamesyili.github.io/rTensor/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2018, Volume 87, Issue 10 *Submitted:* 2016-02-24
doi:10.18637/jss.v087.i10 *Accepted:* 2017-06-12