# PyTensor: A Python based Tensor Library

**Ji Oh Yoo** **Arvind Ramanathan**[†]
**Christopher J. Langmead**[‡]

February 2010
CMU-CS-10-102

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[†]Joint CMU-Pitt Program in Computational Biology, Carnegie Mellon University, Pittsburgh, PA, USA
[‡]Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
Email: cjl@cs.cmu.edu

This research was supported by NIH RC2 GM093307-0110, Oak Ridge National Lab's LDRD program, the National Science Foundation through TeraGrid resources provided by TACC and PSC, and the Howard Hughes Medical Institute.

**Keywords:** tensor analysis, multi-way analysis, proteins, molecular dynamics, MD-simulations, online analysis

**Abstract**

Multi-dimensional (or multi-way) analysis have evinced considerable interest within the data-mining community. In this technical report, we describe the development and implementation of a python based library to perform multi-way analysis. Our results show that the python implementation runs comparable to the MATLAB implementation. The implementation is available from http://pytensor.googlecode.com/.

# 1 Introduction

With the proliferation of a number of protein structures in the PDB database [2], efforts are now on to systematically understand the relationship between structure and function [7]. A recent and widely acknowledged belief is that local dynamics (local changes to hydrogen bonds and hydrophobic interactions) drives global dynamics (large scale motions including that of domains) [9] and hence, its function [1]. To fully appreciate a protein's structure-function relationship, it would be essential to understand the intrinsic dynamics of proteins. Proteins, even under equilibrium conditions (constant temperature, pressure and solvent/ chemical conditions) undergo a wide range of motions in varying time-scales. Some of these motions may involve bond-stretching/ vibrations and have a time scale of typically a few femto-seconds, where as other motions, including breathing motions or rearrangements of subdomains may have a time-scale of micro- to milli-seconds. The wide gap in time-scales is often a problem in relating the dynamics of a protein to its function, and hence statistical sampling techniques such as Molecular Dynamics (MD) and/ or Monte-Carlo (MC) simulations are used in understanding the dynamics of a protein and how it may relate to its function.

To analyze MD simulations as they are progressing, we recently introduced a method based on Dynamic Tensor Analysis (DTA) [8]. The approach builds multi-dimensional representations of a protein conformation using tensors. Using the approach, we [6, 4, 5] were able to successfully (a) identify residues that play a crucial role in protein dynamics, (b) identify regions of the protein that may exhibit coupled motions and (c) identify time-points along the trajectory where significant changes in collective dynamics have taken place. The above implementation used a MATLAB library described in [3]. However, with increasing support from the Python community for numerical and scientific libraries, it is useful to provide an implementation of the tensor library and its function in Python. In this report, we present the design and implementation of a Python based tensor library that is based on the MATLAB tensor-toolkit library, available from Sandia National Labs (http://csmr.ca.sandia.gov/ tgkolda/TensorToolbox/). All the top level functions are based on the library while providing similar function calls to maintain compatibility with the MATLAB toolkit. We believe that the availability of such a library will benefit the community as a whole in terms of providing an open-source python implementation of tensor operations as well as functions.

# 2 Organization of Tensor Library in Python

Python offers an inbuilt library called **`numpy`** to manipulate multi-dimensional arrays. The organization and use of this library is a primary requirement for developing the **`pytensor`** library. A brief overview of the functions that are available as part of **`numpy`** are explained here for completeness. More information regarding the use of the **`numpy`** can be found in the Numpy reference.

A class diagram showing the implementation and design of **`pytensor`** library is shown in Figure 2.

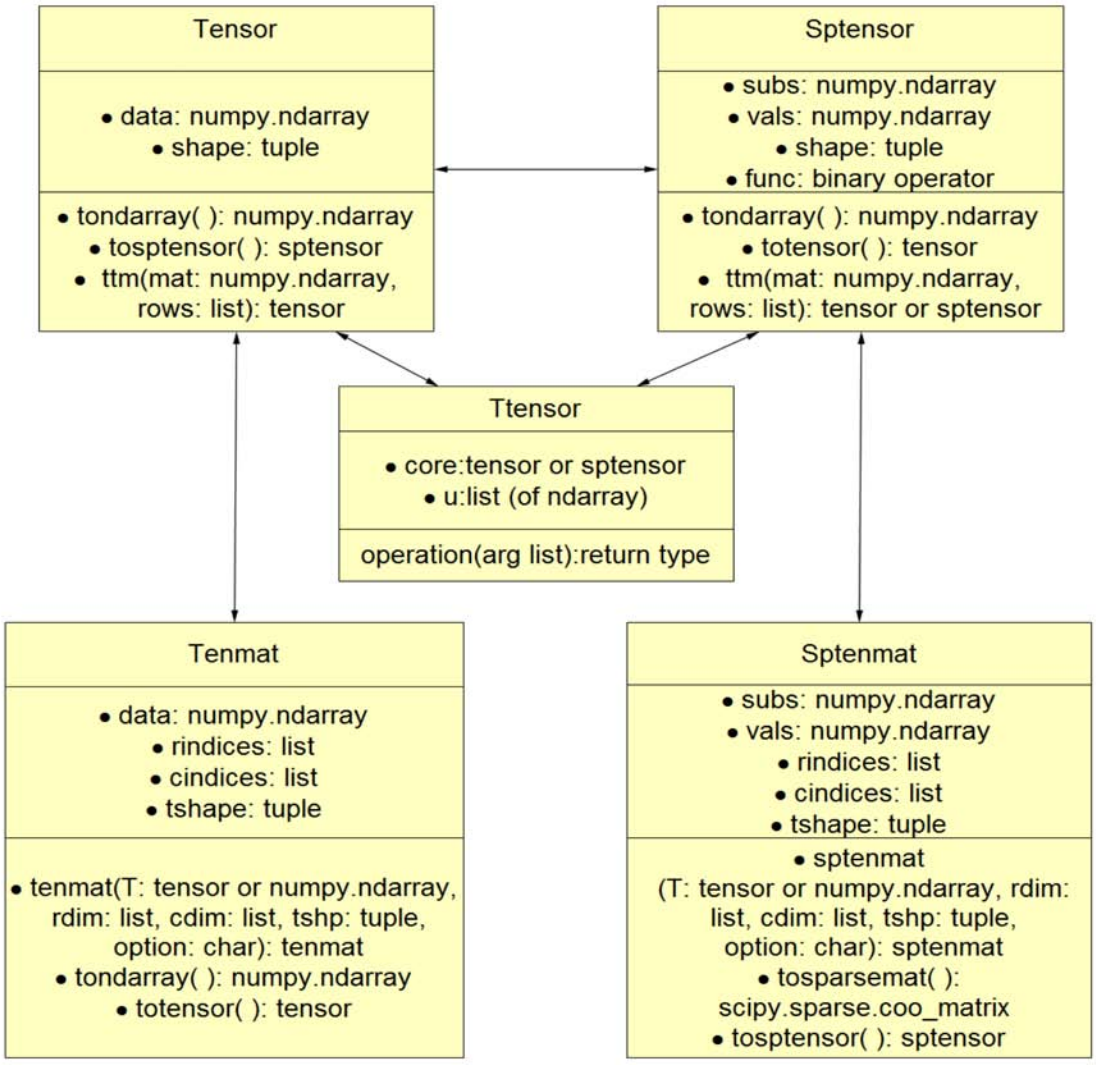


Figure 1: Class implementation and relationships in **`pytensor`**

.

# 3 Class Description and Basic Operations

## 3.1 Tensor

Tensor is a class that represents the n-dimensional array `numpy.ndarray`.

### Attributes

- `data (numpy.ndarray)`

  : n-dimensional array that the tensor object represents.

- `shape (tuple)`

  : shape of the tensor

### Functions

- `__init__(self, data, shape = None)`

  : Constructor for tensor object. Data is a n-dimensional array (numpy.ndarray) and shape is a shape (can be tuple, list) of the tensor.

- `size(self)`

  : Returns the number of elements of the tensor.

- `copy(self)`

  : Returns the copied tensor object of the tensor.

- `ndims(self)`

  : Returns the number of dimensions of the tensor.

- `dimsize(self, ind)`

  : Returns the size of the specified dimension. Same as `shape[ind]`.

- `tosptensor(self)`

  : Returns the sptensor object that has the same values with the tensor.

- `permute(self, order)`

  : Returns the tensor object that is permuted by the given order (list).

- `ipermute(self, order)`

  : Returns the tensor object that is permuted by the inverse of the given order (list).

- `ttm(self, mat, dims = None, option = None)`

  : Returns the tensor object that is the result of the tensor multiplied by the given matrix (numpy.ndarray). `dims` is the list of integers that represent the dimensions for rows that are used to matricize the tensor object. If `option = t`, then the dims represent the dimensions for columns to matricize the tensor. If mat is a list of matrices, then dims need to be list of row indices, and the same option can be applied.

- `tondarray(self )`

  : Returns the numpy.ndarray object that contains the same value with the tensor

**Special Constructors**

- `tenzeros(shp)`

  : Returns a tensor with the shape filled with zeroes.

- `tenones(shp)`

  : Returns a tensor with the shape filled with ones.

- `tenrand(shp)`

  : Returns a tensor with the shape filled with random number between 0 and 1.

- `tendiag(vals, shp=None)`

  : Returns a tensor with the given shape, which has vals as its diagonal entries

## 3.2 Sptensor

Sptensor is a class that represents the sparse tensor (with many zero elements). It does not store the whole values of the tensor object but stores the non-zero values and the corresponding coordinates of them.

**Attributes**

- `vals (numpy.ndarray)`

  : 1-dimensional array of non-zero values of the sptensor.

- `subs (numpy.ndarray)`

  : 2-dimensional array of coordinates of the values in vals.

- `shape(tuple)`

  : shape of the sptensor.

- `func(binary operator)`

  : function that is used to construct the sptensor as an accumulator.

**Functions**

- `__init__(self, subs, vals, shape = None, func=sum.__call__)`

  : Consturctor for the sptensor class. `subs` and `vals (numpy.ndarray)` or `(list)` are coordinates and values of the `sptensor`.

- `tondarray(self)`

  : Returns a numpy.ndarray object that has the same values with the sptensor.

- `ttm(self, mat, dims = None, option = None)`

  : Returns the tensor object that is the result of the tensor multiplied by the given matrix (numpy.ndarray). dims is the list of integers that represent the dimensions for rows that are used to matricize the tensor object. If option = t, then the dims represent the dimensions for columns to matricize the tensor. If mat is a list of matrices, then dims need to be list of row indices, and the same option can be applied.

- `permute(self, order)`

  : Returns the sptensor object that is permuted by the given order (list).

- `ipermute(self, order)`

  : Returns the sptensor object that is permuted by the inverse of the given order (list).

- `copy(self)`

  : Returns the copied sptensor object of the sptensor.

- `totensor(self)`

  : Returns the tensor object that has the same values with the sptensor.

- `nnz(self)`

  : Returns the number of non-zero elements in the sptensor.

- `ndims(self)`

  : Returns the number of dimensions of the tensor.

- `dimsize(self, ind)`

  : Returns the size of the specified dimension. Same as shape[ind].

**Special Constructors**

- `sptendiag(vals, shape = None)`

  : Construct a sptensor object that has given values in its diagonal entries.

## 3.3 tenmat

Tenmat is a class that represents the matricized (2-dimensionally unfolded) version of a tensor object. Representation of tenmat depends on the row and column indices used for matricize the tensor.

### Attributes

- `data (numpy.ndarray)`

  : 2-dimensional array that represents the matricized tensor.

- `rindices (numpy.ndarray)`

  : 1-dimensional array of row indices used to matricize the tensor.

- `cindices (numpy.ndarray)`

  : 1-dimensional array of column indices used to matricize the tensor.

- `tshape (tuple)`

  : shape of the original tensor.

### Functions

- `__init__(self, T, rdim = None, cdim = None, tshp = None, option = None)`

  : `rdim` and `cdim` are the row and column indices to matricize the tensor. If `rdim` is given but `cdim` is None, then `cdim` contains the dimensions not in `rdim`. If `cdim` is given but `rdim` is None and `option = t`, then `rdim` contains the dimensions not in `cdim`. If `rdim` has one dimension and `option = fc`, then `rdim` is to be `[rdim, rdim+1, , n-1, 0, , rdim-1]`. If `rdim` has one dimension and `option = bc`, then `rdim` is to be `[rdim, rdim-1, , 0, n-1, , rdim+1]`.

- `copy(self)`

  : Returns the copied tenmat object of the tenmat.

- `totensor(self)`

  : Returns the original tensor object of the tenmat.

- `tondarray(self)`

  : Returns the (2-dimensional) numpy.ndarray object that has the same values with the tenmat.

## 3.4 sptenmat

Sptenmat is a class that represents the matricized (2-dimensionally unfolded) version of a sptensor object. Representation of sptenmat depends on the row and column indices used for matricize the sptensor.

### Attributes

- `subs (numpy.ndarray)`

  : array that represents the coordinates of the matricized sptensor.

- `vals (numpy.ndarray)`

  : array that represents the values of the matricized sptensor.

- `rindices (numpy.ndarray)`

  : 1-dimensional array of row indices used to matricize the sptensor.

- `cindices (numpy.ndarray)`

  : 1-dimensional array of column indices used to matricize the sptensor.

- `tshape (tuple)`

  : shape of the original sptensor.

### Functions

- `__init__(self, T, rdim = None, cdim = None, tshp = None, option = None)`

  : rdim and cdim are the row and column indices to matricize the tensor. If rdim is given but cdim is None, then cdim contains the dimensions not in rdim. If cdim is given but rdim is None and option = t, then rdim contains the dimensions not in cdim. If rdim has one dimension and option = fc, then rdim is to be [rdim, rdim+1, , n-1, 0, , rdim-1]. If rdim has one dimension and option = bc, then rdim is to be [rdim, rdim-1, , 0, n-1, , rdim+1].

- `tosptensor(self)`

  : Returns the original sptensor object of the sptenmat.

- `tosparsemat(self)`

  : Returns the (2-dimensional) scipy.sparse object that has the same values with the sptenmat.

### 3.5 ttensor

Ttensor represents the Tucker decomposition of a tensor or a sptensor object. Tucker decomposition decomposes a tensor/sptensor into a core tensor/sptensor and a list of 2-dimensional factoring matrices, whose number is same with the number of dimension of the original tensor/sptensor object. If the core tensor gets multiplied by the factoring matrices, the original tensor/sptensor object is returned.

#### Attributes

- `core (tensor or sptensor)`
  : core tensor/sptensor of the ttensor object.
- `u (list of numpy.ndarray)`
  : list of 2-dimensional numpy.ndarray that are the factoring matrices of the ttensor object.

#### Functions

- `__init__(self, core, uIn)`
  :
- `size(self)`
  : Returns the number of elements of the original tensor/sptensor object.
- `totensor(self)`
  : Returns the original tensor object that is represented by the ttensor object.
- `dimsize(self, ind)`
  : Returns the size of the specified dimension. Same as shape[ind].

## 4 Usage

### 4.1 tensor and sptensor

```
dat = numpy.arange(24).resize([2,3,4])
ten1 = tensor.tensor(dat)
#same with ten1 = tensor.tensor(dat, [2,3,4])
print ten1
print ten1.size()
print ten1.ndims()
spten1 = ten.tosptensor()
print spten1
```

```
vals = numpy.array([[0.5], [1.5], [2.5], [3.5], [4.5], [5.5]])
subs = numpy.array([[1,1,1], [0,0,0], [1,2,3], [1,0,1], [1,1,2], [1,1,1]])
spten2 = sptensor.sptensor(subs, vals)
print spten2
#same with spten2 = sptensor.sptensor(subs, vals, [2,3,4])
print spten2.nnz()
print spten2.ndims()
ten2 = spten2.totensor()
print ten2

A = numpy.arange(18).reshape([6,3]);
print ten1.ttm(A,1);
B = numpy.arange(12).reshape([3,4]);
print ten1.ttm([A,B],[1,2]);
print obj.ttm([A.transpose(),B.transpose()],[1,2],'t');

print spten2.ttm(A,1);
print spten2.ttm([A,B],[1,2]);
print spten2.ttm([A.transpose(),B.transpose()],[1,2],'t');
```

## 4.2 tenmat and sptenmat

```
dat = numpy.arange(24).resize([2,3,4])
ten1 = tensor.tensor(dat)
print tenmat.tenmat(ten1, [1,0])
print tenmat.tenmat(ten1, [0], [1,2])
print tenmat.tenmat(ten1, None, [1], t)

vals = numpy.array([[0.5], [1.5], [2.5], [3.5], [4.5], [5.5]])
subs = numpy.array([[1,1,1], [0,0,0], [1,2,3], [1,0,1], [1,1,2], [1,1,1]])
spten2 = sptensor.sptensor(subs, vals)
print sptenmat.sptenmat(spten2, [1,0])
print sptenmat.sptenmat(spten2, [0], [1,2])
print sptenmat.sptenmat(spten2, None, [1], t)
```

## 4.3 Ttensor

```
arr = numpy.arange(24).reshape([2,3,4]);
A = numpy.array([[1,2],[3,4],[5,6]]);
B = numpy.array([[1,2,3],[4,5,6]]);
C = numpy.array([[1,2,3,4]]);
obj = ttensor.ttensor(tensor.tensor(arr), [A, B, C]);
```

```
print obj;
print obj.totensor();
```

### 4.4 Dynamic Tensor Analysis

```
A = ttensor.ttensor(tensor.tenrands([2,3,4]),
                    [numpy.random.random([10,2]),
                    numpy.random.random([30,3]),
                    numpy.random.random([40,4])]).totensor();

    [a,b] = DTA.DTA(A, [1,2,3]);
    #print a;
    #print b;

    Core = numpy.arange(24).reshape([2,3,4]);
    u1 = numpy.array([[1,2],[3,4]]);
    u2 = numpy.array([[0,1,0],[1,0,1],[1,1,1]]);
    u3 = numpy.array([[1,1,1,1],[1,2,3,4],[1,1,1,1]]);
    tt = ttensor.ttensor(tensor.tensor(Core), [u1,u2,u3]);

    print tt;
    [a,b] = DTA.DTA(tt.totensor(), [1,2,3]);
    print a;
    print b;
```

# References


[1] P. K. Agarwal. Enzymes: An integrated view of structure, dynamics and function. *Microbial Cell Factories*, 5, 2006.

[2] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2002.

[3] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. Technical report, Sandia National Laboratories, 2007.

[4] Arvind Ramanathan, P. K. Agarwal, M. Kurnikova, and C. J. Langmead. An online approach for mining collective behaviors from molecular dynamics simulations. In S. Batzoglou, editor, *Lecture Notes in Computer Science*, volume 5541, pages 138–154, 2009.

[5] Arvind Ramanathan, P. K. Agarwal, M. Kurnikova, and C. J. Langmead. An online approach for mining collective behaviors from molecular dynamics simulations. *J. Comp. Bio.*, (in press), 2010.

[6] Arvind Ramanathan, P. K. Agarwal, and C. J. Langmead. Using tensor analysis to characterize contact-map dynamics of proteins. Technical Report CMU-CS-08-109, Carnegie Mellon University, 2008.

[7] Tamar Schlick, R. D. Skeel, A. T. Brunger, L. V. Kale, J. Hermans, K. Schulten, and J. A. Board, Jr. Algorithmic challenges in computational molecular biophysics. *J. Comp. Phys.*, 151:9–48, 1999.

[8] Jimeng Sun, Spiros Papadimitrou, and Christos Faloutsos. Distributed pattern discovery in multiple streams. Technical report, Carnegie Mellon University, 2006.

[9] L. Yang, E. Eyal, C. Chennubhotla, J. Lee, A. M. Gronenborn, and I. Bahar. Insights into equilibrium dynamics of proteins from comparison of nmr and x-ray data with computational predictions. *Structure*, 15:1–9, 2007.