# ADVANCED ALGORITHMS AND PROGRAMMING METHODS -2

Professor Andrea Torsello

**[Templated library handling tensors, i.e., multidimensional arrays]**

- Shehzad Inayat: 888374
- Brahmashwini Regonda: 887689

# Abstract

Implement a templated library handling tensor (multi-multidimensional arrays), where Tensors are specified by the type they are holding, their rank.

The tensor class must provide:

**Direct access:** Must be able to directly access any entry of the tensor by providing all the indexes

**Slicing:** Can fix one (or more) index producing a lower rank tensor sharing the same data

The library must provide also a product operation that takes two tensor and a list of pairs of indices (one from the first tensor and one for the second and computes the contraction over those indices, i.e it should sum over those indices the product of the entries of the matrices, producing a new tensor with the indices that are not contracted in the two matrices.

As an example, let A be a rank 3 matrix of dimensions 2, 3, and 4, and let B be a rank 4 tensor of dimensions 2,3,5,2, and suppose that we invoke the product of these two with indices {(0,3), (1,1)}, then the first index of A will be contracted with the last of B, and the second of A with the second of be, resulting in a new tensor of rank 3 and dimensions 4,2,5.

The entries of this tensor (that we will call C) are defined by the following equation

$$c_{i,j,k} = \sum_{n=0}^1\sum_{m=0}^2 a_{n,m,i} * b_{j,m,k,n}$$

# 1. Implementation

The description and explanation of templates, parameters and functions which we used in the header file are given below.

# tensor< T, Rank > Class Template Reference

Template container class for implementing a multi-dimensional array (or tensor) with fixed size.

```
#include <tensor.hpp>
```

## Public Member Functions

**tensor** (const std::array< std::size_t, Rank > &dims)
Construct a new tensor object using an array of dimension sizes.

template<typename... Args>
**tensor** (Args... args)
Construct a new tensor object using a variable argument dimensions.

template<typename... Args>
T & **operator()** (Args... args)
Access an item in the tensor object.

T & **operator()** (const std::array< std::size_t, Rank > &args)
Access an item in the tensor object.

template<typename... Args>
const T & **operator()** (Args... args) const
Access an item in the constant tensor object.

const T & **operator()** (const std::array< std::size_t, Rank > &args) const
Access an item in the tensor object.

template<std::size_t N>
**tensor**< T, Rank - N/2 > **slice** (const std::array< std::size_t, N > &args)
Slice the tensor object given an array of (dimension, index). The list of (dimension, index) are used to as reference for the list of items to retain.

template<typename... Args>

**tensor**< T, Rank - sizeof...(Args)/2 > **slice** (Args... args)

Slice the tensor object given an array of (dimension, index). The list of (dimension, index) are used to as reference for the list of items to retain.

const std::array< std::size_t, Rank > & **size** ()

Gets the array of dimension sizes of the tensor.

## Friends

template<std::size_t OtherRank, std::size_t N>

**tensor**< T, Rank+OtherRank - N > **tensor_product** (const **tensor**< T, Rank > &left, const **tensor**< T, OtherRank > &right, const std::array< std::size_t, N > &args)

Computes the tensor product. If c is the result, then the tensor

product is given as c_{i,j,k} = \sum_{n=0}^1\sum_{m=0}^2 a_{n,m,i} * b_{j,m,k,n}.

template<std::size_t OtherRank, typename... Args>

**tensor**< T, Rank+OtherRank - sizeof...(Args)> **tensor_product** (const **tensor**< T, Rank > &left, const **tensor**< T, OtherRank > &right, Args... args)

Computes the tensor product. If c is the result, then the tensor

product is given as c_{i,j,k} = \sum_{n=0}^1\sum_{m=0}^2 a_{n,m,i} * b_{j,m,k,n}.

## Detailed Description

template<typename T, std::size_t Rank>
class tensor< T, Rank >

Template container class for implementing a multi-dimensional array (or tensor) with fixed size.

**Template Parameters**

| | |
|---|---|
| **T** | typename of the tensor value |
| **Rank** | number of dimensions |

## Constructor & Destructor Documentation

### ◆ tensor() [1/2]

```
template<typename T , std::size_t Rank>
tensor< T, Rank >::tensor ( const std::array< std::size_t, Rank > &  dims )    inline
```

Construct a new tensor object using an array of dimension sizes. Each element of the array is a corresponding dimension and the number of elements that can be placed in the size is the number given.

**Parameters**

| | |
|---|---|
| **dims** | array of dimension sizes |

### ◆ tensor() [2/2]

```
template<typename T , std::size_t Rank>
template<typename... Args>
tensor< T, Rank >::tensor ( Args...  args )    inline
```

Construct a new tensor object using a variable argument dimension. Instead of an array, it allows the user to input the sizes of each dimension using function arguments.

**Template Parameters**

| | |
|---|---|
| **Args** | variable size arguments of the dimension sizes. |

**Parameters**

| | |
|---|---|
| **args** | variable list of dimensions sizes. |

## Member Function Documentation

### ◆ operator()() [1/4]

template<typename T , std::size_t Rank>
template<typename... Args>
T & tensor< T, Rank >::operator() ( Args... args ) inline

Access an item in the tensor object given the array of indices. As a multi-dimensional array, the number of arguments should be equal to the rank of the tensor. Each index should be within the bounds of the dimension.

**Template Parameters**

**Args** variable list of indices

**Parameters**

**args** variable list of indices

**Returns**

T& reference to the item

### ◆ operator()() [2/4]

template<typename T , std::size_t Rank>
template<typename... Args>
const T & **tensor**< T, Rank >::operator() ( Args... args ) const inline

Access an item in the constant tensor object.

**Template Parameters**

**Args** variable list of indices

**Parameters**

**args** variable list of indices

**Returns**

const T& constant reference to the item

## ◆ operator()() [3/4]

template<typename T , std::size_t Rank>
T & **tensor**< T, Rank >::operator() ( const std::array< std::size_t, Rank > & args ) inline

Access an item in the tensor object.

**Parameters**

**args** array of indices

**Returns**

T& reference to the item

## ◆ operator()() [4/4]

template<typename T , std::size_t Rank>
const T & **tensor**< T, Rank >::operator() ( const std::array< std::size_t, Rank > & args ) const inline

Access an item in the tensor object.

**Parameters**

**args** array of indices

**Returns**

T& reference to the item

## ◆ size()

template<typename T , std::size_t Rank>

`const std::array< std::size_t, Rank > & tensor< T, Rank >::size ( )` inline

Gets the array of dimension sizes of the tensor.

**Returns**

const std::array<std::size_t, Rank> array of dimensions.

## ◆ slice() [1/2]

template<typename T , std::size_t Rank>

template<typename... Args>

`tensor< T, Rank - sizeof...(Args)/2 > tensor< T, Rank >::slice ( Args... args )` inline

Slice the tensor object given an array of (dimension, index). The list of (dimension, index) are used to as reference for the list of items to retain.

**Template Parameters**

**variable** list of arguments

**Parameters**

**args** list of (dimension, index) tuples

**Returns**

tensor<T, Rank - N(args) / 2> reduced tensor object

## ◆ slice() [2/2]

template<typename T , std::size_t Rank>

template<std::size_t N>

**tensor< T, Rank - N/2 > tensor< T, Rank >::slice ( const std::array< std::size_t, N > & args )** inline

Slice the tensor object given an array of (dimension, index). The list of (dimension, index) are used to as reference for the list of items to retain.

**Template Parameters**

**N** length of the args.

**Parameters**

**args** array of (dimension, index) tuples

**Returns**

tensor<T, Rank - N / 2> reduced tensor object

## Friends And Related Function Documentation

◆ tensor_product [1/2]

```
template<typename T , std::size_t Rank>
template<std::size_t OtherRank, typename... Args>
tensor< T, Rank+OtherRank - sizeof...(Args)> tensor_product ( const tensor< T, Rank > &       left,
                                                              const tensor< T, OtherRank > &  right,
                                                              Args...                         args
                                                            )
```

friend

Computes the tensor product. If c is the result, then the tensor product is given as c_{i,j,k} = \sum_{n=0}^1\sum_{m=0}^2 a_{n,m,i} * b_{j,m,k,n}.

**Template Parameters**

| | |
|---|---|
| **OtherRank** | Rank of tensor 2 |
| **Args** | list of indices |

**Parameters**

| | |
|---|---|
| **left** | tensor 1 |
| **right** | tensor 2 |
| **args** | number of indices to contract summed over |

**Returns**

tensor<T, Rank + OtherRank - sizeof...(Args)> product

## ◆ tensor_product [2/2]

```
template<typename T , std::size_t Rank>
template<std::size_t OtherRank, std::size_t N>
tensor< T, Rank+OtherRank - N > tensor_product ( const tensor< T, Rank > &            left,
                                                 const tensor< T, OtherRank > &       right,
                                                 const std::array< std::size_t, N > & args
                                               )
```

friend

Computes the tensor product. If c is the result, then the tensor product is given as c_{i,j,k} = \sum_{n=0}^1\sum_{m=0}^2 a_{n,m,i} * b_{j,m,k,n}.

**Template Parameters**

| | |
|---|---|
| **OtherRank** | Rank of the other tensor |
| **N** | number of indices |

**Parameters**

| | |
|---|---|
| **left** | tensor 1 |
| **right** | tensor 2 |
| **args** | pairs of indices to contract (summed over) |

**Returns**

tensor<T, Rank + OtherRank - N> product