# Multidimensional Tensor Library for perturbation methods applied to DSGE models

Ondra Kamenik

2004

## 1 Introduction

The design of the library was driven by the needs of perturbation methods for solving Stochastic Dynamic General Equilibrium models. The aim of the library is not to provide an exhaustive interface to multidimensional linear algebra. The tensor library's main purposes include:

- Define types for tensors, for a multidimensional index of a tensor, and types for folded and unfolded tensors. The tensors defined here have only one multidimensional index and one reserved one-dimensional index. The tensors should allow modelling of higher order derivatives with respect to a few vectors with different sizes (for example $\left[g_{y^2u^3}\right]$). The tensors should allow folded and unfolded storage modes and conversion between them. A folded tensor stores symmetric elements only once, while an unfolded stores data as a whole multidimensional cube.

- Define both sparse and dense tensors. We need only one particular type of sparse tensor. This in contrast to dense tensors, where we need much wider family of types.

- Implement the Faa Di Bruno multidimensional formula. So, the main purpose of the library is to implement the following step of Faa Di Bruno:

$$[B_{s^k}]_{\alpha_1\ldots\alpha_k} = \left[h_{y^l}\right]_{\gamma_1\ldots\gamma_l} \left( \sum_{c\in M_{l,k}} \prod_{m=1}^{l} [g_{s^{|c_m|}}]^{\gamma_m}_{c_m(\alpha)} \right)$$

  where $s$ can be a compound vector of variables, where $h_{y^l}$ and $g_{s^i}$ are tensors, $M_{l,k}$ is a set of all equivalences of $k$ element set having $l$ classes, $c_m$ is $m$-th class of equivalence $c$, $|c_m|$ is its cardinality, and $c_m(\alpha)$ is a tuple of picked indices from $\alpha$ by class $c_m$.

  Note that the sparse tensors play a role of $h$ in the Faa Di Bruno, not of $B$ nor $g$.

## 2 Classes

The following list is a road-map to various classes in the library.

**Tensor** Virtual base class for all dense tensors, defines *index* as the multidimensonal iterator

**FTensor** Virtual base class for all folded tensors

**UTensor** Virtual base class for all unfolded tensors

**FFSTensor** Class representing folded full symmetry dense tensor, for instance $\left[g_{y^3}\right]$

**FGSTensor** Class representing folded general symmetry dense tensor, for instance $\left[g_{y^2u^3}\right]$

**UFSTensor** Class representing unfolded full symmetry dense tensor, for instance $\left[g_{y^3}\right]$

**UGSTensor** Class representing unfolded general symmetry dense tensor, for instance $\left[g_{y^2u^3}\right]$

**URTensor** Class representing unfolded/folded full symmetry, row-oriented, dense tensor. Row-oriented tensors are used in the Faa Di Bruno above as some part (few or one column) of a product of $g$'s. Their fold/unfold conversions are special in such a way, that they must yield equivalent results if multiplied with folded/unfolded column-oriented counterparts.

**URSingleTensor** Class representing unfolded/folded full symmetry, row-oriented, single column, dense tensor. Besides use in the Faa Di Bruno, the single column row oriented tensor models also higher moments of normal distribution.

**UPSTensor** Class representing unfolded, column-oriented tensor whose symmetry is not that of the $\left[B_{y^2u^3}\right]$ but rather of something as $[B_{yuuyu}]$. This tensor evolves during the product operation for unfolded tensors and its basic operation is to add itself to a tensor with nicer symmetry, here $\left[B_{y^2u^3}\right]$.

**FPSTensor** Class representing partially folded, column-oriented tensor whose symmetry is not that of the $\left[B_{y^3u^4}\right]$ but rather something as $\left[B_{yu|y^3u|u^4}\right]$, where the portions of symmetries represent folded dimensions which are combined in unfolded manner. This tensor evolves during the Faa Di Bruno for folded tensors and its basic operation is to add itself to a tensor with nicer symmetry, here folded $\left[B_{y^3u^4}\right]$.

**USubTensor** Class representing unfolded full symmetry, row-oriented tensor which contains a few columns of huge product $\prod_{m=1}^{l}\left[g_{s^{|c_m|}}\right]^{\gamma_m}_{c_m(\alpha)}$. This is needed during the Faa Di Bruno for folded matrices.

**IrregTensor** Class representing a product of columns of derivatives $\left[z_{y^ku^l}\right]$, where $z=[y^T,v^T,w^T]^T$. Since the first part of $z$ is $y$, the derivatives contain many zeros, which are not stored, hence the tensor's irregularity. The tensor is used when calculating one step of Faa Di Bruno formula, i.e. $\left[h_{y^l}\right]_{\gamma_1\dots\gamma_l}\left(\sum_{c\in M_{l,k}}\prod_{m=1}^{l}\left[g_{s^{|c_m|}}\right]^{\gamma_m}_{c_m(\alpha)}\right)$.

**FSSparseTensor** Class representing full symmetry, column-oriented, sparse tensor. It is able to store elements keyed by the multidimensional index, and multiply itself with one column of row-oriented tensor.

**FGSContainer** Container of `FGSTensor`s. It implements the Faa Di Bruno with unfolded or folded tensor $h$ yielding folded $B$. The methods are `FGSContainer::multAndAdd`.

**UGSContainer** Container of `UGSTensor`s. It implements the Faa Di Bruno with unfolded tensor $h$ yielding unfolded $B$. The method is `UGSContainer::multAndAdd`.

**StackContainerInterface** Virtual pure interface describing all logic of stacked containers for which we will do the Faa Di Bruno operation.

**UnfoldedStackContainer** Implements the Faa Di Bruno operation for stack of containers of unfolded tensors.

**FoldedStackContainer** Implements the Faa Di Bruno for stack of containers of folded tensors.

**ZContainer** The class implements the interface `StackContainerInterface` according to $z$ appearing in context of DSGE models. By a simple inheritance, we obtain `UnfoldedZContainer` and also `FoldedZContainer`.

**GContainer** The class implements the interface `StackContainerInterface` according to $G$ appearing in context of DSGE models. By a simple inheritance, we obtain `UnfoldedGContainer` and also `FoldedGContainer`.

**Equivalence** The class represents an equivalence on $n$-element set. Useful in the Faa Di Bruno.

**EquivalenceSet** The class representing all equivalences on $n$-element set. Useful in the Faa Di Bruno.

**Symmetry** The class defines a symmetry of general symmetry tensor. This is it defines a basic shape of the tensor. For $\left[B_{y^2u^3}\right]$, the symmetry is $y^2u^3$.

**Permutation** The class represents a permutation of $n$ indices. Useful in the Faa Di Bruno.

**IntSequence** The class represents a sequence of integers. Useful everywhere.

**TwoDMatrix and ConstTwoDMatrix** These classes provide an interface to a code handling two-dimensional matrices. The code resides in Sylvester module, in directory `sylv/cc`. There is no similar interface to `Vector` and `ConstVector` classes from the Sylvester module and they are used directly.

**KronProdAll** The class represents a Kronecker product of a sequence of arbitrary matrices and is able to multiply a matrix from the right without storing the Kronecker product in memory.

**KronProdAllOptim** The same as `KronProdAll` but it optimizes the order of matrices in the product to minimize the used memory during the Faa Di Bruno operation. Note that it is close to optimal flops.

**FTensorPolynomial and UTensorPolynomial** Abstractions representing a polynomial whose coefficients are folded/unfolded tensors and variable is a column vector. The classes provide methods for traditional and horner-like polynomial evaluation. This is useful in simulation code.

**FNormalMoments and UNormalMoments** These are containers for folded/unfolded single column tensors for higher moments of normal distribution. The code contains an algorithm for generating the moments for arbitrary covariance matrix.

**TLStatic** The class encapsulates all static information needed for the library. It includes precalculated equivalence sets.

**TLException** Simple class thrown as an exception.

# 3 Multi-threading

The tensor library is multi-threaded. The basic property of the thread implementation in the library is that we do not allow running more concurrent threads than the preset limit. This prevents threads from competing for memory in such a way that the OS constantly switches among threads with frequent I/O for swaps. This may occur since one thread might need much

own memory. The threading support allows for detached threads, the synchronization points during the Faa Di Bruno operation are relatively short, so the resulting load is close to the preset maximum number parallel threads.

# 4 Tests

A few words to the library's test suite. The suite resides in directory `tl/testing`. There is a file `tests.cc` which contains all tests and `main()` function. Also there are files `factory.hh` and `factory.cc` implementing random generation of various objects. The important property of these random objects is that they are the same for all object's invocations. This is very important in testing and debugging. Further, one can find files `monoms.hh` and `monoms.cc`. See below for their explanation.

There are a few types of tests:

- We test for tensor indices. We go through various tensors with various symmetries, convert indices from folded to unfolded and vice-versa. We test whether their coordinates are as expected.
- We test the Faa Di Bruno by comparison of the results of `FGSContainer::multAndAdd` against the results of `UGSContainer::multAndAdd`. The two implementations are pretty different, so this is a good test.
- We use a code in `monoms.hh` and `monoms.cc` to generate a random vector function $f(x(y,u))$ along with derivatives of $[f_x]$, $\left[x_{y^k u^l}\right]$, and $\left[f_{y^k u^l}\right]$. Then we calculate the resulting derivatives $\left[f_{y^k u^l}\right]$ using `multAndAdd` method of `UGSContainer` or `FGSContainer` and compare the derivatives provided by `monoms`. The functions generated in `monoms` are monomials with integer exponents, so the implementation of `monoms` is quite easy.
- We do a similar thing for sparse tensors. In this case the `monoms` generate a function $f(y, v(y,u), w(y,u))$, provide all the derivatives and the result $\left[f_{y^k u^l}\right]$. Then we calculate the derivatives with `multAndAdd` of `ZContainer` and compare.
- We test the polynomial evaluation by evaluating a folded and unfolded polynomial in traditional and Horner-like fashion. This gives four methods in total. The four results are compared.