Vectorized algorithms for regular tessellations of d-orthotopes and their faces

François Cuvelier[*] Gilles Scarella [†]

2017/01/02

Abstract

Tessellation of hypercubes or orthotopes and all their faces in any dimension is a nice challenge. The purpose of this paper is to describe efficient vectorized algorithms to obtain regular tessellations made up by simplices or orthotopes. These vectorized algorithms have been implemented in array programming languages such as Matlab/Octave, Python.

Contents

[*]Université Paris 13, Sorbonne Paris Cité, LAGA, CNRS UMR 7539, 99 Avenue J-B Clément, F-93430 Villetaneuse, France, cuvelier@math.univ-paris13.fr

[†]Université Côte d'Azur, CNRS, LJAD, F-06108 Nice, France, gilles.scarella@unice.fr.

This work was partially supported by ANR Dedales.

Warning GIT! Working tree is dirty!!

GIT commit fccade66a71a15d72ac7eb0c8e0af85ff5efe09f

Date: Tue Dec 6 08:06:30 2016 +0100

mHyperMesh :commit 239dd7c5ba58a298e17dcd4837b584c2f6b36d37

pyHyperMesh :commit e17d0023c9895a9d659a40a25c49c22ac4571ace

1 Definitions and notations

1.1 d-orthotope and d-hypercube

We first recall the definitions of a d-orthotope and a d-hypercube

Definition 1. *In geometry, a* ***d-orthotope*** *(also called a hyperrectangle or a box) is the generalization of a rectangle for higher dimensions, formally defined as the Cartesian product of intervals.*

Definition 2. *A d-orthotope with all edges of the same length is a* ***d-hypercube****.*
A d-orthotope with all edges of length one is ***a unit d-hypercube****.*
The hypercube $[0,1]^d$ is called ***the unit d-hypercube*** *or the* ***unit reference d-hypercube****.*

The m-orthotopes on the boundary of a d-orthotope, $0 \leqslant m \leqslant d$, are called the **$m$-faces of the d-orthotope**.

The number of m-faces of a d-orthotope is

$$E_{m,d} = 2^{d-m}\binom{d}{m} \quad \text{where} \quad \binom{d}{m} = \frac{d!}{m!(d-m)!}$$

For example, the 2-faces of the unit 3-hypercube $[0,1]^3$ are the sets

$$\begin{array}{ll} \{0\}\times[0,1]\times[0,1], & \{1\}\times[0,1]\times[0,1], \\ [0,1]\times\{0\}\times[0,1], & [0,1]\times\{1\}\times[0,1], \\ [0,1]\times[0,1]\times\{0\}, & [0,1]\times[0,1]\times\{1\}. \end{array}$$

Its 1-faces are

$$\begin{array}{ll}
\{0\}\times\{0\}\times[0,1], & \{0\}\times\{1\}\times[0,1],\\
\{1\}\times\{0\}\times[0,1], & \{1\}\times\{1\}\times[0,1],\\
\{0\}\times[0,1]\times\{0\}, & \{0\}\times[0,1]\times\{1\},\\
\{1\}\times[0,1]\times\{0\}, & \{1\}\times[0,1]\times\{1\},\\
[0,1]\times\{0\}\times\{0\}, & [0,1]\times\{0\}\times\{1\},\\
[0,1]\times\{1\}\times\{0\}, & [0,1]\times\{1\}\times\{1\},
\end{array}$$

and its 0-faces are

$$\begin{array}{ll}
\{0\}\times\{0\}\times\{0\}, & \{1\}\times\{0\}\times\{0\},\\
\{0\}\times\{1\}\times\{0\}, & \{0\}\times\{0\}\times\{1\},\\
\{1\}\times\{1\}\times\{0\}, & \{1\}\times\{0\}\times\{1\},\\
\{0\}\times\{1\}\times\{1\}, & \{1\}\times\{1\}\times\{1\}.
\end{array}$$

We represent in Figure 1 all the m-faces of a 3D hypercube.

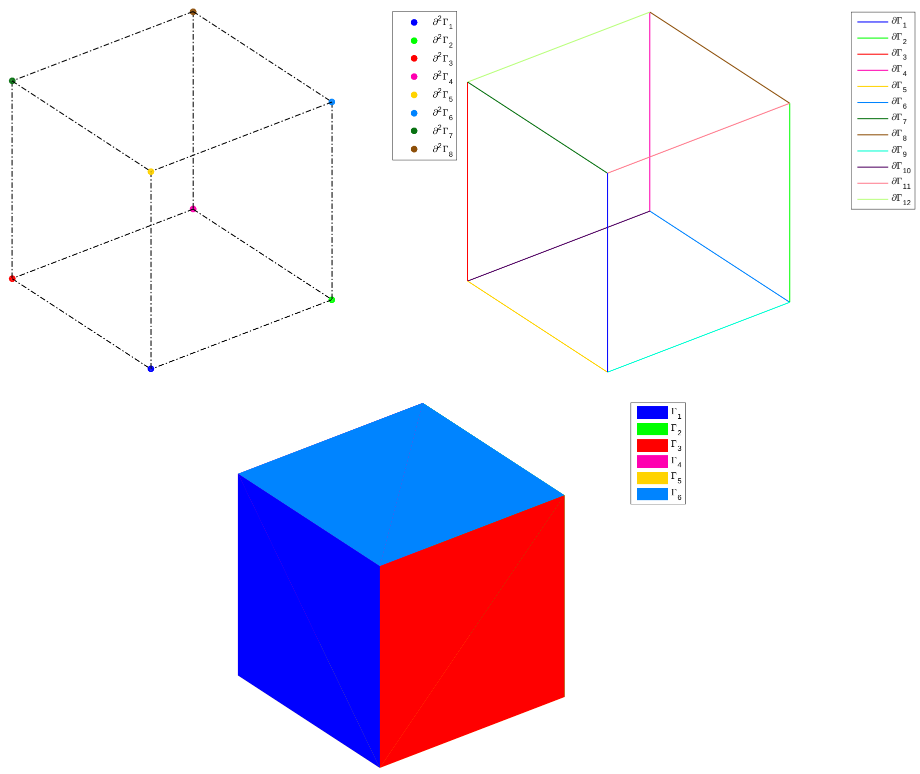


Figure 1: m-faces of a 3D hypercube : 0-faces (upper left), 1-faces (upper right) and 2-faces (bottom)

We give in Table 1 this number for $d \in [\![0,6]\!]$ and $0 \leqslant m \leqslant d$.

	m	0	1	2	3	4	5	6
d	Names	0-face	1-face	2-face	3-face	4-face	5-face	6-face
0	*Point*	1						
1	*Segment*	2	1					
2	*Square*	4	4	1				
3	*Cube*	8	12	6	1			
4	*Tesseract*	16	32	24	8	1		
5	*Penteract*	32	80	80	40	10	1	
6	*Hexeract*	64	192	240	160	60	12	1

Table 1: Number of m-faces of a d-hypercube

The identification/numbering of the m-faces is given in section2.4.

1.2 d-simplex

Definition 3. *In geometry, a* ***simplex*** *(plural: simplexes or simplices) is a generalization of the notion of a triangle or tetrahedron to arbitrary dimensions. Specifically, a d-simplex is a d-dimensional polytope which is the convex hull of its $d+1$ vertices. More formally, suppose the $d+1$ points $\mathbf{q}^0, \dots, \mathbf{q}^d \in \mathbb{R}^d$ are affinely independent, which means $\mathbf{q}^1 - \mathbf{q}^0, \dots, \mathbf{q}^d - \mathbf{q}^0$ are linearly independent. Then, the simplex determined by them is the set of points*

$$C = \{\theta_0 \mathbf{q}^0 + \dots + \theta_d \mathbf{q}^d | \theta_i \geqslant 0, 0 \leqslant i \leqslant d, \sum_{i=0}^{d} \theta_i = 1\}.$$

For example, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, and a 4-simplex is a 5-cell. A single point may be considered as a 0-simplex and a line segment may be considered as a 1-simplex. A simplex may be defined as the smallest convex set containing the given vertices.

Definition 4. *Let $\mathbf{q}^0, \dots, \mathbf{q}^d \in \mathbb{R}^d$ be the $d+1$ vertices of a d-simplex K and $\mathbb{D}_K$ be the $(d+1)$-by-$(d+1)$ matrix defined by*

$$\mathbb{D}_K = \left(\begin{array}{c|c|c} \mathbf{q}^0 & \dots & \mathbf{q}^d \\ \hline 1 & \dots & 1 \end{array}\right)$$

The d-simplex K is

- ***degenerated*** *if* $\det \mathbb{D}_K = 0$,
- ***positive oriented*** *if* $\det \mathbb{D}_K > 0$,
- ***negative oriented*** *if* $\det \mathbb{D}_K < 0$.

The m-simplices on the boundary of a d-simplex, $0 \leqslant m \leqslant d$, are called the **$m$-faces of the d-simplex**. If a d-simplex is nondegenerate, its number of m-faces is equal to the binomial coefficient $\begin{pmatrix} d+1 \\ m+1 \end{pmatrix}$

We give in Table 2 this number for $d \in [\![0, 6]\!]$ and $0 \leqslant m \leqslant d$.

	m	0	1	2	3	4	5	6
d	Names	0-face	1-face	2-face	3-face	4-face	5-face	6-face
0	*Point*	1						
1	*Segment*	2	1					
2	*triangle*	3	3	1				
3	*tetrahedron*	4	6	4	1			
4	4-*simplex*	5	10	10	5	1		
5	5-*simplex*	6	15	20	15	6	1	
6	6-*simplex*	7	21	35	35	21	7	1

Table 2: Number of m-faces of a nondegenerate d-simplex

2 Tessellation by d-orthotopes

2.1 The unit hypercube vertices

The unit d-dimensional hypercube $\widehat{\mathrm{H}} = [0,1]^d$ has $n = 2^d$ vertices. A vertex may be identified by a d-tuple $\boldsymbol{\imath} = (\boldsymbol{\imath}_1, \boldsymbol{\imath}_2, \cdots, \boldsymbol{\imath}_d) \in [\![0,1]\!]^d$ and we denoted by $\boldsymbol{x}^{\boldsymbol{\imath}} = (\boldsymbol{x}^{\boldsymbol{\imath}}_1, \ldots, \boldsymbol{x}^{\boldsymbol{\imath}}_d)^{\mathtt{t}} \in \mathbb{R}^d$ the vertex defined by

$$\boldsymbol{x}^{\boldsymbol{\imath}}_l = \boldsymbol{\imath}_l, \quad \forall l \in [\![1,d]\!].$$

Let $\mathcal{L}$ be the function mapping all the d-tuples $\boldsymbol{\imath} \in [\![0,1]\!]^d$ into $[\![1,2^d]\!]$ defined by

$$\mathcal{L}(\boldsymbol{\imath}) = 1 + \sum_{l=1}^{d} 2^{l-1}\boldsymbol{\imath}_l. \tag{1}$$

We can note that $\mathcal{L}(\boldsymbol{\imath}) - 1$ has for binary representation $(\boldsymbol{\imath}_d\boldsymbol{\imath}_{d-1}\cdots\boldsymbol{\imath}_1)_2$. Let $\widehat{\mathbf{q}}$ be the d-by-2^d array containing all the vertices of $\widehat{\mathrm{H}}$ and defined by

$$\widehat{\mathbf{q}}(:,j) \stackrel{\mathsf{def}}{=} \boldsymbol{x}^{\mathcal{L}^{-1}(j)}, \quad \forall j \in [\![1,2^d]\!].$$

where $\widehat{\mathbf{q}}(:,j)$ denotes the j-th column of the array $\widehat{\mathbf{q}}$.

For example, with $d = 3$, the array $\widehat{\mathbf{q}}$ is given by

$$\widehat{\mathbf{q}} \stackrel{\mathsf{def}}{=} \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

and it can be obtained from the more general function CartesianGridPoints introduced in section 2.2.1 and described in Appendix B by using command

$$\widehat{\mathbf{q}} \leftarrow \textsc{CartesianGridPoints}(\textsc{Ones}(1,d))$$

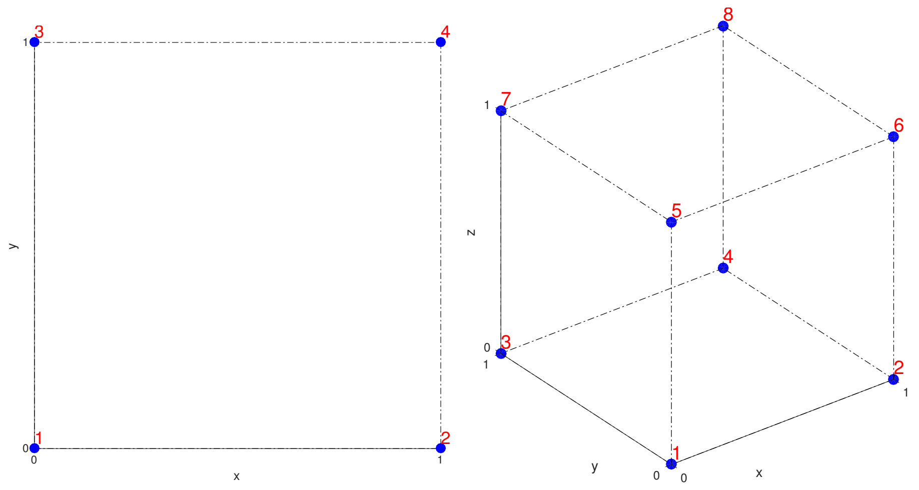


Figure 2: Vertices of the unit hypercube $[0, 1]^d$, $d = 2$ (left) and $d = 3$ (right) with their indices in the array $\widehat{\mathbf{q}}$

2.2 Cartesian grid

Definition 5. *A cartesian grid in $\mathbb{R}^d$ is a tessellation where the elements are unit d-hypercubes and the vertices are integer lattices.*

Let $\mathbf{N} = (N_1, \dots, N_d) \in (\mathbb{N}^*)^d$. We denote by $\mathcal{Q}_{\mathbf{N}}$ the cartesian grid of $[\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!]$. The cartesian grid $\mathcal{Q}_{\mathbf{N}}$ is composed with n_{q} grid points and n_{me} unit d-hypercubes where

$$n_{\mathrm{q}} = \prod_{l=1}^{d} (N_l + 1) \quad \text{and} \quad n_{\mathrm{me}} = \prod_{l=1}^{d} N_l. \tag{2}$$

The objective of this section is to describe the construction of the vertices (or points) array $\mathbf{q}$ (section 2.2.1) and the connectivity array $\mathbf{me}$ associated with this cartesian grid (section 2.2.2). More precisely,

- $\mathbf{q}(\nu, j)$ is the ν-th coordinate of the j-th vertex, $\nu \in \{1, \dots, d\}$, $j \in \{1, \dots, n_{\mathrm{q}}\}$. The j-th vertex will be also denoted by $\mathbf{q}^j = \mathbf{q}(:, j)$.
- $\mathbf{me}(\beta, k)$ is the storage index of the β-th vertex of the k-th element (unit hypercube), in the array q, for $\beta \in \{1, ..., 2^d\}$ and $k \in \{1, \dots, n_{\mathrm{me}}\}$. So $\mathbf{q}(:, \mathbf{me}(\beta, k))$ represents the coordinates of the β-th vertex of the k-th cartesian grid element.

We represent in Figure 3 two cartesian grids with the numbering of the n_{me} unit d-hypercubes. For example, on the left figure ($d = 2$), the 5-th unit hypercube is given by the vertices of numbers $6, 7, 10, 11$ and so $\mathbf{me}(:, 5) = (6, 7, 10, 11)$ and on the right figure ($d = 3$), for the 9-th hypercube, we have $\mathbf{me}(:, 9) = (16, 17, 19, 20, 28, 29, 31, 32)$.

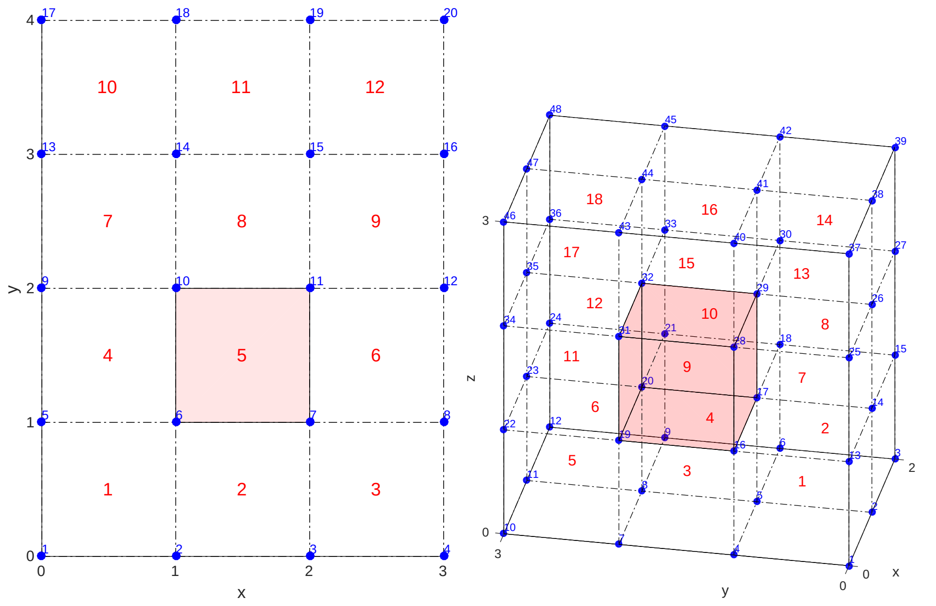


Figure 3: In blue, vertices of cartesian Grid in $\mathbb{R}^d$, $d = 2$ with $N_1 = 3$, $N_2 = 4$ (left) and $d = 3$ with $(N_1, N_2, N_3) = (2, 3, 3)$ (right). The red numbers are the indices of unit hypercubes in the array **me**

2.2.1 Points of the cartesian grid

The grid points may be identified by a d-tuple $\boldsymbol{\imath} = (i_1, i_2, \cdots, i_d) \in [\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!]$ and the corresponding grid point denoted by $\boldsymbol{x}^{\boldsymbol{\imath}}$, which has integer coordinates, is given by

$$\boldsymbol{x}^{\boldsymbol{\imath}} = \sum_{l=1}^{d} i_l \boldsymbol{e}^{[l]} = (i_1, i_2, \cdots, i_d)^{\mathtt{t}} = \boldsymbol{\imath} \tag{3}$$

where $\{\boldsymbol{e}^{[1]}, \ldots, \boldsymbol{e}^{[d]}\}$ denote the standard basis of $\mathbb{R}^d$.

We want to store all the grid points in a 2D-array $\mathbf{q}$ of size d-by-n_{q}. To define an order of storage in the array $\mathbf{q}$, we will use the bijective function $\mathcal{G}$ mapping the tuple points set $[\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!]$ in the global points index set $[\![1, n_{\mathrm{q}}]\!]$ defined by

$$\mathcal{G}(\boldsymbol{\imath}) = 1 + \sum_{l=1}^{d} i_l \beta_l = 1 + \langle \boldsymbol{\imath}, \boldsymbol{\beta} \rangle, \quad \forall \boldsymbol{\imath} \in [\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!] \tag{4}$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d) \in \mathbb{N}^d$ and

$$\beta_l = \prod_{j=1}^{l-1} (N_j + 1), \ \forall l \in [\![1, d]\!]. \tag{5}$$

From this mapping function, we define the vertex array $\mathbf{q}$ as

$$\mathbf{q}(:, \mathcal{G}(\boldsymbol{\imath})) = \boldsymbol{x}^{\boldsymbol{\imath}} = \boldsymbol{\imath}, \quad \forall \boldsymbol{\imath} \in [\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!] \tag{6}$$

According to the numbering choice $\mathcal{G}$, we give in Algorithm 1 the vectorized function CartesianGridPoints which returns the array $\mathbf{q}$. In Appendix B, we explain how this function was written

Algorithm 1 Function CartesianGridPoints : compute the d-by-n_q array $\mathbf{q}$ which contains all the points of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$ (vectorized version)

```
Input :
  $\boldsymbol{N}$  :  array of $d$ integers, $\boldsymbol{N}(i) = N_i$.

Output :
  $\mathbf{q}$  :  array of $d$-by-$n_q$ integers.

   Function $\mathbf{q} \leftarrow$ CartesianGridPoints ($\boldsymbol{N}$)
     $\boldsymbol{\beta} \leftarrow$ CGbeta($\boldsymbol{N}$)
     for $r \leftarrow 1$ to $d$ do
       $\boldsymbol{A} \leftarrow$ Reshape(RepTile([0 : $\boldsymbol{N}(r)$], $\beta_r$, 1), 1, ($\boldsymbol{N}(r) + 1)\beta_r$)
       $\mathbf{q}(r, :) \leftarrow$ RepTile($\boldsymbol{A}$, 1, Prod($\boldsymbol{N}(r + 1 : d) + 1$))
     end for
   end Function
```

From this array $\mathbf{q}$, we can now construct the tessellation by unit d-hypercubes of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$.

2.2.2 Connectivity array of a cartesian grid

The d-dimensional cartesian grid $\mathcal{Q}_{\mathbf{N}}$ contains n_{me} unit d-hypercubes having for vertices the cartesian grid points. All these unit hypercubes can be only identified by their vertex of minimal coordinates. Let $\boldsymbol{\imath} \in [\![0, N_1[\![\times \cdots \times [\![0, N_d[\![$.

We denote by $\mathrm{H}_{\boldsymbol{\imath}}$ the hypercube with $\boldsymbol{x}^{\boldsymbol{\imath}}$ as vertex of minimal coordinates. The 2^d vertices of $\mathrm{H}_{\boldsymbol{\imath}}$ are the vertices $\boldsymbol{x}^{\boldsymbol{\imath}+\boldsymbol{p}}$, $\forall \boldsymbol{p} \in [\![0, 1]\!]^d$.

We want to build the connectivity array $\mathbf{me}$ of dimensions 2^d-by-n_{me} such that $\mathbf{me}(l, k)$ is the index in array $\mathbf{q}$ of the l-th vertex of the k-th hypercube : this vertex is $\mathbf{q}(:, \mathbf{me}(l, k))$.

To define an order of storage in the array $\mathbf{me}$, we will use the bijective function $\mathcal{H}$ mapping the tuple points set $[\![0, N_1[\![\times \cdots \times [\![0, N_d[\![$ in the global points index set $[\![1, n_{\mathrm{me}}]\!]$ such that $k = \mathcal{H}(\boldsymbol{\imath})$ and defined by

$$\mathcal{H}(\boldsymbol{\imath}) = 1 + \sum_{l=1}^{d} i_l \prod_{j=1}^{l-1} N_j, \quad \boldsymbol{\imath} \in [\![0, N_1[\![\times \cdots \times [\![0, N_d[\![\tag{7}$$

The inverse function $\mathcal{H}^{-1}$ can be easily built. Indeed, if we define the d-by-n_{me} array $\mathbf{Hinv}$ by

$$\mathbf{Hinv} \leftarrow \text{CartesianGridPoints}(\boldsymbol{N} - 1).$$

then by construction we have

$$\mathcal{H}^{-1}(k) = \mathbf{Hinv}(:, k), \quad \forall k \in [\![1, n_{\mathrm{me}}]\!]$$

Let $k \in [\![1, n_{\mathrm{me}}]\!]$ and $\boldsymbol{\imath} = \mathcal{H}^{-1}(k)$. The k-th hypercube is $\mathrm{H}_{\boldsymbol{\imath}}$ and has for vertex of minimal coordinates $\boldsymbol{x}^{\boldsymbol{\imath}}$. By construction of array $\mathbf{q}$ we have

$$\boldsymbol{x}^{\boldsymbol{\imath}} = \mathbf{q}(:, \mathcal{G}(\boldsymbol{\imath}))$$

From vector $\boldsymbol{\beta}$ defined in (5), we have $\mathcal{G}(\boldsymbol{\imath}) = 1 + \langle \boldsymbol{\imath}, \boldsymbol{\beta} \rangle$. Using matricial operations we can define the 1-by-n_{me} array **iBase** by

$$\mathbf{iBase} \leftarrow \boldsymbol{\beta}^{\mathrm{t}} * \mathbf{Hinv} + 1$$

such that

$$\mathcal{G}(\boldsymbol{\imath}) = \mathcal{G} \circ \mathcal{H}^{-1}(k) = \mathbf{iBase}(k). \tag{8}$$

Let $\boldsymbol{\imath} \in [\![0, N_1[\![\times \cdots \times [\![0, N_d[\![$ and $k = \mathcal{H}(\boldsymbol{\imath})$ then we take

$$\mathbf{q}(:, \mathbf{me}(l,k)) = \boldsymbol{x}^{\boldsymbol{\imath}} + \widehat{\mathbf{q}}(:,l) = \boldsymbol{x}^{\boldsymbol{\imath} + \widehat{\mathbf{q}}(:,l)}$$

where $\widehat{\mathbf{q}}$ is defined in section 2.1. So we obtain

$$\mathbf{me}(l,k) = \mathcal{G}(\boldsymbol{\imath} + \widehat{\mathbf{q}}(:,l)) \tag{9}$$

Lemma 6. *Let $\boldsymbol{\imath} \in \mathcal{Q}_{\mathbf{N}}$ and $\boldsymbol{p} \in \mathbb{Z}^d$, such that $\boldsymbol{\imath} + \boldsymbol{p} \in \mathcal{Q}_{\mathbf{N}}$. We have*

$$\mathcal{G}(\boldsymbol{\imath} + \boldsymbol{p}) = \mathcal{G}(\boldsymbol{\imath}) + \langle \boldsymbol{p}, \boldsymbol{\beta} \rangle \tag{10}$$

where $\boldsymbol{\beta}$ is defined in (5).

Proof. We have

$$\begin{aligned}
\mathcal{G}(\boldsymbol{\imath} + \boldsymbol{p}) &\overset{\text{def}}{=} 1 + \sum_{s=1}^{d} (i_s + p_l) \prod_{j=1}^{s-1} (N_j + 1) \\
&= 1 + \sum_{s=1}^{d} i_s \prod_{j=1}^{s-1} (N_j + 1) + \sum_{s=1}^{d} p_s \prod_{j=1}^{s-1} (N_j + 1) \\
&= \mathcal{G}(\boldsymbol{\imath}) + \sum_{s=1}^{d} p_s \beta_s.
\end{aligned}$$

$\square$

From Lemma 6 and definition of $\boldsymbol{\beta}$ in (5), we obtain

$$\mathcal{G}(\boldsymbol{\imath} + \widehat{\mathbf{q}}(:,l)) = \mathcal{G}(\boldsymbol{\imath}) + \sum_{s=1}^{d} \widehat{\mathbf{q}}(s,l) \beta_s.$$

From (9), we have, for all $l \in [\![1, d]\!]$,

$$\mathbf{me}(l,k) \leftarrow \mathbf{iBase}(k) + \langle \widehat{\mathbf{q}}(:,l), \boldsymbol{\beta} \rangle$$

or in a vectorized form

$$\mathbf{me}(l,:) \leftarrow \mathbf{iBase} + \langle \widehat{\mathbf{q}}(:,l), \boldsymbol{\beta} \rangle$$

We can now easily write the function CGTessHyp in Algorithm 2 which computes the arrays **q** and **me**.

Algorithm 2 Function CGTessHyp :

Input :

$\boldsymbol{N}$: array of d integers, $\boldsymbol{N}(i) = N_i$.

Output :

$\mathbf{q}$: array of d-by-n_{q} array of integers.

$\mathbf{me}$: 2-dimensional connectivity array of sizes 2^d-by-N_h. $\mathbf{me}(l,k)$ is the index in array $\mathbf{q}$ of the l-th vertex of the k-th hypercube : this vertex is $\mathbf{q}(:,\mathbf{me}(l,k))$.

```
Function $[\mathbf{q},\mathbf{me}] \leftarrow$ CGTessHyp $(\boldsymbol{N})$
  $\mathbf{q} \leftarrow$ CartesianGridPoints$(\boldsymbol{N})$
  $\mathbf{Hinv} \leftarrow$ CartesianGridPoints$(\boldsymbol{N}-1)$
  $\widehat{\mathbf{q}} \leftarrow$ CartesianGridPoints(Ones$(1,d)$)
  $\boldsymbol{\beta} \leftarrow$ CGbeta$(\boldsymbol{N})$
  $\mathbf{iBase} \leftarrow \boldsymbol{\beta}^{\mathrm{t}} * \mathbf{Hinv} + 1$
  for $l \leftarrow 1$ to $2^d$ do
    $\mathbf{me}(l,:) \leftarrow \mathbf{iBase} + \langle \boldsymbol{\beta}, \widehat{\mathbf{q}}(:,l) \rangle$
  end for
end Function
```

Now we can generalize to the tessellation by d-orthotopes of a d-orthotope.

2.3 Tessellation of a d-orthotope by d-orthotopes

Let $\mathcal{O}_d$ be the d-orthotope $[a_1, b_1] \times \cdots \times [a_d, b_d]$. To construct a regular grid on $\mathcal{O}_d$ with N_i+1 points in $\boldsymbol{e}^{[i]}$ direction, $i \in [\![1,d]\!]$, we use an affine transformation of the cartesian grid $\mathcal{Q}_{\mathbf{N}} = [\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!]$ to $\mathcal{O}_d$. Let $\boldsymbol{a} = (a_1, \ldots, a_d)$, $\boldsymbol{b} = (b_1, \ldots, b_d)$ and $\boldsymbol{h} = (h_1, \ldots, h_d)$ with $h_i = (b_i - a_i)/N_i$ be three vectors of $\mathbb{R}^d$. Let $\mathbb{H} \in \mathcal{M}_d(\mathbb{R})$ be the diagonal matrix with $\boldsymbol{h}$ as diagonal. Then the affine transformation is given by

$$\begin{array}{rccc} \mathcal{A} & : & \mathcal{Q}_{\mathbf{N}} & \longrightarrow & \mathcal{O}_d \\ & & \boldsymbol{x} & \longmapsto & \boldsymbol{y} = \boldsymbol{a} + \mathbb{H}\boldsymbol{x} \end{array}$$

Let $\boldsymbol{N} \leftarrow [N_1, \ldots, N_d]$. The tessellation of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$ is obtained by

$$[\mathbf{q}, \mathbf{me}] \leftarrow \textsc{CGTessHyp}(\boldsymbol{N})$$

To obtain the tessellation of the orthotope $\mathcal{O}_d$ we only have to apply the affine transformation $\mathcal{A}$ to array $\mathbf{q}$. In a vectorized form, one can write for all $i \in [\![1,d]\!]$

$$\mathbf{q}(i,:) \leftarrow \boldsymbol{a}(i) + (\boldsymbol{b}(i) - \boldsymbol{a}(i))/\boldsymbol{N}(i) * \mathbf{q}(i,:)$$

This operation is done by the function boxMapping given in Algorithm 3.

Algorithm 3 Function boxMapping : mapping unit d-hypercube $[0,1]^d$ to the d-orthotope $[a_1, b_1] \times \cdots \times [a_d, b_d]$

Input :
- $\mathbf{q}$: d-by-n array of reals in $[0,1]^d$
- $\boldsymbol{a}, \boldsymbol{b}$: arrays of d reals, $\boldsymbol{a}(i) = a_i$, $\boldsymbol{b}(i) = b_i$ with $a_i < b_i$

Output :
- $\mathbf{q}$: array of d-by-n array of reals in the orthotope.

```
Function $\mathbf{q} \leftarrow$ boxMapping ($\mathbf{q}, \boldsymbol{a}, \boldsymbol{b}$)
  for $i \leftarrow 1$ to $d$ do
    $h \leftarrow (\boldsymbol{b}(i) - \boldsymbol{a}(i))/\boldsymbol{N}(i)$
    $\mathbf{q}(i,:) \leftarrow \boldsymbol{a}(i) + h * \mathbf{q}(i,:)$
  end for
end Function
```

We write in Algorithm 4, the function OrthTessOrth which returns the arrays $\mathbf{q}$ and $\mathbf{me}$ corresponding to the regular tessellation of $\mathcal{O}_d$ by d-orthotopes.

Algorithm 4 Function OrthTessOrth : d-orthotope regular tessellation by orthotopes

Input :
- $\boldsymbol{N}$: array of d integers, $\boldsymbol{N}(i) = N_i$.
- $\boldsymbol{a}, \boldsymbol{b}$: arrays of d reals, $\boldsymbol{a}(i) = a_i$, $\boldsymbol{b}(i) = b_i$ with $a_i < b_i$

Output :
- $\mathbf{q}$: array of d-by-n_{q} reals.
- $\mathbf{me}$: array of 2^d-by-n_{me} integers.

```
Function $[\mathbf{q}, \mathbf{me}] \leftarrow$ OrthTessOrth ($\boldsymbol{N}, \boldsymbol{a}, \boldsymbol{b}$)
  $[\mathbf{q}, \mathbf{me}] \leftarrow$ CGTessHyp($\boldsymbol{N}$)
  $\mathbf{q} \leftarrow$ boxMapping($\mathbf{q}, \boldsymbol{a}, \boldsymbol{b}$)
end Function
```

2.4 Numbering of the m-faces of the unit d-hypercube

Let $m \in [\![0, d]\!]$. As introduced in section 1, the m-faces of the unit d-hypercube $[0,1]^d$ are unit m-hypercubes in $\mathbb{R}^d$ where $d-m$ dimensions are reduced to the singleton $\{0\}$ or $\{1\}$.

We have $n_c = \begin{pmatrix} d \\ m \end{pmatrix}$ possible choices to select the index of the $d-m$ reduced dimensions (combination of d elements taken $d-m$ at a time) and for each selected dimension 2 choices : $\{0\}$ or $\{1\}$.

So if $l \in [\![1, d]\!]$ is the index of a reduced dimension then vertices $\boldsymbol{x}^{\boldsymbol{\imath}} (= \boldsymbol{\imath} = (i_1, \ldots, i_d))$ is such that $i_l = 0$ (minimum value) or $i_l = 1$ (maximum value).

Let $\mathbb{L}^{[d,m]}$ be the set of all the combinations of $[\![1, d]\!]$ taken $d-m$ at a time :

$$\mathbb{L}^{[d,m]} \leftarrow \text{Combs}([\![1, d]\!], d-m).$$

Then the length of array $\mathbb{L}^{[d,m]}$ is n_c-by-$(d-m)$. Each row of $\mathbb{L}^{[d,m]}$ contains the index of the $d-m$ reduced dimensions of an m-face.

Let $\mathbb{S}^{[d-m]}$ be the $(d-m)$-by-2^{d-m} array containing all the possible choices of the constants for the $d-m$ reduced dimensions (2 choices per dimension) : values are 0 for constant minimal value or 1 for maximal value. This array can be built by using function CartesianGridPoints defined in Algorithm 1 and we have

$$\mathbb{S}^{[d-m]} \leftarrow \textsc{CartesianGridPoints}(\textsc{Ones}(1, d-m))$$

Let $l \in [\![1, n_c]\!]$, $r \in [\![1, 2^{d-m}]\!]$ and $k = 2^{d-m}(l-1) + r$. We define the k-th m-faces of the unit d-hypercube as

$$\left\{\boldsymbol{x} \in [0,1]^d \text{ such that } \boldsymbol{x}(\mathbb{L}^{[d,m]}(l,s)) = \mathbb{S}^{[d-m]}(s,r),\ \forall s \in [\![1, d-m]\!]\right\}$$

or in a vectorized form

$$\left\{\boldsymbol{x} \in [0,1]^d \text{ such that } \boldsymbol{x}(\mathbb{L}^{[d,m]}(l,:)) = \mathbb{S}^{[d-m]}(:,r)^{\mathsf{t}}\right\} \tag{11}$$

For example, to obtain the ordered 2-faces of the unit 3-hypercube we compute

$$\mathbb{L}^{[3,2]} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \text{and} \quad \mathbb{S}^{[1]} = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

and then we have

2-face number	**Set**
1	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0\}$
2	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1\}$
3	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_2 = 0\}$
4	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_2 = 1\}$
5	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_3 = 0\}$
6	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_3 = 1\}$

To obtain the ordered 1-faces of the unit 3-hypercube we compute

$$\mathbb{L}^{[3,1]} = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 2 & 3 \end{pmatrix} \quad \text{and} \quad \mathbb{S}^{[2]} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

and then we have

1-face number	**Set**
1	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_2 = 0\}$
2	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_2 = 0\}$
3	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_2 = 1\}$
4	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_2 = 1\}$
5	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_3 = 0\}$
6	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_3 = 0\}$
7	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_3 = 1\}$
8	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_3 = 1\}$
9	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_2 = 0,\ \boldsymbol{x}_3 = 0\}$
10	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_2 = 1,\ \boldsymbol{x}_3 = 0\}$
11	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_2 = 0,\ \boldsymbol{x}_3 = 1\}$
12	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_2 = 1,\ \boldsymbol{x}_3 = 1\}$

To obtain the ordered 0-faces of the unit 3-hypercube we compute

$$\mathbb{L}^{[3,0]} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \quad \text{and} \quad \mathbb{S}^{[3]} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

and then we have

1-face number	Set
1	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_2 = 0,\ \boldsymbol{x}_3 = 0\}$
2	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_2 = 0,\ \boldsymbol{x}_3 = 0\}$
3	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_2 = 1,\ \boldsymbol{x}_3 = 0\}$
4	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_2 = 1,\ \boldsymbol{x}_3 = 0\}$
5	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_2 = 0,\ \boldsymbol{x}_3 = 1\}$
6	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_2 = 0,\ \boldsymbol{x}_3 = 1\}$
7	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 0,\ \boldsymbol{x}_2 = 1,\ \boldsymbol{x}_3 = 1\}$
8	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } \boldsymbol{x}_1 = 1,\ \boldsymbol{x}_2 = 1,\ \boldsymbol{x}_3 = 1\}$

2.5 m-faces tessellations of a cartesian grid

Let $\mathcal{Q}_{\mathbf{N}}$ be the d-dimensional cartesian grid defined in section 2.2. Not to confuse the notations, we denote by $\mathcal{Q}_{\mathbf{N}}.\mathbf{q}$ and $\mathcal{Q}_{\mathbf{N}}.\mathbf{me}$ respectively the vertices and connectivity arrays of the tessellation by unit hypercubes of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$.

Let $m \in [\![0, d[\![$ and $k \in [\![1, E_{m,d}]\!]$. We want to determine $\mathcal{Q}^m_{\mathbf{N}}(k)$ the tessellation obtained from the restriction of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$ to its k-th m-face where the numbering of the m-faces is specified in section 2.4. We denote by

- $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}$, the (local) vertex array
- $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{me}$, the (local) connectivity array
- $\mathcal{Q}^m_{\mathbf{N}}(k)$.toGlobal, the global indices such that

$$\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q} \equiv \mathcal{Q}_{\mathbf{N}}.\mathbf{q}(:, \mathcal{Q}^m_{\mathbf{N}}(k).\text{toGlobal}).$$

By construction, $\mathcal{Q}^m_{\mathbf{N}}(k)$ is the tessellation by unit m-hypercubes of an m-hypercube in $\mathbb{R}^d$.

Let $l \in [\![1, n_c]\!]$, $r \in [\![1, 2^{d-m}]\!]$ and $k = 2^{d-m}(l-1)+r$. The cartesian grid point $\boldsymbol{x} = (x_1, \dots, x_d)^{\mathsf{t}}$ is on the k-th m-face $\mathcal{Q}^m_{\mathbf{N}}(k)$ if and only if for all $s \in [\![1, d-m]\!]$ and with $j = \mathbb{L}(l, s)$ we have

$$x_j = N_j \times \mathbb{S}^{[d-m]}(s, r) = \begin{cases} 0 & \text{if } \mathbb{S}^{[d-m]}(s,r) == 0, \quad \text{(minimum value)} \\ N_j & \text{if } \mathbb{S}^{[d-m]}(s,r) == 1, \quad \text{(maximum value)} \end{cases}$$

or in a vectorized form using element-wise multiplication operator $\boldsymbol{.*}$:

$$\boldsymbol{x}(\mathbb{L}(l, :)) = \boldsymbol{N}(\mathbb{L}(l, :)) \boldsymbol{.*} \mathbb{S}^{[d-m]}(:, r)^{\mathsf{t}}. \tag{12}$$

Let $l \in [\![1, n_c]\!]$, $r \in [\![1, 2^{d-m}]\!]$ and $k = 2^{d-m}(l-1) + r$. We define the k-th m-faces of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$ as

$$\left\{\boldsymbol{x} \in \mathcal{Q}_{\mathbf{N}} \text{ such that } \boldsymbol{x}(\mathbb{L}^{[d,m]}(l, :)) = \boldsymbol{N}(\mathbb{L}(l, :)) \boldsymbol{.*} \mathbb{S}^{[d-m]}(:, r)^{\mathsf{t}}\right\} \tag{13}$$

2.5.1 Case $m = 0$.

If $m = 0$, the m-faces are the 2^d corner points of the cartesian grid. We also have $\mathbb{L}^{[d,0]} = 1 : d$ and $\mathbb{S}^{[d]}$ a d-by-2^d array

From (13), we obtain that $\forall k \in [\![1, 2^d]\!]$ the k-th 0-face of $\mathcal{Q}_{\mathbf{N}}$ is reduced to the point

$$\boldsymbol{x} = \boldsymbol{N} \,\textbf{.*}\, \mathbb{S}^{[d]}(:, k)^{\mathsf{t}}$$

and it is also the k-th column of the array Q of dimensions d-by-2^d given by

$$Q \leftarrow \begin{pmatrix} N_1 & 0 & \ldots & \ldots & 0 \\ 0 & N_2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & \ldots & 0 & N_d \end{pmatrix} \mathbb{S}^{[d]}$$

So we obtain

$$\mathcal{Q}^0_{\mathbf{N}}(k).\mathbf{q} = Q(:, k)$$
$$\mathcal{Q}^0_{\mathbf{N}}(k).\mathbf{me} = 1$$
$$\mathcal{Q}^0_{\mathbf{N}}(k).\text{toGlobal} = \langle \boldsymbol{\beta}, Q(:, k) \rangle + 1$$

2.5.2 Case $m > 0$

Let $l \in [\![1, n_c]\!]$, $r \in [\![1, 2^{d-m}]\!]$ and $k = 2^{d-m}(l-1) + r$. To construct $\mathcal{Q}^m_{\mathbf{N}}(k)$ we first set a tessellation without constant dimensions $\mathbf{idc} = \mathbb{L}(l, :)$ (i.e. only with nonconstant dimensions $\mathbf{idnc} = [\![1, d]\!] \backslash \mathbf{idc}$):

$$[\mathbf{q}^w, \mathbf{me}^w] \leftarrow \textsc{CGTessHyp}(\boldsymbol{N}(\mathbf{idnc}))$$

The dimension of the array $\mathbf{q}^w$ is d-by-n^l_{q} where $n^l_{\mathrm{q}} = \prod_{i \in \mathbf{idnc}} (N_i + 1)$. Then the nonconstant rows are

$$\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}(\mathbf{idnc}(i), :) \leftarrow \mathbf{q}^w(i, :), \quad \forall i \in [\![1, m]\!]$$

and the constants rows

$$\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}(\mathbf{idc}(i), :) \leftarrow \boldsymbol{N}(\mathbf{idc}(i)) * \mathbb{S}^{[d-m]}(i, r) * \textsc{Ones}(1, n^l_{\mathrm{q}}), \quad \forall i \in [\![1, d-m]\!]$$

In a vectorized way, we have

$$\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}(\mathbf{idnc}, :) \leftarrow \mathbf{q}^w$$
$$\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}(\mathbf{idc}, :) \leftarrow \left(\boldsymbol{N}(\mathbf{idc})^{\mathsf{t}} \,\textbf{.*}\, \mathbb{S}^{[d-m]}(:, r)\right) * \textsc{Ones}(1, n^l_{\mathrm{q}})$$

We immediately have the connectivity array

$$\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{me} = \mathbf{me}^w.$$

It remains us to compute $\mathcal{Q}^m_{\mathbf{N}}(k)$.toGlobal. For that we use the mapping function $\mathcal{G}$ defined in section 2.2.1 and more particularly (6). Indeed, for all $j \in [\![1, n^l_{\mathrm{q}}]\!]$, we can identify the point $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}(:, j)$ by the d-tuple $\boldsymbol{\imath}$ and use it with the mapping function $\mathcal{G}$ to obtain the index in array $\mathcal{Q}_{\mathbf{N}}.\mathbf{q}$ of the point $\mathcal{Q}_{\mathbf{N}}^{m}(k).\mathbf{q}(:,j)$. So we have

$$\boldsymbol{\imath} = \mathcal{Q}_{\mathbf{N}}^{m}(k).\mathbf{q}(:,j) = \mathcal{Q}_{\mathbf{N}}.\mathbf{q}(:,\mathcal{G}(\boldsymbol{\imath}))$$

and then

$$\mathcal{Q}_{\mathbf{N}}^{m}(k).\text{toGlobal}(j) = \mathcal{G}(\mathcal{Q}_{\mathbf{N}}^{m}(k).\mathbf{q}(:,j)).$$

In a vectorized way, we set

$$\mathcal{Q}_{\mathbf{N}}^{m}(k).\text{toGlobal} \leftarrow 1 + \boldsymbol{\beta}^{\mathtt{t}} * \mathcal{Q}_{\mathbf{N}}^{m}(k).\mathbf{q}$$

with the vector $\boldsymbol{\beta}$ defined in (5).

One can also compute the connectivity array of $\mathcal{Q}_{\mathbf{N}}^{m}(k)$ associated with global vertices array $\mathcal{Q}_{\mathbf{N}}.\mathbf{q}$ which is given by $\mathcal{Q}_{\mathbf{N}}^{m}(k).\text{toGlobal}(\mathbf{me}^{w})$.

We give in Algorithm 5 the function CGTessFaces which computes $\mathcal{Q}_{\mathbf{N}}^{m}(k)$, $\forall k \in [\![1, 2^{d-m}n_c]\!]$.

Algorithm 5 Function CGTessFaces : compute all the m-face tessellations by m-hypercubes of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$.

```
Input :
  $\boldsymbol{N}$  :  1-by-$d$ array of integers, $\boldsymbol{N}(i) = N_i$.
  $m$  :  integer, $0 \leqslant m < d$
Output :
  $\mathcal{Q}^m_{\mathbf{N}}$  :  array of the tessellations of each $m$-faces of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$.
        Its length is $E_{m,d} = 2^{d-m}\binom{d}{m}$.

  Function $s\mathcal{Q}_{\mathbf{N}} \leftarrow$ CGTessFaces $(\boldsymbol{N}, m)$
    $\boldsymbol{\beta} \leftarrow$ CGbeta$(\boldsymbol{N})$
    if $m == 0$ then
      $\mathbb{Q} \leftarrow$ Diag$(\boldsymbol{N}) *$ CartesianGridPoints(Ones$(1, d)$)
      for $k \leftarrow 1$ to $2^d$ do
        $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q} \leftarrow \mathbb{Q}(:, k)$
        $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{me} \leftarrow 1$
        $\mathcal{Q}^m_{\mathbf{N}}(k).\mathrm{toGlobal} \leftarrow 1 + \langle \boldsymbol{\beta}, \mathbb{Q}(:, k)\rangle$
      end for
    else
      $n_c \leftarrow \binom{d}{m}$
      $\mathbb{L} \leftarrow$ Combs$([\![1, d]\!], d - m)$
      $\mathbb{S} \leftarrow$ CartesianGridPoints(Ones$(1, d - m)$)
      $k \leftarrow 1$
      for $l \leftarrow 1$ to $n_c$ do
        $\mathbf{idc} \leftarrow \mathbb{L}(l, :)$
        $\mathbf{idnc} \leftarrow [\![1, d]\!] \backslash \mathbf{idc}$
        $[\mathbf{q}^w, \mathbf{me}^w] \leftarrow$ CGTessHyp$(\boldsymbol{N}(\mathbf{idnc}))$
        $n^l_{\mathrm{q}} \leftarrow \prod_{s=1}^m (\boldsymbol{N}(\mathbf{idnc}(s)) + 1)$                  ▷ or length of $\mathbf{q}^w$
        for $r \leftarrow 1$ to $2^{d-m}$ do
          $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}(\mathbf{idnc}, :) \leftarrow \mathbf{q}^w$
          $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}(\mathbf{idc}, :) \leftarrow \left(\boldsymbol{N}(\mathbf{idc})^{\mathsf{t}} \,.\!*\, \mathbb{S}(:, r)\right) *$ Ones$(1, n^l_{\mathrm{q}})$
          $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{me} \leftarrow \mathbf{me}^w$
          $\mathcal{Q}^m_{\mathbf{N}}(k).\mathrm{toGlobal} \leftarrow 1 + \boldsymbol{\beta}^{\mathsf{t}} * \mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}$
          $k \leftarrow k + 1$
        end for
      end for
    end if
  end Function
```

2.6 m-faces tessellations of a d-orthotope

As seen in section 2.3, we only have to apply the function boxMapping to each array $\mathcal{Q}^m_{\mathbf{N}}(k).\mathbf{q}$ of the tessellations of the m-faces of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$. This is the object of the function OrthTessFaces given in Algorithm 6.

Algorithm 6 Function ORTHTESSFACES : compute the conforming tessellations of all the m-faces of the d-orthotope $[a_1, b_1] \times \cdots \times [a_d, b_d]$

```
Input :
  N      : array of d integers, N(i) = N_i.
  a, b   : arrays of d reals, a(i) = a_i, b(i) = b_i
  m      : integer, 0 ⩽ m < d
Output :
  sO_h   : array of the tessellations of each m-faces of the orthotope.
           Its length is $E_{m,d} = 2^{d-m}\binom{d}{m}$.

  Function $s\mathcal{O}_{\boldsymbol{h}} \leftarrow$ OrthTessFaces $(\boldsymbol{N}, \boldsymbol{a}, \boldsymbol{b}, m)$
    $s\mathcal{O}_{\boldsymbol{h}} \leftarrow$ CGTessFaces$(\boldsymbol{N}, m)$
    for $k \leftarrow 1$ to len$(s\mathcal{O}_{\boldsymbol{h}})$ do
      $s\mathcal{O}_{\boldsymbol{h}}(k).\mathbf{q} \leftarrow$ boxMapping$(s\mathcal{O}_{\boldsymbol{h}}(k).\mathbf{q}, \boldsymbol{a}, \boldsymbol{b})$
    end for
  end Function
```

3 Tessellation by d-simplices

The goal of this section is to obtain a *conforming* triangulation (or tessellation) of a d-orthotope named $\mathcal{O}_d$ by d-simplices.

The basic principle selected here is to start from a regular grid of $\mathcal{O}_d$ obtained via a d-orthotope tessellation. on each d-polytope of the regular grid to obtain a conforming triangulation of $\mathcal{O}_d$. Then, using affine transformations, one can use the Kuhn's d-simplex decomposition on each d-polytope of the regular grid to obtain a conforming triangulation of $\mathcal{O}_d$.

3.1 Kuhn's decomposition of a d-hypercube

Kuhn's subdivision (see [1, 4, 5]) is a good way to divide a d-hypercube into d-simplices ($d \geqslant 2$). We recall that a d-simplex is made of $(d+1)$ vertices.

Definition 7. *Let* $\mathrm{H} = [0,1]^d$ *be the unit* d*-hypercube in* $\mathbb{R}^d$. *Let* $\mathbf{e}^{[1]}, \ldots, \mathbf{e}^{[d]}$ *be the standard unit basis vectors of* $\mathbb{R}^d$ *and denote by* S_d *the group of permutation of* $[\![1, d]\!]$. *For all* $\pi \in S_d$, *the simplex* K_π *has for vertices* $\{\boldsymbol{x}_\pi^{[0]}, \ldots, \boldsymbol{x}_\pi^{[d]}\}$ *defined by*

$$\boldsymbol{x}_\pi^{[0]} = (0, \ldots, 0)^t, \quad \boldsymbol{x}_\pi^{[j]} = \boldsymbol{x}_\pi^{[j-1]} + \mathbf{e}^{[\pi(j)]}, \ \forall j \in [\![1, d]\!]. \tag{14}$$

The set $\mathcal{K}(\mathrm{H})$ *defined by*

$$\mathcal{K}(\mathrm{H}) = \{K_\pi \mid \pi \in S_d\} \tag{15}$$

is called the ***Kuhn's subdivision*** *of* H *and its cardinality is* $d!$.

As sample, we give in Figure 4 the Kuhn'subdivision of an d-hypercube with $d = 2$ and $d = 3$. We choose the **positive orientation** for all the d simplices. The corresponding vertex array $\mathbf{q}$ and the connectivity array **me** are given by (préciser comment **me** est ordonné):

- for $d = 2$,

$$\mathbf{q} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \quad \mathbf{me} = \begin{pmatrix} 4 & 1 \\ 3 & 2 \\ 1 & 4 \end{pmatrix}$$

- for $d = 3$,

$$\mathbf{q} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{me} = \begin{pmatrix} 1 & 8 & 8 & 1 & 1 & 8 \\ 5 & 3 & 5 & 3 & 2 & 2 \\ 7 & 7 & 6 & 4 & 6 & 4 \\ 8 & 1 & 1 & 8 & 8 & 1 \end{pmatrix}$$

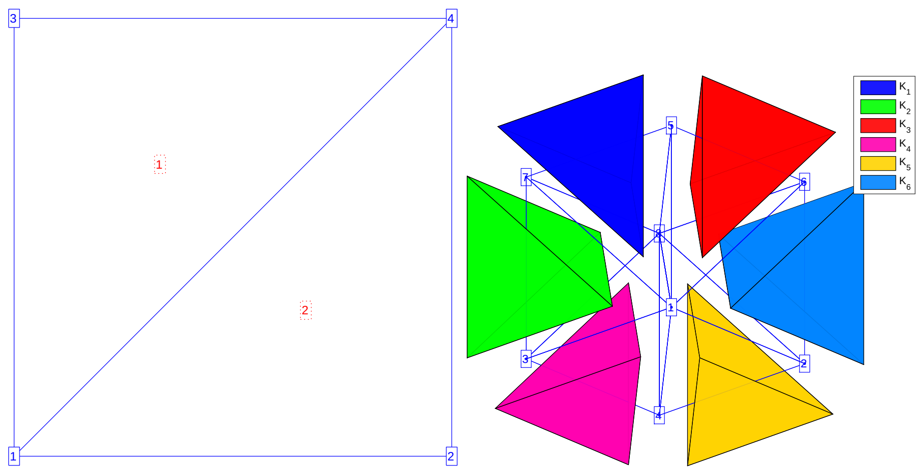


Figure 4: Kuhn's subdivision

Let K_{ref} be the *base simplex* or *reference simplex* with vertices denoted by $\{\boldsymbol{x}^{[0]}, \dots, \boldsymbol{x}^{[d]}\}$ and such that

$$\boldsymbol{x}^{[0]} = (0, \dots, 0)^t, \quad \boldsymbol{x}^{[j]} = \boldsymbol{x}^{[j-1]} + \boldsymbol{e}^{[j]}, \ \forall j \in [\![1, d]\!]. \tag{16}$$

Let $\pi \in S_n$ and $\pi(\boldsymbol{x})$ indicate the application of permutation π to the coordinates of vertex $\boldsymbol{x}$. The vertices of the simplex K_π defined in (14) can be derived from the reference simplex K_{ref} by

$$\boldsymbol{x}_\pi^{[j]} = \pi(\boldsymbol{x}^{[j]}), \quad \forall j \in [\![0, d]\!]. \tag{17}$$

Let $\pi(K_{\text{ref}})$ denote the application of permutation to each vertex of K_{ref}. Then we have

$$\pi(K_{\text{ref}}) = K_\pi \tag{18}$$

Lemma 8 ([1], Lemma 4.1)**.** *The **Kuhn's subdivision** $\mathcal{K}(\mathrm{H})$ of the unit d-hypercube* H *has the following properties:*

1. 0^d *and* 1^d *are common vertices of all elements* $K_\pi \in \mathcal{K}(\mathrm{H})$.
2. $\mathcal{K}(\mathrm{H})$ *is a consistent/conforming triangulation of* H.

3. *$\mathcal{K}(\mathrm{H})$ is compatible with translation, i.e., for each vector $\boldsymbol{v} \in [\![0,1]\!]^d$ the union of $\mathcal{K}(\mathrm{H})$ and $\mathcal{K}(\boldsymbol{v}+\mathrm{H})$ is a consistent/conforming triangulation of the set $\mathrm{H} \cup (\boldsymbol{v}+\mathrm{H})$.*

4. *For any affine transformation $\mathcal{F}$, the Kuhn's triangulation of $\mathcal{F}(\mathrm{H})$ is defined by $\mathcal{K}(\mathcal{F}(\mathrm{H})) \overset{\mathsf{def}}{=} \mathcal{F}(\mathcal{K}(\mathrm{H}))$.*

To explicitly obtain a Kuhn's triangulation $\mathcal{K}(\mathrm{H})$ of the unit d-hypercube H we must build the connectivity array, denoted by **me**, associated with the vertex array **q**. The dimension of the array **me** is $(d+1)$-by-$d!$.

Let $\mathbf{q}^{\mathrm{ref}}$ be the d-by-$(d+1)$ array of vertex coordinates of reference d-simplex K^{ref} :

$$\mathbf{q}^{\mathrm{ref}} = \left(\begin{array}{c:c:c:c} \boldsymbol{x}^{[0]} & \boldsymbol{x}^{[1]} & \dots \quad \dots & \boldsymbol{x}^{[d]} \end{array} \right) = \left(\begin{array}{c:cccc} 0 & 1 & \dots & \dots & 1 \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{array} \right)$$

Let $\mathbf{P}$ be the d-by-$d!$ array of all permutations of the set $[\![1,d]\!]$ and $\pi = \mathbf{P}(:,k)$ the k-th permutation. We use (17) and (18) to build the vertices of K_π. So the j-th vertices of K_π is given by

$$\boldsymbol{x}_\pi^{[j-1]} \leftarrow \mathbf{q}^{\mathrm{ref}}(\mathbf{P}(:,k),j)$$

To find which column in array **q** corresponding to $\boldsymbol{x}_\pi^{[j-1]}$ we use the mapping function $\mathcal{L}$ defined in (1) and we set

$$\mathbf{me}(j,k) \leftarrow \mathcal{L}(\mathbf{q}^{\mathrm{ref}}(P(:,k),j)) = \left\langle \begin{pmatrix} 2^0 \\ \vdots \\ 2^{d-1} \end{pmatrix}, \mathbf{q}^{\mathrm{ref}}(\mathbf{P}(:,k),j)) \right\rangle + 1$$

If the k-th d-simplex have a negative orientation, one can permute the index of the first and the last points to obtain a positive orientation:

$$\mathbf{me}(1,k) \leftrightarrow \mathbf{me}(d+1,k).$$

In Algorithm 7, we give the function KuhnTriangulation which returns the points array **q** and the connectivity array **me** where all the d-simplices have a positive orientation.

Algorithm 7 Kuhn's triangulation of the unit d-hypercube $[0,1]^d$ with $d!$ simplices (positive orientation)

```
1: Function $[\mathbf{q},\mathbf{me}] \leftarrow$ KuhnTriangulation $(d)$
2:   $\mathbf{q} \leftarrow$ CartesianGridPoints(Ones$(1,d)$)
3:   $\mathbf{q}^{\mathrm{ref}} \leftarrow \left(\begin{array}{c|cccc} 0 & 1 & \dots & \dots & 1 \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{array}\right)$    ▷ a $d$-by-$(d+1)$ array
4:   $\mathbf{P} \leftarrow$ perms$(1:d)$    ▷ perms$(V)$ generate all permutations of $V$. One column per permutation.
5:   $\mathbf{me} \leftarrow \mathbb{0}_{d+1,d!}$
6:   $\boldsymbol{a} \leftarrow [2^0, 2^1, \dots, 2^{d-2}, 2^{d-1}]$
7:   for $k \leftarrow 1$ to $d!$ do
8:     for $j \leftarrow 1$ to $d+1$ do
9:       $\mathbf{me}(j,k) \leftarrow$ dot$(\boldsymbol{a}, \mathbf{q}^{\mathrm{ref}}(\mathbf{P}(:,k),j)) + 1$
10:    end for
11:    if det$([q(:,me(:,k)); $ones$(1,d+1)]) < 0$ then
12:      $t \leftarrow \mathbf{me}(1,k)$, $\mathbf{me}(1,k) \leftarrow \mathbf{me}(d+1,k)$, $\mathbf{me}(d+1,k) \leftarrow t$
13:    end if
14:  end for
15: end Function
```

3.2 Cartesian grid triangulation

Let $\mathcal{Q}_{\mathbf{N}}$ be the d-dimensional cartesian grid defined in section 2.2. Not to confuse the notations, we denote by $\mathcal{Q}_{\mathbf{N}}.\mathbf{q}$ and $\mathcal{Q}_{\mathbf{N}}.\mathbf{me}$ respectively the vertices and connectivity arrays of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$. There are $N_h = \prod_{i=1}^{d} N_i$ unit hypercubes in this tessellation.

Let $\mathcal{I} = [\![0, N_1[\![\times \dots \times [\![0, N_d[\![$. We have

$$\mathcal{Q}_{\mathbf{N}} = \bigcup_{\boldsymbol{\imath}\in\mathcal{I}} \mathrm{H}_{\boldsymbol{\imath}}$$

where $\mathrm{H}_{\boldsymbol{\imath}}$ is the unit hypercube with $\boldsymbol{x}^{\boldsymbol{\imath}}$ as vertex of minimal coordinates.

From Lemma 8, the triangulation

$$\mathcal{T}_{\mathbf{N}} = \bigcup_{\boldsymbol{\imath}\in\mathcal{I}} \mathcal{K}(\mathrm{H}_{\boldsymbol{\imath}})$$

is a conforming triangulation of $\mathcal{Q}_{\mathbf{N}}$ with $n_{\mathrm{me}} = d! \times N_h$ d-simplices and by construction the vertices of $\mathcal{T}_{\mathbf{N}}$ are the vertices of $\mathcal{Q}_{\mathbf{N}}$:

$$\mathcal{T}_{\mathbf{N}}.\mathbf{q} = \mathcal{Q}_{\mathbf{N}}.\mathbf{q}.$$

It thus remains us to calculate the connectivity array $\mathbf{me}$ of $\mathcal{T}_{\mathbf{N}}$ also denoted by $\mathcal{T}_{\mathbf{N}}.\mathbf{me}$. This is a $(d+1)$-by-n_{me} array. For a given hypercube $\mathrm{H}_{\boldsymbol{\imath}}$ we store consecutively in the array $\mathbf{me}$, the $d!$ simplices given by $\mathcal{K}(\mathrm{H}_{\boldsymbol{\imath}})$

The Kuhn's triangulation for the reference hypercube $[0,1]^d$ can be obtained from the function KuhnTriangulation :

$$[\mathbf{q}_{\mathrm{K}}, \mathbf{me}_{\mathrm{K}}] \leftarrow \mathrm{KuhnTriangulation}(d)$$

Let $\boldsymbol{\imath} \in \mathcal{I}$ and $k = \mathcal{H}(\boldsymbol{\imath})$ where $\mathcal{H}$ is defined by (7). Let $l \in [\![1, d!]\!]$. We choose to store the l-th simplex of $\mathcal{K}(\mathrm{H}_{\boldsymbol{\imath}})$ in $\mathbf{me}(:, d!(k-1)+l)$.

Let $j \in [\![1, d]\!]$. The j-th vertices of the l-th simplex of $\mathcal{K}(\mathrm{H}_{\boldsymbol{\imath}})$ is $\mathbf{q}(:, \mathbf{me}(j, d!(k-1)+l))$ but it's also given by

$$\boldsymbol{x}^{\boldsymbol{\imath}} + \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l)) = \boldsymbol{\imath} + \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l))$$

So we want to determine the index $\mathbf{me}(j, d!(k-1)+l)$. From (6), we obtain

$$\mathbf{me}(j, d!(k-1)+l) = \mathcal{G}(\boldsymbol{\imath} + \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l))).$$

By using Lemma 6, we deduce that

$$\mathbf{me}(j, d!(k-1)+l) = \mathcal{G}(\boldsymbol{\imath}) + \langle \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l)), \boldsymbol{\beta} \rangle$$

and with (8) we obtain

$$\mathbf{me}(j, d!(k-1)+l) = \mathbf{iBase}(k) + \langle \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l)), \boldsymbol{\beta} \rangle, \ \forall k \in [\![1, N_h]\!]$$

This formula can be vectorized in k: with $\mathbf{Idx} \leftarrow d![0 : N_h - 1] + l$ then

$$\mathbf{me}(j, \mathbf{Idx}) \leftarrow \mathbf{iBase} + \langle \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l)), \boldsymbol{\beta} \rangle.$$

We give in Algorithm 8 the function CGTriangulation which compute the triangulation of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$.

Algorithm 8 Function CGTriangulation : compute the triangulation of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$

```
Input :
  $\boldsymbol{N}$  :  array of $d$ integers, $\boldsymbol{N}(i) = N_i$.

Output :
  q   :  vertices array of the triangulation of $\mathcal{Q}_{\mathbf{N}}$.
         $d$-by-$n_{\mathrm{q}}$ array of reals (integer in fact) where $n_{\mathrm{q}} = \prod_{i=1}^{d}(N_i+1)$.
  me  :  connectivity array of the triangulation of $\mathcal{Q}_{\mathbf{N}}$.
         $(d+1)$-by-$n_{\mathrm{me}}$ array of integers where $n_{\mathrm{me}} = d! \prod_{i=1}^{d} N_i$. .

  Function [q, me] ← CGTriangulation ($\boldsymbol{N}$)
    q ← CartesianGridPoints($\boldsymbol{N}$)
    Hinv ← CartesianGridPoints($\boldsymbol{N} - 1$)
    [$\mathbf{q}_{\mathrm{K}}$, $\mathbf{me}_{\mathrm{K}}$] ← KuhnTriangulation($d$)
    $\boldsymbol{\beta}$ ← CGbeta($\boldsymbol{N}$)
    iBase ← $\boldsymbol{\beta}^{\mathrm{t}} * \mathbf{Hinv} + 1$
    Idx ← $d! * [0 : (N_h - 1)]$
    for $l$ ← 1 to $d!$ do
      Idx ← Idx + 1
      for $j$ ← 1 to $d + 1$ do
        me($j$, Idx) ← iBase + $\langle \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l)), \boldsymbol{\beta} \rangle$
      end for
    end for
  end Function
```

3.3 d-orthotope triangulation

Let $\mathcal{O}_d$ be the d-orthotope $[a_1, b_1] \times \cdots \times [a_d, b_d]$.

The mechanism is similar to that seen in section 2.3 while taking as a starting point the cartesian grid triangulation.

Algorithm 9 Function ORTHTRIANGULATION : d-orthotope regular tessellation by orthotopes

```
Input :
  N     : array of $d$ integers, $\boldsymbol{N}(i) = N_i$.
  a, b  : arrays of $d$ reals, $\boldsymbol{a}(i) = a_i$, $\boldsymbol{b}(i) = b_i$
Output :
  q     : array of $d$-by-$n_{\mathrm{q}}$ array of reals.
  me    : ...

    Function $[\mathbf{q}, \mathbf{me}] \leftarrow$ ORTHTRIANGULATION $(\boldsymbol{N}, \boldsymbol{a}, \boldsymbol{b})$
      $[\mathbf{q}, \mathbf{me}] \leftarrow$ CGTRIANGULATION$(\boldsymbol{N})$
      $\mathbf{q} \leftarrow$ BOXMAPPING$(\mathbf{q}, \boldsymbol{a}, \boldsymbol{b})$
    end Function
```

3.4 m-faces triangulations of a cartesian grid

Let $\mathcal{Q}_{\mathbf{N}}$ be the d-dimensional cartesian grid defined in section 2.2. Not to confuse the notations, we denote by $\mathcal{T}_{\boldsymbol{N}}.\mathbf{q}$ and $\mathcal{T}_{\boldsymbol{N}}.\mathbf{me}$ respectively the vertices and connectivity arrays of the triangulation of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$.

Let $m \in [\![0, d[\![$ and $k \in [\![1, E_{m,d}]\!]$. We want to determine $\mathcal{T}_{\boldsymbol{N}}^m(k)$ the triangulation obtained from the restriction of $\mathcal{T}_{\boldsymbol{N}}$ to its k-th m-face where the numbering of the m-faces is specified in section 2.4. We denote by

- $\mathcal{T}_{\boldsymbol{N}}^m(k).\mathbf{q}$, the (local) vertex array
- $\mathcal{T}_{\boldsymbol{N}}^m(k).\mathbf{me}$, the (local) connectivity array
- $\mathcal{T}_{\boldsymbol{N}}^m(k)$.toGlobal, the global indices such that

$$\mathcal{T}_{\boldsymbol{N}}^m(k).\mathbf{q} \equiv \mathcal{T}_{\boldsymbol{N}}.\mathbf{q}(:, \mathcal{T}_{\boldsymbol{N}}^m(k).\text{toGlobal}).$$

By construction, $\mathcal{T}_{\boldsymbol{N}}^m(k)$ is the triangulation by m-simplices of an m-hypercube in $\mathbb{R}^d$.

The only difference with the construction of $\mathcal{Q}_{\mathbf{N}}^m(k)$ given in section 2.5 is on the array $\mathbf{me}^w$. For $\mathcal{Q}_{\mathbf{N}}^m(k)$, we took

$$[\mathbf{q}^w, \mathbf{me}^w] \leftarrow \text{CGTESSHYP}(\boldsymbol{N}(\mathbb{R}(l, :)))$$

whereas for $\mathcal{T}_{\boldsymbol{N}}^m(k)$ we must take instead

$$[\mathbf{q}^w, \mathbf{me}^w] \leftarrow \text{CGTRIANGULATION}(\boldsymbol{N}(\mathbb{R}(l, :)))$$

So only one line changes in the Algorithm 5 to obtain the new one given in Algorithm 10 where the function CGTRIFACES computes $\mathcal{T}_{\boldsymbol{N}}^m(k)$, $\forall k \in 2^{d-m} n_c$.

Algorithm 10 Function CGTriFaces : compute all the m-face triangulations by m-simplices of the cartesian grid triangulation $\mathcal{T}_{\boldsymbol{N}}$.

```
Input :
  N    :  array of d integers, $\boldsymbol{N}(i) = N_i$.
  m    :  integer, $0 \leqslant m < d$
Output :
  $s\mathcal{T}_{\boldsymbol{N}}$  :  array of the triangulations of all the $m$-faces comming from
          the cartesian grid triangulation $\mathcal{T}_{\boldsymbol{N}}$.
          The length of $s\mathcal{T}_{\boldsymbol{N}}$ is $E_{m,d} = 2^{d-m}\binom{d}{m}$ (number of $m$-faces).

  Function $s\mathcal{T}_{\boldsymbol{N}} \leftarrow$ CGTriFaces $(\boldsymbol{N}, m)$
    $\boldsymbol{\beta} \leftarrow$ CGbeta$(\boldsymbol{N})$
    $n_c \leftarrow \binom{d}{m}$
    $\mathbb{L} \leftarrow$ Combs$([\![1, d]\!], d - m)$
    $\mathbb{R}(l, :) \leftarrow [\![1, d]\!] \backslash \mathbb{L}(l, :), \ \forall l \in [\![1, n_c]\!]$
    $\mathbb{S} \leftarrow$ CartesianGridPoints(Ones$(1, d - m)$)
    $k \leftarrow 1$
    for $l \leftarrow 1$ to $n_c$ do
      $[\mathbf{q}^w, \mathbf{me}^w] \leftarrow$ CGTriangulation$(\boldsymbol{N}(\mathbb{R}(l, :)))$
      $n_{\mathrm{q}}^l \leftarrow \prod_{s=1}^{m}(\boldsymbol{N}(\mathbb{R}(l, s)) + 1)$          ▷ or length of $\mathbf{q}^w$
      for $r \leftarrow 1$ to $2^{d-m}$ do
        $s\mathcal{T}_{\boldsymbol{N}}(k).\mathbf{q}(\mathbb{R}(l, :), :) \leftarrow \mathbf{q}^w$
        $s\mathcal{T}_{\boldsymbol{N}}(k).\mathbf{q}(\mathbb{L}(l, :), :) \leftarrow \left(\boldsymbol{N}(\mathbb{L}(l, :))^{\mathrm{t}} .* \mathbb{S}(:, r)\right) * $ Ones$(1, n_{\mathrm{q}}^l)$
        $s\mathcal{T}_{\boldsymbol{N}}(k).\mathbf{me} \leftarrow \mathbf{me}^w$
        $s\mathcal{T}_{\boldsymbol{N}}(k).\mathrm{toGlobal} \leftarrow 1 + \boldsymbol{\beta}^{\mathrm{t}} * s\mathcal{T}_{\boldsymbol{N}}(k).\mathbf{q}$
        $k \leftarrow k + 1$
      end for
    end for
  end Function
```

4 Efficiency of the algorithms

Based on previous algorithms, a Matlab toolbox, an Octave package and a python package were developped. They contains a simple class object `OrthMesh` which permits, in any dimension $d \geqslant 1$, to obtain a simplicial mesh or orthotope mesh with all their m-faces, $0 \leqslant m < d$. It is also possible with the method function `plot` of the class object `OrthMesh` to represent a mesh or its m-faces for $d \leqslant 3$.

In the folowing section, the class object `OrthMesh` is presented. Thereafter some warning statements on the memory used by these objects in high dimension are given. Finally computation times for orthotope meshes and simplicial meshes are given in dimension $d \in [\![1, 5]\!]$.

4.1 Class object `OrthMesh`

The aim of the class object `OrthMesh` is to use previous algorithms for creating an object which contains a mesh of a d-orthotope and all its m-face meshes. An elementary mesh class object `EltMesh` is used to store only one mesh, the main mesh as well as any of the m-face meshes. This class `EltMesh` also simplify the codes writing. Its fields are the following:

- d, space dimension
- m, kind of mesh ($m = d$ for the main mesh)
- type, 0 for simplicial mesh or 1 for orthotope mesh
- n_{q}, number of vertices
- $\mathbf{q}$, vertices array of dimension d-by-n_{q}
- n_{me}, number of mesh elements
- **me**, connectivity array of dimension $(d+1)$-by-n_{me} for simplices elements or 2^d-by-n_{me} for orthotopes elements
- toGlobal, index array linking local array $\mathbf{q}$ to the one of the main mesh
- label, name/number of this elementary mesh
- color, color of this elementary mesh (for plotting purpose)

Let the d-orthotope defined by $[a_1, b_1] \times \cdots \times [a_d, b_d]$. The class object `OrthMesh` corresponding to this d-orthotope contains the main mesh and all its m-face meshes, $0 \leqslant m < d$. Its Fields are the following

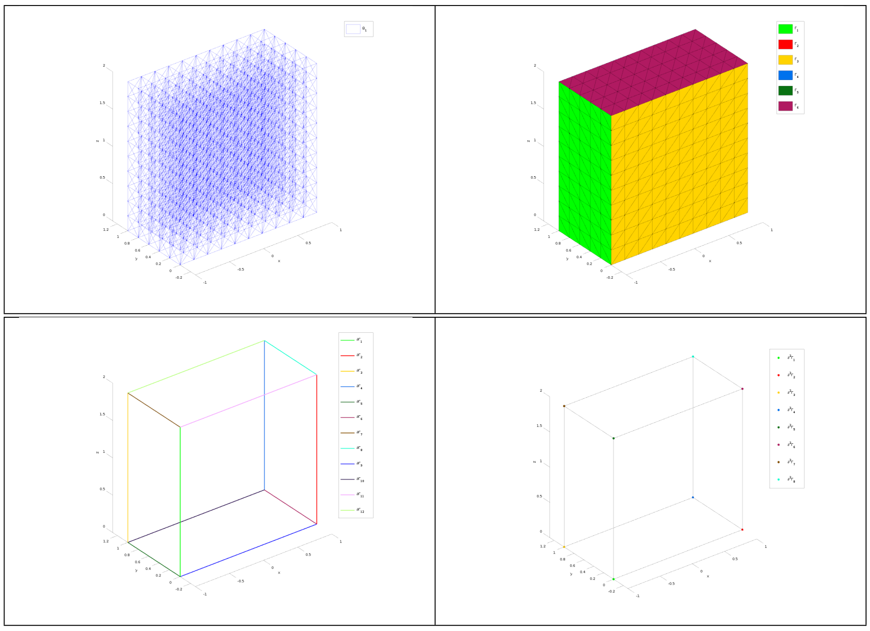
- d: space dimension
- type: string 'simplicial' or 'orthotope' mesh
- Mesh: main mesh as an `EltMesh` object
- Faces: list of arrays of `EltMesh` objects such that Faces(1) is an array of all the $(d-1)$-face meshes, Faces(2) is an array of all the $(d-2)$-face meshes, and so on
- box: a d-by-2 array such that box$(i, 1) = a_i$ and box$(i, 2) = b_i$.

The `OrthMesh` constructor is

$$Oh \leftarrow \textsc{OrthMesh}(d, \mathbf{N}, < \text{box} >, < \text{type} >)$$

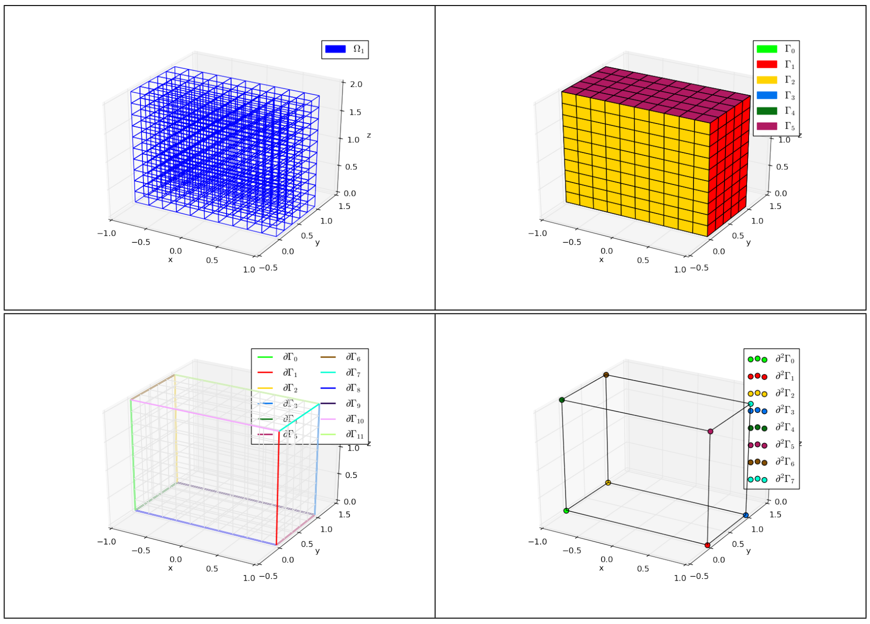
where $\mathbf{N}$ is either a 1-by-d array such that $\mathbf{N}(i)$ is the number of discretization for $[a_i, b_i]$ or either an integer if the the number of discretization is the same in all space directions. The optional parameter box previously described as for default value $a_i = 0$ and $b_i = 1$. The default value for optional parameter type is 'simplicial', otherwise 'orthotope' can be used.

In Listing 1, an `OrthMesh` object is built under Octave for the orthotope $[-1, 1] \times [0, 1] \times [0, 2]$ with simplicial elements and $\mathbf{N} = (10, 5, 10)$. The main mesh and all the m-face meshes of the resulting object are plotted. In Listing 2, similar operations are done under Python with orthotope elements.



Listing 1: 3D simplicial `OrthMesh` object with Octave 4.2.0, main mesh (upper left), 2-face meshes (upper right), 1-face meshes (bottom left) and 0-face meshes (bottom right)

```
Oh=OrthMesh(3,[10,5,10],'box',[-1,1;0,1;0,2])
% plot the main mesh
figure(1)
Oh.plot('legend',true)
axis equal;xlabel('x');ylabel('y');zlabel('z')
% plot the 2-face meshes
figure(2)
Oh.plot('m',2,'legend',true)
axis equal;xlabel('x');ylabel('y');zlabel('z')
% plot the 1-face meshes
figure(3)
Oh.plot('m',2,'color',[0.8,0.8,0.8],'EdgeAlpha',0.2, ...
    'FaceColor','none')
hold on
Oh.plot('m',1,'Linewidth',2,'legend',true)
axis equal;xlabel('x');ylabel('y');zlabel('z')
% plot the 0-face meshes
figure(4)
Oh.plot('m',1,'color','k')
hold on
Oh.plot('m',0,'legend',true)
axis equal;xlabel('x');ylabel('y');zlabel('z')
```



Listing 2: 3D orthotope `OrthMesh` object with Python 3.5.1, main mesh (upper left), 2-face meshes (upper right), 1-face meshes (bottom left) and 0-face meshes (bottom right)

```python
import matplotlib.pyplot as plt
from pyHyperMesh.OrthMesh import OrthMesh

oTh=OrthMesh(3,[10,5,10],type='orthotope',
    box=[[-1,1],[0,1],[0,2]])
# plot the main mesh
plt.ion()
plt.figure(1)
plt.clf()
oTh.plot(legend=True)
# plot the 2-face meshes
plt.figure(2)
plt.clf()
oTh.plot(m=2,legend=True,edgecolor=[0,0,0])
# plot the 1-face meshes
plt.figure(3)
plt.clf()
oTh.plot(m=2,edgecolor=[0.9,0.9,0.9],facecolor=None)
oTh.plot(m=1,legend=True,linewidth=2)
# plot the 0-face meshes
plt.figure(4)
plt.clf()
oTh.plot(m=1,color='black')
oTh.plot(m=0,legend=True,s=55)
```

Of course, the `plot` method doesn't work in dimension $d > 3$.

4.2 Memory consuming

Take care when using theses codes with memory consuming : the number of points n_{q} and the number of elements increases quickly according to the space dimension d. If $(N+1)$ points are taken in each space direction, we have

$$n_{\mathrm{q}} = (N+1)^d, \quad \text{for both tessellation and triangulation}$$

and

$$\begin{array}{lcll} n_{\mathrm{me}} & = & N^d, & \text{for tessellation by orthotopes} \\ n_{\mathrm{me}} & = & d!N^d, & \text{for tessellation by simplices.} \end{array}$$

If the array **q** is stored as *double* (8 octets) then

$$\text{mem. size of } \mathbf{q} = d \times n_{\mathrm{q}} \times 8 \text{ octets}$$

and if the array **me** as *int* (4 octets) then

$$\text{mem. size of } \mathbf{me} = \begin{cases} 2^d \times n_{\mathrm{me}} \times 4 \text{ octets} & \text{(tessellation by orthotopes)} \\ (d+1) \times n_{\mathrm{me}} \times 4 \text{ octets} & \text{(tessellation by simplices)} \end{cases}$$

For $N = 10$ and $d \in [\![1, 8]\!]$, the values of n_{q} and n_{me} are given in Table 3. The memory usage for the corresponding array **q** and array **me** is available in Table 4.

d	$n_{\mathrm{q}} = (N+1)^d$	$n_{\mathrm{me}} = N^d$ (orthotopes)	$n_{\mathrm{me}} = d!N^d$ (simplices)
1	11	10	10
2	121	100	200
3	1 331	1 000	6 000
4	14 641	10 000	240 000
5	161 051	100 000	12 000 000
6	1 771 561	1 000 000	720 000 000
7	19 487 171	10 000 000	50 400 000 000
8	214 358 881	100 000 000	4 032 000 000 000

Table 3: Number of vertices n_{q} and number of elements n_{me} for the tessellation of an orthotope by orthotopes and by simplices according to the space dimension d and with $N = 10$.

d	**q**	**me** (orthotopes)	**me** (simplices)
1	88 o	80 o	80 o
2	1 ko	1 ko	2 ko
3	31 ko	32 ko	96 ko
4	468 ko	640 ko	4 Mo
5	6 Mo	12 Mo	288 Mo
6	85 Mo	256 Mo	20 Go
7	1 Go	5 Go	1 612 Go
8	13 Go	102 Go	145 152 Go

Table 4: Memory usage of the array **q** and the array **me** for the tessellation of an orthotope by orthotopes and by simplices according to the space dimension d and with $N = 10$.

In the following pages, computational costs of the `OrthMesh` constructor will be presented.

4.3 Computational times

For all the following tables, the computational costs of the `OrthMesh` constructor are given for the orthotope $[-1,1]^d$ under Matlab R2016b, Octave 4.2.0 and Python 3.5.1. The computations were done on a laptop with `Core i7-4800MQ` processor and 16Go of RAM under Ubuntu 14.04 LTS (64bits). The details of Octave and Python installations for this Linux distribution are given respectively in [2] and [3].

In Table 5, some computational costs of the `OrthMesh` constructor

$$Oh \leftarrow \textsc{OrthMesh}(d, N, [-1;1]^d,\ '\mathit{orthotope}')$$

are given for $d \in [\![2,5]\!]$. Computational costs for tessellation with simplices are presented in Table 6 for $d \in [\![2,5]\!]$. In Appendix C, more detailed tables are given.

d	N	n_q	n_{me}	Python	Matlab	Octave
2	4000	16 008 001	16 000 000	0.725	0.678	1.077
3	250	15 813 251	15 625 000	1.151	1.616	2.415
4	62	15 752 961	14 776 336	1.803	3.188	5.142
5	27	17 210 368	14 348 907	3.167	6.032	10.559

Table 5: Tessellation of $[-1,1]^d$ by orthotopes with approximatively 15 millions elements. Computational times in seconds for Python 3.5.1, Matlab R2016b and Octave 4.2.0.

d	N	n_q	n_{me}	Python	Matlab	Octave
2	5000	25 010 001	50 000 000	1.831	2.428	3.834
3	180	5 929 741	34 992 000	1.881	2.583	4.488
4	40	2 825 761	61 440 000	4.855	5.411	8.992
5	12	371 293	29 859 840	2.914	3.091	6.726

Table 6: Tessellation of $[-1,1]^d$ with tens of millions of simplices. Computational times in seconds for Python 3.5.1, Matlab R2016b and Octave 4.2.0.

We can note that Octave is

A Vectorized algorithmic language

A.1 Common operators and functions

We also provide below some common functions and operators of the vectorized algorithmic language used in this article which generalize the operations on scalars to higher dimensional arrays, matrices and vectors:

$\mathbb{A} \leftarrow \mathbb{B}$	Assignment
$\mathbb{A} * \mathbb{B}$	matrix multiplication,
$\mathbb{A} \,.\!* \mathbb{B}$	element-wise multiplication,
$\mathbb{A} \,./\mathbb{B}$	element-wise division,
$\mathbb{A}(:)$	all the elements of $\mathbb{A}$, regarded as a single column.
$[,]$	Horizontal concatenation,
$[;]$	Vertical concatenation,
$\mathbb{A}(:, J)$	J-th column of $\mathbb{A}$,
$\mathbb{A}(I, :)$	I-th row of $\mathbb{A}$,
$\textsc{Sum}(\mathbb{A}, dim)$	sums along the dimension dim,
$\textsc{Prod}(\mathbb{A}, dim)$	product along the dimension dim,
$\mathbb{I}_n$	n-by-n identity matrix,
$\mathbb{1}_{m\times n}$ (or $\mathbb{1}_n$)	m-by-n (or n-by-n) matrix or sparse matrix of ones,
$\mathbb{O}_{m\times n}$ (or $\mathbb{O}_n$)	m-by-n (or n-by-n) matrix or sparse matrix of zeros,
$\textsc{Ones}(m, n)$	m-by-n array/matrix of ones,
$\textsc{Zeros}(m, n)$	m-by-n array/matrix of zeros,
$\textsc{RepTile}(\mathbb{A}, m, n)$	tiles the p-by-q array/matrix $\mathbb{A}$ to produce the $(m \times p)$-by-$(n \times q)$ array composed of copies of $\mathbb{A}$,
$\textsc{Reshape}(\mathbb{A}, m, n)$	returns the m-by-n array/matrix whose elements are taken columnwise from $\mathbb{A}$.

A.2 Combinatorial functions

$\textsc{Perms}(\boldsymbol{V})$	where $\boldsymbol{V}$ is an array of length n. Returns a $n!$-by-n array containing all the permutations of $\boldsymbol{V}$ elements. The lexicographical order is choosen.
$\textsc{Combs}(\boldsymbol{V}, k)$	where $\boldsymbol{V}$ is an array of length n and $k \in [\![1, n]\!]$. Returns a $\frac{n!}{k!(n-k)!}$-by-k array containing all the combinations of n elements taken k at a time. The lexicographical order is choosen.

B Function CartesianGridPoints

The objective is to explain how to obtain the vectorized function CartesianGridPoints given in Algorithm 1, section 2.2.1. This function returns the vertex array $\mathbf{q}$ of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$. The dimension of $\mathbf{q}$ is d-by-n_{q} with $n_{\mathrm{q}} = \prod_{i=1}^{d}(N_i + 1)$.

According to the numbering choice describe in section 2.2.1 the Algorithm 11 give the most simple presentation of $\mathbf{q}$ computation

Algorithm 11 Building $\mathbf{q}$ the d-by-n_{q} array of cartesian grid points

```
$k \leftarrow 1$
for $i_d \leftarrow 0$ to $N_d$ do
  for $i_{d-1} \leftarrow 0$ to $N_{d-1}$ do
    $\ddots$
    for $i_2 \leftarrow 0$ to $N_2$ do
      for $i_1 \leftarrow 0$ to $N_1$ do
        $\mathbf{q}(:, k) \leftarrow [i_1, i_2, \ldots, i_{d-1}, i_d]$
        $k \leftarrow k + 1$
      end for
    end for
    $\iddots$
  end for
end for
```

An other way to write this algorithm, with the coordinates for loop written, is given in Algorithm 12.

Algorithm 12 Building $\mathbf{q}$ the d-by-n_{q} array of cartesian grid points

```
$k \leftarrow 1$
for $i_d \leftarrow 0$ to $N_d$ do
  for $i_{d-1} \leftarrow 0$ to $N_{d-1}$ do
    $\ddots$
    for $i_2 \leftarrow 0$ to $N_2$ do
      for $i_1 \leftarrow 0$ to $N_1$ do
        for $r \leftarrow 1$ to $d$ do
          $\mathbf{q}(r, k) \leftarrow i_r$
        end for
        $k \leftarrow k + 1$
      end for
    end for
    $\iddots$
  end for
end for
```

Let $r \in [\![1, d]\!]$. From Algorithm 12, we deduce the Algorithm 13 which only computes the component r of the cartesian grid point C.

Algorithm 13 Setting component $r \in [\![1, d]\!]$ of cartesian grid points in $\mathbf{q}$ the d-by-n_q array

```
$k \leftarrow 1$,
**for** $i_d \leftarrow 0$ **to** $N_d$ **do**
  $\ddots$
  **for** $i_r \leftarrow 0$ **to** $N_r$ **do**
    **for** $i_{r-1} \leftarrow 0$ **to** $N_{r-1}$ **do**
      $\ddots$
      **for** $i_1 \leftarrow 0$ **to** $N_1$ **do**
        $\mathbf{q}(r, k) \leftarrow i_r$
        $k \leftarrow k + 1$
      **end for**
      $\iddots$
    **end for**
  **end for**
  $\iddots$
**end for**
```

One can replace the for loops i_1 to i_{r-1} by a for loop in j with number of iterations equal to $(N_1 + 1) \times \cdots \times (N_{r-1} + 1) = \beta_{r-1}$. This is done in Algorithm 14.

Algorithm 14 Setting component $r \in [\![1, d]\!]$ of cartesian grid points in $\mathbf{q}$ the d-by-n_q array

```
$k \leftarrow 1$,
**for** $i_d \leftarrow 0$ **to** $N_d$ **do**
  $\ddots$
  **for** $i_r \leftarrow 0$ **to** $N_r$ **do**
    **for** $j \leftarrow 1$ **to** $\beta_r$ **do**
      $\mathbf{q}(r, k) \leftarrow i_r$
      $k \leftarrow k + 1$
    **end for**
  **end for**
  $\iddots$
**end for**
```

We can replace the for loops in i_r and j by a call to the function BUILDPA given in Algorithm 15. The modified code using this function is given in Algorithm 16.

Algorithm 15 blabla

```
Input :
  $\boldsymbol{N}$ : array of $d$ integers, $\boldsymbol{N}(i) = N_i$.
  $r$ : $r \in [\![1, d]\!]$
Output :
  $\boldsymbol{A}$ : array of $\beta_{r+1} = (N_r + 1)\beta_r$ integers.

  Function $\boldsymbol{A} \leftarrow$ BuildPA $(\boldsymbol{N}, r)$
    $\beta_r \leftarrow \prod_{l=1}^{r-1}(\boldsymbol{N}(l) + 1)$, $k \leftarrow 1$,
    for $i_r \leftarrow 0$ to $N_r$ do
      for $j \leftarrow 1$ to $\beta_r$ do
        $\boldsymbol{A}(k) \leftarrow i$
        $k \leftarrow k + 1$
      end for
    end for
  end Function
```

Algorithm 16 Setting component $r \in [\![1, d]\!]$ of cartesian grid points in **q** the d-by-n_{q} array

```
  $\boldsymbol{I} \leftarrow 1 : \beta_{r+1}$
  for $i_d \leftarrow 0$ to $\boldsymbol{N}(d)$ do
    $\ddots$
    for $i_{r+1} \leftarrow 0$ to $\boldsymbol{N}(r + 1)$ do
      $\mathbf{q}(r, \boldsymbol{I}) \leftarrow$ BuildPA$(\boldsymbol{N}, r)$
      $\boldsymbol{I} \leftarrow \boldsymbol{I} + \beta_{r+1}$
    end for
    $\iddots$
  end for
```

As we can see, the BuildPA call in Algorithm 16 does not depend on the for loops i_d to i_{r+1}. Using this property and replacing the for loops i_d to i_{r+1} by a for loop in i with a number of iterations equal to $(N_d + 1) \times \cdots \times (N_{r+1} + 1)$ gives the first writeable code in Algorithm 17.

Algorithm 17 Setting component $r \in [\![1, d]\!]$ of cartesian grid points in the d-by-n_{q} array **q**

```
  $\boldsymbol{I} \leftarrow 1 : \beta_{r+1}$
  $\boldsymbol{A} \leftarrow$ BuildPA$(\boldsymbol{N}, r)$
  for $i \leftarrow 1$ to $(\boldsymbol{N}(d) + 1) \times \cdots \times (\boldsymbol{N}(r + 1) + 1)$ do
    $\mathbf{q}(r, \boldsymbol{I}) \leftarrow \boldsymbol{A}$
    $\boldsymbol{I} \leftarrow \boldsymbol{I} + \beta_{r+1}$
  end for
```

We can now write a complete nonvectorized function

Algorithm 18 Function CartesianGridPointsv0 : compute the d-by-n_q array $\mathbf{q}$ which contains all the points of the cartesian grid C (none vectorized version)

```
Input :
  $\boldsymbol{N}$   :   array of $d$ integers, $\boldsymbol{N}(i) = N_i$.

Output :
  $\mathbf{q}$   :   array of $d$-by-$n_\mathrm{q}$ array of integers.

  Function $\mathbf{q} \leftarrow$ CartesianGridPointsv0 $(\boldsymbol{N})$
    $\boldsymbol{\beta} \leftarrow$ CGbeta$(\boldsymbol{N})$
    for $r \leftarrow 1$ to $d$ do
      $\boldsymbol{I} \leftarrow 1 : \beta_{r+1}$
      $\boldsymbol{A} \leftarrow$ BuildPA$(\boldsymbol{N}, r)$
      for $i \leftarrow 1$ to $(\boldsymbol{N}(d)+1) \times \cdots \times (\boldsymbol{N}(r+1)+1)$ do
        $\mathbf{q}(r, \boldsymbol{I}) \leftarrow \boldsymbol{A}$
        $\boldsymbol{I} \leftarrow \boldsymbol{I} + \beta_{r+1}$
      end for
    end for
  end Function
```

To obtain a vectorized function, we must *work* on the i for loop and on the construction of the array $\boldsymbol{A}$.

We first vectorize the computation of array $\boldsymbol{A}$. Let us define the β_r-by-$(N_r + 1)$ array

$$\mathbb{A} = \begin{pmatrix} 0 & 1 & \dots & \boldsymbol{N}(r) \\ 0 & 1 & \dots & \boldsymbol{N}(r) \\ \vdots & \vdots & & \vdots \\ 0 & 1 & \dots & \boldsymbol{N}(r) \end{pmatrix}$$

obtained by copying array $[0 : \boldsymbol{N}(r)]$ on each row of $\mathbb{A}$ from

$$\mathbb{A} \leftarrow \textsc{RepTile}([0 : \boldsymbol{N}(r)], \beta_r, 1)$$

So array $\boldsymbol{A}$ can be obtained with the command

$$\boldsymbol{A} \leftarrow \textsc{Reshape}(\mathbb{A}, 1, (\boldsymbol{N}(r)+1)\beta_r)$$

or directly by

$$\boldsymbol{A} \leftarrow \textsc{Reshape}(\textsc{RepTile}([0 : \boldsymbol{N}(r)], \beta_r, 1), 1, (\boldsymbol{N}(r)+1)\beta_r)$$

We can easily vectorize the for loop i in function CartesianGridPointsv0 by using the RepTile function as follows

$$\mathbf{q}(r, :) \leftarrow \textsc{RepTile}(\boldsymbol{A}, 1, \textsc{Prod}(\boldsymbol{N}(r+1 : d) + 1))$$

With these two vectorizations we obtain the function CartesianGridPoints given in Algorithm 1.

C Computational costs

C.0.1 Tessellation by d-orthotopes

N	n_q	n_{me}	Python	Matlab	Octave
1000	1 002 001	1 000 000	0.143	0.33	0.192
2000	4 004 001	4 000 000	0.253	0.219	0.36
3000	9 006 001	9 000 000	0.444	0.41	0.656
4000	16 008 001	16 000 000	0.725	0.678	1.077
5000	25 010 001	25 000 000	1.075	1.03	1.622

Table 8: Tessellation of $[-1,1]^2$ by orthotopes

N	n_q	n_{me}	Python	Matlab	Octave
50	132 651	125 000	0.175	0.383	0.261
100	1 030 301	1 000 000	0.254	0.204	0.378
150	3 442 951	3 375 000	0.379	0.421	0.709
200	8 120 601	8 000 000	0.658	0.882	1.348
250	15 813 251	15 625 000	1.151	1.616	2.415
300	27 270 901	27 000 000	2.013	2.688	3.983
350	43 243 551	42 875 000	3.1	4.239	6.173

Table 9: Tessellation of $[-1,1]^3$ by orthotopes

N	n_q	n_{me}	Python	Matlab	Octave
10	14 641	10 000	0.27	0.448	0.531
20	194 481	160 000	0.268	0.146	0.565
30	923 521	810 000	0.338	0.269	0.783
40	2 825 761	2 560 000	0.532	0.604	1.338
50	6 765 201	6 250 000	0.921	1.337	2.551
62	15 752 961	14 776 336	1.803	3.188	5.142

Table 10: Tessellation of $[-1,1]^4$ by orthotopes

N	n_q	n_{me}	Python	Matlab	Octave
5	7 776	3 125	0.478	0.474	1.404
10	161 051	100 000	0.469	0.227	1.453
15	1 048 576	759 375	0.614	0.517	1.907
20	4 084 101	3 200 000	1.071	1.538	3.563
25	11 881 376	9 765 625	2.316	4.212	7.669
27	17 210 368	14 348 907	3.167	6.032	10.559

Table 11: Tessellation of $[-1,1]^5$ by orthotopes

C.0.2 Tessellation by d-simplices

N	n_q	n_{me}	Python	Matlab	Octave
1000	1 002 001	2 000 000	0.174	0.439	0.287
2000	4 004 001	8 000 000	0.378	0.448	0.727
3000	9 006 001	18 000 000	0.721	0.92	1.46
4000	16 008 001	32 000 000	1.216	1.577	2.512
5000	25 010 001	50 000 000	1.831	2.428	3.834

Table 12: Triangulation of $[-1, 1]^2$ by simplices

N	n_q	n_{me}	Python	Matlab	Octave
40	68 921	384 000	0.205	0.43	0.303
60	226 981	1 296 000	0.242	0.18	0.404
80	531 441	3 072 000	0.331	0.301	0.634
100	1 030 301	6 000 000	0.468	0.481	0.993
120	1 771 561	10 368 000	0.669	0.827	1.529
140	2 803 221	16 464 000	0.955	1.266	2.259
160	4 173 281	24 576 000	1.381	1.853	3.223
180	5 929 741	34 992 000	1.881	2.583	4.488

Table 13: Triangulation of $[-1, 1]^3$ by simplices

N	n_q	n_{me}	Python	Matlab	Octave
10	14 641	240 000	0.333	0.456	0.581
20	194 481	3 840 000	0.561	0.443	1.059
25	456 976	9 375 000	0.977	0.904	1.844
30	923 521	19 440 000	1.749	1.775	3.252
35	1 679 616	36 015 000	3.001	3.169	5.558

Table 14: Triangulation of $[-1, 1]^4$ by simplices

N	n_q	n_{me}	Python	Matlab	Octave
2	243	3 840	0.496	0.534	1.516
4	3 125	122 880	0.494	0.185	1.526
6	16 807	933 120	0.541	0.227	1.685
8	59 049	3 932 160	0.654	0.449	2.189
10	161 051	12 000 000	1.314	1.232	3.691
12	371 293	29 859 840	2.914	3.091	6.726

Table 15: Trianglation of $[-1, 1]^5$ by simplices

References

[1] Jürgen Bey. Simplicial grid refinement: on freudenthal's algorithm and the optimal number of congruence classes. *Numerische Mathematik*, 85(1):1–29,

2000.

[2] F. Cuvelier. Octave installation. `http://www.math.univ-paris13.fr/~cuvelier/Octave.html`, 2016.

[3] F. Cuvelier. Python installation. `http://www.math.univ-paris13.fr/~cuvelier/Python.html`, 2016.

[4] H. W. Kuhn. Some combinatorial lemmas in topology. *IBM Journal of Research and Development*, 4:518–524, 1960.

[5] K. Weiss. *Diamond-Based Models for Scientific Visualization*. PhD thesis, University of Maryland, 2011.