Vectorized algorithms for regular and conforming tessellations of d-orthotopes and their faces with high-order orthotopes or simplicial elements

Francois Cuvelier

Vectorized algorithms for regular and conforming tessellations of d-orthotopes and their faces with high-order orthotopes or simplicial elements

François Cuvelier *

2019/12/30

Abstract

In [8], vectorized algorithms are proposed to build regular and conforming tessellations of a d-orthotope made up by orthotopes or by simplices. We extend theses results to the tessellations of a d-orthotope with high-order elements.

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.

In [7] or [8], we explain how to efficiently build regular tessellations of a d-orthotope made up by orthotopes or by simplices and how to recover all the meshes associated to the m-faces of the d-orthotope, $0 \leqslant m \leqslant d$. In Figure 1 small meshes of the unit hypercube are given for both tessellations with orthotopes and simplices. From these two meshes, all the associated 2-faces meshes are represented in Figure 2.

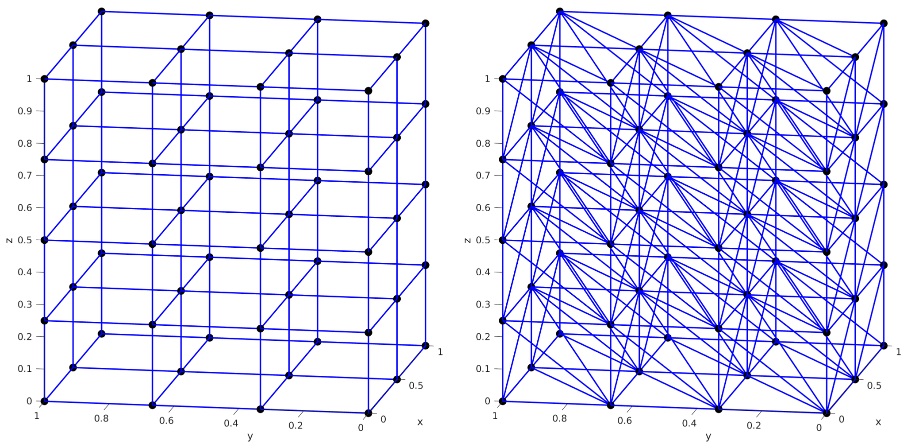

Figure 1: Tesselation samples of $[0,1]^3$ with 3-orthotopes (left) and 3-simplices (right) where vertices of all mesh elements are represented by a small black sphere.

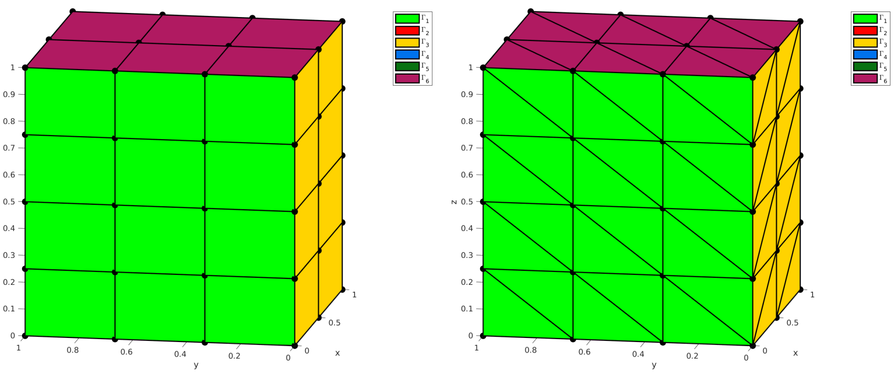

Figure 2: Representation of all the 2-faces meshes with 2-orthotopes (left) and 2-simplices (right) obtained from the tesselation samples of the Figure 1

The aim of this paper is to extend these results/algorithms to tessalations with high order elements: p-order orthotopes or p-order simplices. Theses elements have additionnal nodes regularly distributed added to their vertices. For dimension 1 to 3 and order 1 to 4, orthotope elements and simplicial elements are respectively represented In Table 1 and Table 2. In [8] the only mesh elements used are order 1.

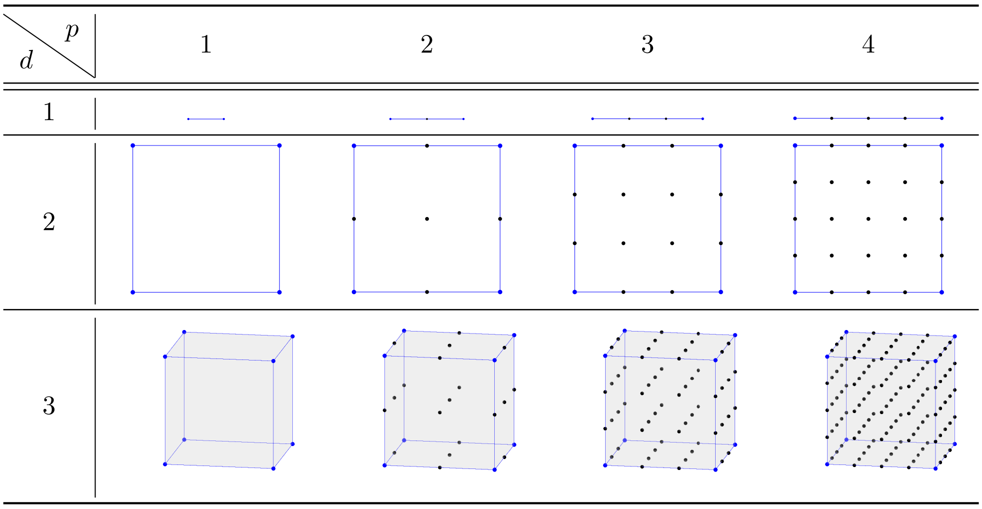

Table 1: p-order d-orthotope mesh element in $\mathbb{R}^d$. Nodes are the points.

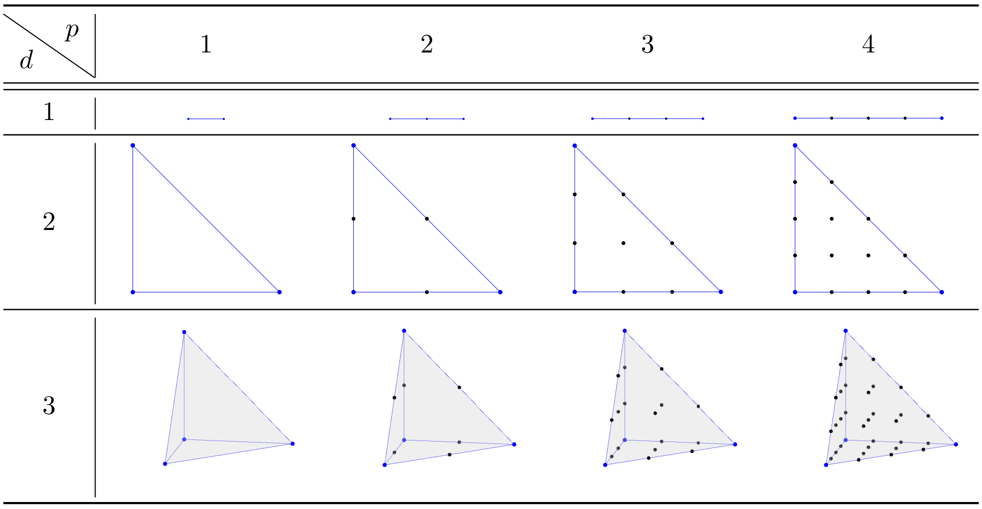

Table 2: p order d-simplicial mesh element in $\mathbb{R}^d$. Nodes are the points.

By taking back the meshes represented in Figure 1 and Figure 2, but this time using 3-order mesh element we want to get new meshes given by Figure 3 and Figure 4

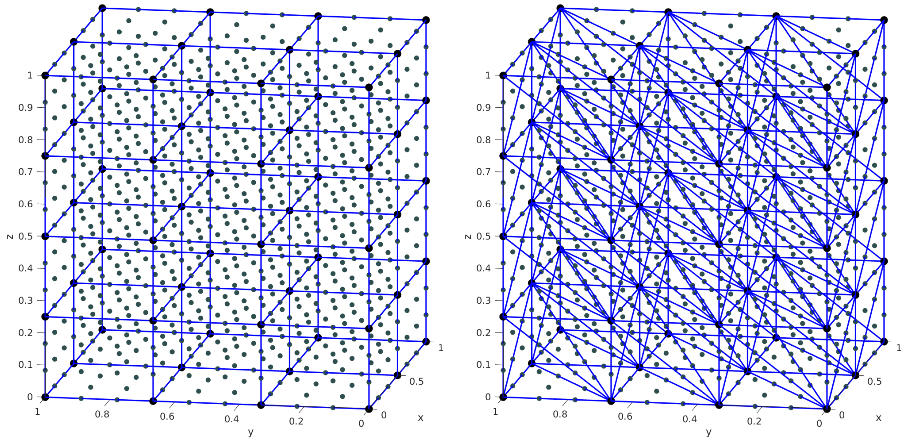

Figure 3: Tesselation samples of $[0,1]^3$ with 3-order 3-orthotopes (left) and 3-order 3-simplices (right) where nodes of all mesh elements are represented by small black (vertices) and grey spheres.

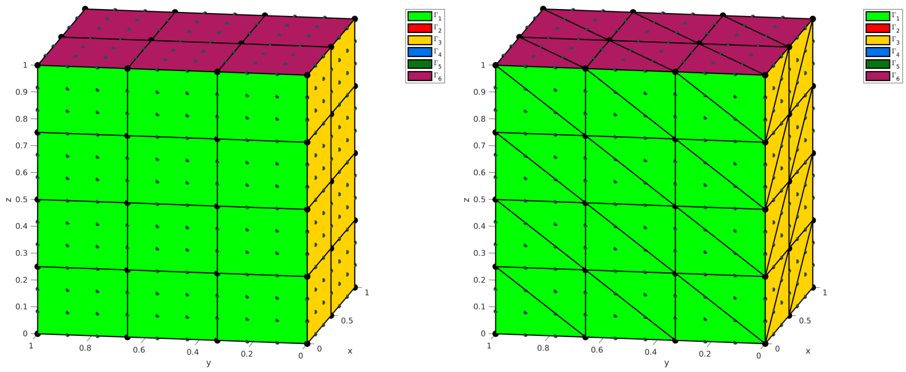

Figure 4: Representation of all the 2-faces meshes with 3-order 2-orthotopes (left) and 3-order 2-simplices (right) obtained from the tesselation samples of the Figure 3

In the following of the paper we will use notations and definitions given in [8].

1 Definitions and notations

In this part, we characterize the basic geometric elements that will be used later on. Some of their properties are recalled. But before we specify notations commonly used in this paper to define set of integers:

$$\llbracket i,j \rrbracket \stackrel{\mathsf{def}}{=} \{i,\cdots,j\}, \qquad \llbracket i,j \llbracket \stackrel{\mathsf{def}}{=} \{i,\cdots,j-1\},$$
$$\rrbracket i,j \rrbracket \stackrel{\mathsf{def}}{=} \{i+1,\cdots,j\}, \qquad \rrbracket i,j \llbracket \stackrel{\mathsf{def}}{=} \{i+1,\cdots,j-1\}.$$

1.1 d-orthotope and d-hypercube

We first recall the definitions of a d-orthotope and a d-hypercube given in [3].

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 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

$$E_{m,d} = 2^{d-m} \binom{d}{m} \quad \text{where} \quad \binom{d}{m} = \frac{d!}{m!(d-m)!} \tag{1}$$

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 5 all the m-faces of a 3D hypercube.

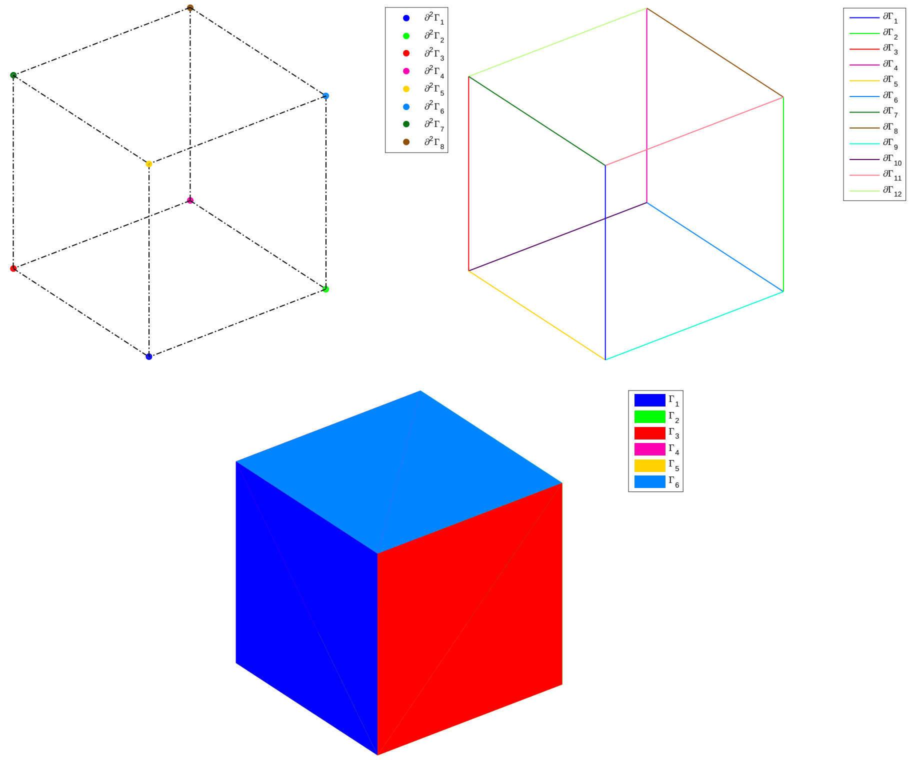

Figure 5: m-faces of a 3D hypercube : 0-faces (upper left), 1-faces (upper right) and 2-faces (bottom)

In Table 3 is given the number of m-faces for $m \in [\![0, d]\!]$ and $d \in [\![0, 6]\!]$.

	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 3: Number of m-faces of a d-hypercube

The identification/numbering of the m-faces is given in section2.3.

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 $\mathbf{q}^0, \ldots, \mathbf{q}^d \in \mathbb{R}^d$ are affinely independent, which means $\mathbf{q}^1 - \mathbf{q}^0, \ldots, \mathbf{q}^d - \mathbf{q}^0$ are linearly independent. Then, the simplex determined by them is the set of points*

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

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 which contain the given vertices.

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

$$\mathbb{D}_K = \left(\begin{array}{c:c:c} \mathbf{q}^0 & \ldots & \mathbf{q}^d \\ \hdashline 1 & \ldots & 1 \end{array}\right)$$

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, denoted by $S_{m,d}$, is given by

$$S_{m,d} = \binom{d+1}{m+1} \tag{2}$$

We give in Table 4 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 4: Number of m-faces of a nondegenerate d-simplex

2 Tessellation with high-order d-orthotope elements

2.1 High-order d-orthotope mesh elements in $\mathbb{R}^d$

The reference element of the p-order d-orthotope mesh element in $\mathbb{R}^d$ is $\widehat{\mathrm{H}} = [0,1]^d$. Its $(p+1)^d$ nodes are

$$\boldsymbol{x}^{\boldsymbol{i}} = \frac{\boldsymbol{i}}{p}, \quad \forall \boldsymbol{i} \in [\![0, p]\!]^d \tag{3}$$

and they contains the 2^d vertices of $\widehat{\mathrm{H}}$

$$\boldsymbol{x}^{\boldsymbol{i}} = \frac{\boldsymbol{i}}{p}, \quad \forall \boldsymbol{i} \in \{0, p\}^d. \tag{4}$$

Let $\widehat{\mathbf{q}}$ be the d-by-$(p+1)^d$ array containing all the nodes of $\widehat{\mathrm{H}}$. To choose the storage order of the nodes in the $\widehat{\mathbf{q}}$ array we define the $\mathcal{L}_p$ function that maps all the d-tuples $\boldsymbol{\imath} \in [\![0, p]\!]^d$ into $[\![1, (p+1)^d]\!]$ by

$$\mathcal{L}_p(\boldsymbol{\imath}) = 1 + \sum_{l=1}^{d} (p+1)^{l-1} \boldsymbol{\imath}_l. \tag{5}$$

Then the $\widehat{\mathbf{q}}$ is given by

$$\widehat{\mathbf{q}}(:, j) \stackrel{\text{def}}{=} \boldsymbol{x}^{\mathcal{L}_p^{-1}(j)}, \quad \forall j \in [\![1, (p+1)^d]\!]. \tag{6}$$

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

For example, with $d = 3$ and $p = 2$, the array $\widehat{\mathbf{q}}$ is given by

$$\widehat{\mathbf{q}} \stackrel{\text{def}}{=} \frac{1}{2} \begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

This array can be obtained from the CartesianGridPoints function, described in [8], by using

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

In Figure 6 and Figure 7, nodes numbering is represented respectively for the 2-orthotope and 3-orthotope reference elements of order 2 and 3.

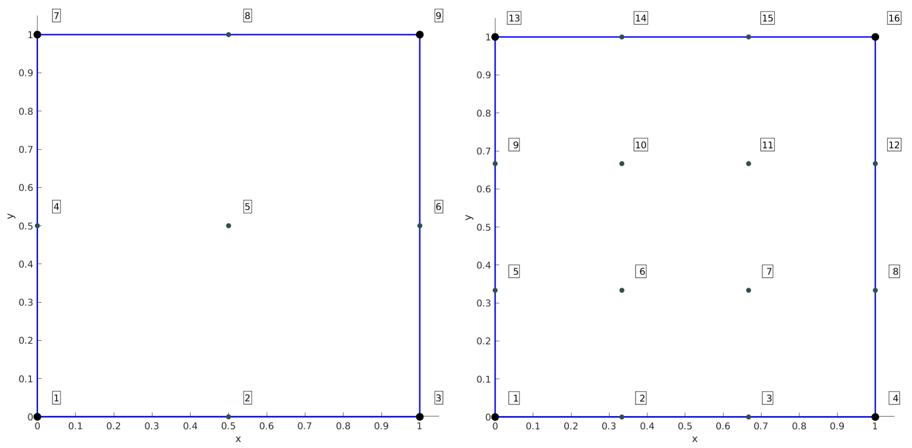

Figure 6: Nodes numbering of the unit 2-orthotope element in $\mathbb{R}^2$: order 2 (left) and order 3 (right)

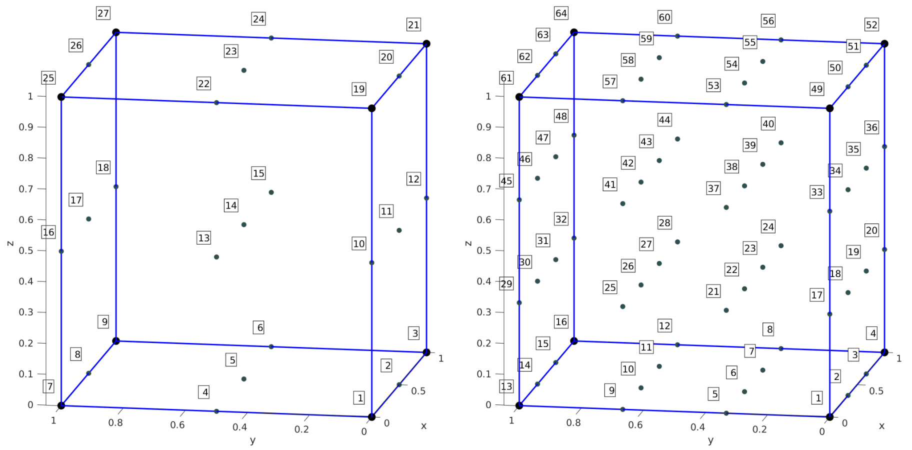

Figure 7: Nodes numbering of the unit 3-orthotope element in $\mathbb{R}^3$: order 2 (left) and order 3 (right)

2.2 Tessellation of a cartesian grid with p-order orthotopes

Let $\mathbf{N} = (N_1, \ldots, N_d) \in (\mathbb{N}^*)^d$. The cartesian grid of $[\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!]$ can be tessellated with p-order unit d-orthotopes which vertices are integer lattices. We denote by $\mathcal{Q}_{p,\mathbf{N}}$ this tessellation of $[\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!]$ composed of n_{q} node points and n_{me} p-order unit d-orthotopes where

$$n_{\mathrm{q}} = \prod_{l=1}^{d}(pN_l + 1) \quad \text{and} \quad n_{\mathrm{me}} = \prod_{l=1}^{d} N_l. \tag{7}$$

The objective of this section is to describe the construction of the nodes array $\mathbf{q}$ and the connectivity array $\mathbf{me}$ associated with $\mathcal{Q}_{p,\mathbf{N}}$. More precisely,

- $\mathbf{q}(\nu, j)$ is the ν-th coordinate of the j-th node, $\nu \in [\![1, d]\!]$, $j \in [\![1, n_{\text{q}}]\!]$. The j-th node will be also denoted by $\mathbf{q}^j = \mathbf{q}(:, j)$.
- $\mathbf{me}(\beta, l)$ is the storage index of the β-th node of the l-th element (unit hypercube), in the array q, for $\beta \in [\![1, (p+1)^d]\!]$ and $l \in [\![1, n_{\text{me}}]\!]$. So $\mathbf{q}(:, \mathbf{me}(\beta, l))$ represents the coordinates of the β-th node in the l-th cartesian grid element.

2.2.1 Nodes of the tessellation

Each node of the $\mathcal{Q}_{p,\mathbf{N}}$ tessellation may be identified by a d-tuple $\boldsymbol{\jmath} = (j_1, j_2, \cdots, j_d) \in [\![0, pN_1]\!] \times \cdots \times [\![0, pN_d]\!]$ and the corresponding node denoted by $\boldsymbol{q}^{\boldsymbol{\jmath}}$ is given by

$$\boldsymbol{q}^{\boldsymbol{\jmath}} = \sum_{l=1}^{d} \frac{j_l}{p} \boldsymbol{e}^{[l]} = \frac{1}{p}(j_1, j_2, \cdots, j_d)^{\text{t}} = \frac{\boldsymbol{\jmath}}{p} \tag{8}$$

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

We want to store all the nodes of $\mathcal{Q}_{p,\mathbf{N}}$ 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 mapping function $\mathcal{G}_p$ given by

$$\mathcal{G}_p(\boldsymbol{\jmath}) = 1 + \sum_{l=1}^{d} j_l \beta_l = 1 + \langle \boldsymbol{\jmath}, \boldsymbol{\beta} \rangle, \quad \forall \boldsymbol{\jmath} \in [\![0, pN_1]\!] \times \cdots \times [\![0, pN_d]\!] \tag{9}$$

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

$$\beta_l^p = \prod_{j=1}^{l-1} (pN_j + 1), \ \forall l \in [\![1, d]\!]. \tag{10}$$

To build the $\boldsymbol{\beta}^p$ array one can use the CGBETA function defined in Algorithm 1:

$$\boldsymbol{\beta}^p \leftarrow \text{CGBETA}(p * \mathbf{N})$$

Algorithm 1 Function CGBETA : Computes β_l, $\forall l \in [\![1, d]\!]$, defined in (10)

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

Output :
$\boldsymbol{\beta}$: array of d integers such that $\boldsymbol{\beta}(l) = \beta_l$ defined in (10)

```
Function β ← CGBETA (N)
  β(1) ← 1
  for l ← 2 to d do
    β(l) ← β(l − 1) × (N(l − 1) + 1)
  end for
end Function
```

The $\mathcal{G}_p$ function maps the tuple set $[\![0, pN_1]\!] \times \cdots \times [\![0, pN_d]\!]$ to the global nodes index set $[\![1, n_{\text{q}}]\!]$. From this function, we define the nodes array $\mathbf{q}$ as

$$\mathbf{q}(:, \mathcal{G}_p(\boldsymbol{\jmath})) = \boldsymbol{q}^{\boldsymbol{\jmath}} = \frac{\boldsymbol{\jmath}}{p}, \quad \forall \boldsymbol{\jmath} \in [\![0, kN_1]\!] \times \cdots \times [\![0, kN_d]\!] \tag{11}$$

This array can be obtained from the CartesianGridPoints function, described in [8], by using

$$\mathbf{q} \leftarrow (1/p) * \text{CartesianGridPoints}(p * \mathbf{N})$$

From the array $\mathbf{q}$ defined in (11), we can now construct the tessellation of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$ with unit d-hypercubes.

2.2.2 Connectivity array of the tessellation

The $\mathcal{Q}_{p,\mathbf{N}}$ tessellation contains n_{me} unit p-order d-orthotopes. They can be uniquely identified by their node of minimal coordinates. Let $\boldsymbol{\imath} \in [\![0, N_1[\![\times \cdots \times [\![0, N_d[\![$. We denote by $\mathrm{H}_{\boldsymbol{\imath}}^p$ the unit p-order hypercube defined by its 2^d vertices:

$$\boldsymbol{q}^{p(\boldsymbol{\imath}+\boldsymbol{p})}, \ \forall \boldsymbol{p} \in [\![0, 1]\!]^d.$$

So all the nodes of $\mathrm{H}_{\boldsymbol{\imath}}^p$ are given by:

$$\boldsymbol{q}^{p\boldsymbol{\imath}+\boldsymbol{s}}, \ \forall \boldsymbol{s} \in [\![0, p]\!]^d.$$

We want to build the connectivity array $\mathbf{me}$ of dimensions $(p+1)^d$-by-n_{me} such that $\mathbf{me}(l, r)$ is the index in array $\mathbf{q}$ of the l-th node of the r-th p-order hypercube : this node is $\mathbf{q}(:, \mathbf{me}(l, r))$.

To define an order of storage of the hypercubes in the array $\mathbf{me}$, we will use the function $\mathcal{H}$ defined by

$$\mathcal{H}(\boldsymbol{\imath}) = 1 + \sum_{l=1}^{d} i_l \prod_{j=1}^{l-1} N_i, \quad \boldsymbol{\imath} \in [\![0, N_1[\![\times \cdots \times [\![0, N_d[\![\tag{12}$$

This bijective function maps the multi-index set $\mathcal{I}_{\mathbf{N}} = [\![0, N_1[\![\times \cdots \times [\![0, N_d[\![$ to the set $[\![1, n_{\text{me}}]\!]$ such that $r = \mathcal{H}(\boldsymbol{\imath})$.

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

$$\mathcal{I}_{\mathbf{N}} \leftarrow \text{CartesianGridPoints}(\boldsymbol{N} - 1).$$

then by construction we have

$$\mathcal{H}^{-1}(r) = \mathcal{I}_{\mathbf{N}}(:, r), \quad \forall r \in [\![1, n_{\text{me}}]\!]$$

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

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

From the 1-by-d array $\boldsymbol{\beta}^p$ defined in (10), we have $\mathcal{G}_p(p\boldsymbol{\imath}) = 1 + \langle p\boldsymbol{\imath}, \boldsymbol{\beta}^p \rangle$. Using matricial operations we can define the 1-by-n_{me} array $\mathbf{iBase}$ by

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

such that

$$\mathcal{G}_p(p\boldsymbol{\imath}) = \mathcal{G}_k \circ \mathcal{H}^{-1}(r) = \mathbf{iBase}(r). \tag{13}$$

Let $\boldsymbol{\imath} \in [\![0, N_1[\![\times \cdots \times [\![0, N_d[\![$ and $p = \mathcal{H}(\boldsymbol{\imath})$. We choose vertices local numbering in the r-th hypercube to be identical with that described in section 2.1. That is why we take

$$\mathbf{q}(:, \mathbf{me}(l, r)) = \boldsymbol{q}^{p\boldsymbol{\imath}} + \widehat{\mathbf{q}}(:, l) = \boldsymbol{q}^{p(\boldsymbol{\imath}+\widehat{\mathbf{q}}(:,l)^{\mathsf{t}})}$$

where $\widehat{\mathbf{q}}$ is defined by (6). So we obtain

$$\mathbf{me}(l, r) = \mathcal{G}_p(p(\boldsymbol{\imath} + \widehat{\mathbf{q}}(:, l)^{\mathsf{t}})) \tag{14}$$

From definition of $\mathcal{G}_p$ we have

$$\begin{aligned}\mathbf{me}(l, r) &= 1 + \left\langle p(\boldsymbol{\imath} + \widehat{\mathbf{q}}(:, l)^{\mathsf{t}}), \boldsymbol{\beta}^p \right\rangle \\ &= 1 + \langle p\boldsymbol{\imath}, \boldsymbol{\beta}^p \rangle + \left\langle p\widehat{\mathbf{q}}(:, l), (\boldsymbol{\beta}^p)^{\mathsf{t}} \right\rangle \\ &= \mathcal{G}_p(p\boldsymbol{\imath}) + (\boldsymbol{\beta}^p)^{\mathsf{t}} * p\widehat{\mathbf{q}}(:, l).\end{aligned}$$

Thereafter, using (13) gives $\forall l \in [\![1, (p+1)^d]\!]$

$$\mathbf{me}(l, r) = \mathbf{iBase}(r) + \boldsymbol{\beta}^p * (p\widehat{\mathbf{q}}(:, l)), \;\; \forall r \in [\![1, n_{\mathrm{me}}]\!]$$

or in a partially vectorized form

$$\mathbf{me}(l, :) \leftarrow \mathbf{iBase} + (\boldsymbol{\beta}^p)^{\mathsf{t}} * (p\widehat{\mathbf{q}}(:, l)).$$

We can now give a full vectorized form:

$$\mathbf{me} \leftarrow \textsc{RepTile}(\mathbf{iBase}, (p+1)^d, 1) + \textsc{RepTile}(\textsc{Transpose}(\boldsymbol{\beta}^p * (p\widehat{\mathbf{q}})), 1, n_{\mathrm{me}})$$

So we can easily write the function CGTessHyp in Algorithm 2 which computes the $\mathbf{q}$ and $\mathbf{me}$ arrays.

Algorithm 2 Function CGTessHyp : computes the nodes array $\mathbf{q}$ and the connectivity array $\mathbf{me}$ obtained from a tesselation of the p-order cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$ with unit p-order hypercube.

Input :
- $\boldsymbol{N}$: array of d integers, $\boldsymbol{N}(i) = N_i$.
- p : positive integer.

Output :
- $\mathbf{q}$: nodes array of d-by-n_{q} integers.
- $\mathbf{me}$: connectivity array of $(p+1)^d$-by-n_{me} integers. $\mathbf{me}(l, r)$ is the index in the nodes array $\mathbf{q}$ of the l-th node of the r-th hypercube : this node is $\mathbf{q}(:, \mathbf{me}(l, r))$.

```
Function [q, me] ← CGTessHyp (N, p)
  q ← CartesianGridPoints(p * N)/p
  Hinv ← CartesianGridPoints(N − 1)                ▷ d-by-n_me array
  pq̂ ← CartesianGridPoints(p * Ones(1, d))         ▷ d-by-(p + 1)^d array
  β ← CGBeta(p * N)                                ▷ 1-by-d array
  iBase ← β * Hinv + 1
  me ← RepTile(iBase, (p + 1)^d, 1) + RepTile(Transpose(β * pq̂), 1, n_me)
end Function
```

2.3 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 $\mathbb{R}^d$ defined by the product of d intervals where $d-m$ intervals are reduced to the singleton $\{0\}$ or $\{1\}$ (called reduced dimension)

We have $n_c = \binom{d}{m}$ 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 n_c-by-$(d-m)$ array given by

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

Then each row of $\mathbb{L}^{[d,m]}$ contains the index of the $d-m$ reduced dimensions of an m-face sorted by lexicographical order (see COMBS function description in Appendix A)

Let $\mathbb{S}^{[d-m]}$ be the $(d-m)$-by-2^{d-m} array given by

$$\mathbb{S}^{[d-m]} \leftarrow \text{CARTESIANGRIDPOINTS}(\text{ONES}(1, d-m)).$$

This array contains 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.

Definition 5 *Let $l \in [\![1, n_c]\!]$, $r \in [\![1, 2^{d-m}]\!]$ and $k = 2^{d-m}(l-1) + r$. The k-th m-faces of the unit reference d-hypercube is defined by*

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

or in a vectorized form

$$\left\{\boldsymbol{x} \in [0,1]^d \text{ such that } \boldsymbol{x}(\mathbb{L}^{[d,m]}(l,:)) = \mathbb{S}^{[d-m]}(:,r)\right\} \tag{15}$$

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	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } x_1 = 0\}$
2	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } x_1 = 1\}$
3	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } x_2 = 0\}$
4	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } x_2 = 1\}$
5	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } x_3 = 0\}$
6	$\{\boldsymbol{x} \in [0,1]^3 \text{ such that } x_3 = 1\}$

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	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_2 = 0\}$
2	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_2 = 0\}$
3	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_2 = 1\}$
4	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_2 = 1\}$
5	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_3 = 0\}$
6	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_3 = 0\}$
7	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_3 = 1\}$
8	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_3 = 1\}$
9	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_2 = 0,\ x_3 = 0\}$
10	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_2 = 1,\ x_3 = 0\}$
11	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_2 = 0,\ x_3 = 1\}$
12	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_2 = 1,\ x_3 = 1\}$

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	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_2 = 0,\ x_3 = 0\}$
2	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_2 = 0,\ x_3 = 0\}$
3	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_2 = 1,\ x_3 = 0\}$
4	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_2 = 1,\ x_3 = 0\}$
5	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_2 = 0,\ x_3 = 1\}$
6	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_2 = 0,\ x_3 = 1\}$
7	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 0,\ x_2 = 1,\ x_3 = 1\}$
8	$\{\boldsymbol{x} \in [0,1]^3$ such that $x_1 = 1,\ x_2 = 1,\ x_3 = 1\}$

2.4 m-faces tessellations with high order ortotopes

In section 2.2.2, and especially in Algorithm 2, we have seen how to build the nodes array $\mathbf{q}$ and the connectivity array $\mathbf{me}$ of $\mathcal{Q}_{p,\mathbf{N}}$, the tessellation of cartesian grid with unit p-order d-orthotopes. So as not to confuse notations, we denote by $\mathcal{Q}_{p,\mathbf{N}}.\mathbf{q}$ and $\mathcal{Q}_{p,\mathbf{N}}.\mathbf{me}$ respectively these nodes and connectivity arrays of $\mathcal{Q}_{p,\mathbf{N}}$.

Let $m \in [\![0, d[\![$ and $k \in [\![1, E_{m,d}]\!]$. We want to determine $\mathcal{Q}^m_{p,\mathbf{N}}(k)$ the tessellation obtained from the restriction of tessellation $\mathcal{Q}_{p,\mathbf{N}}$ to its k-th m-face where the numbering of the m-faces is specified in section 2.3. So the $\mathcal{Q}^m_{p,\mathbf{N}}(k)$ tessellation is made with unit p-order m-orthotopes in $\mathbb{R}^d$. We denote by

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

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

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

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

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

So we obtain

$$x_j = N_j \times \mathbb{S}^{[d-m]}(s, r)$$

or, in a vectorized form using element-wise multiplication operator $.*$:

$$\boldsymbol{x}(\mathbb{L}^{[d,m]}(l,:)) = \boldsymbol{N}(\mathbb{L}^{[d,m]}(l,:)) \;.*\; \mathbb{S}^{[d-m]}(:, r). \tag{16}$$

Definition 6 *Let $l \in [\![1, n_c]\!]$, $r \in [\![1, 2^{d-m}]\!]$ and $k = 2^{d-m}(l-1)+r$. Then, the k-th m-faces of $\mathcal{Q}_{p,\mathbf{N}}$ is defined as the set*

$$\left\{\boldsymbol{x} \in \mathcal{Q}_{p,\mathbf{N}} \text{ such that } \boldsymbol{x}(\mathbb{L}^{[d,m]}(l,:)) = \boldsymbol{N}(\mathbb{L}^{[d,m]}(l,:)) \;.*\; \mathbb{S}^{[d-m]}(:, r)\right\} \tag{17}$$

2.4.1 Case $m = 0$.

If $m = 0$, the m-faces are the 2^d corner points of the cartesian grid. Then we have $\mathbb{L}^{[d,0]} = [\![1, d]\!]$ and $\mathbb{S}^{[d]}$ is an d-by-2^d array.

From (17), we obtain that $\forall k \in [\![1, (2^d]\!]$ the k-th 0-face of $\mathcal{Q}_{p,\mathbf{N}}$ is reduced to the point

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

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

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

So we obtain

$$\begin{aligned} \mathcal{Q}^0_{p,\mathbf{N}}(k).\mathbf{q} &= Q(:, k) \\ \mathcal{Q}^0_{p,\mathbf{N}}(k).\mathbf{me} &= 1 \\ \mathcal{Q}^0_{p,\mathbf{N}}(k).\text{toGlobal} &= \boldsymbol{\beta} * (p * Q(:, k)) + 1 \end{aligned}$$

where $\boldsymbol{\beta}$ is given by (10).

2.4.2 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 $\mathcal{Q}^m_{p,\mathbf{N}}(k)$ we first set a tessellation without the m constant dimensions given in $\mathbf{idc} = \mathbb{L}(l,:)$ (i.e. only with nonconstant dimensions in $\mathbf{idnc} = [\![1, d]\!] \backslash \mathbf{idc}$):

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

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

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

and the constants rows

$$\mathcal{Q}_{p,\mathbf{N}}^m(k).\mathbf{q}(\mathbf{idc}(i), :) \leftarrow p * \boldsymbol{N}(\mathbf{idc}(i)) * \mathbb{S}^{[d-m]}(i, r) * \text{ONES}(1, n_{\mathrm{q}}^l), \quad \forall i \in [\![1, d-m]\!]$$

In a vectorized way, we have

$$\begin{aligned} \mathcal{Q}_{p,\mathbf{N}}^m(k).\mathbf{q}(\mathbf{idnc}, :) &\leftarrow \mathbf{q}^w \\ \mathcal{Q}_{p,\mathbf{N}}^m(k).\mathbf{q}(\mathbf{idc}, :) &\leftarrow \left(\boldsymbol{N}(\mathbf{idc})^{\mathtt{t}} \; .\!* \; \mathbb{S}^{[d-m]}(:, r) \right) * \text{ONES}(1, n_{\mathrm{q}}^l) \end{aligned}$$

We immediately have the connectivity array

$$\mathcal{Q}_{p,\mathbf{N}}^m(k).\mathbf{me} = \mathbf{me}^w.$$

There still remains to compute $\mathcal{Q}_{p,\mathbf{N}}^m(k)$.toGlobal. For that we use the mapping function $\mathcal{G}_p$ defined in section 2.2.1 and more particularly (9). Indeed, for all $j \in [\![1, n_{\mathrm{q}}^l]\!]$, we can identify the point $\mathcal{Q}_{p,\mathbf{N}}^m(k).\mathbf{q}(:, j)$ by the d-tuple $\boldsymbol{\imath}$ and use it with the mapping function $\mathcal{G}_p$ to obtain the index in array $\mathcal{Q}_{p,\mathbf{N}}.\mathbf{q}$ of the point $\mathcal{Q}_{p,\mathbf{N}}^m(k).\mathbf{q}(:, j)$. So we have

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

and then

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

In a vectorized way, we set

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

with the vector $\boldsymbol{\beta}$ defined in (10).

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

We give in Algorithm 3 the function CGTESSHYPFACES which computes $\mathcal{Q}_{p,\mathbf{N}}^m(k)$, $\forall k \in [\![1, 2^{d-m} n_c]\!]$.

Algorithm 3 Function CGTessHypFaces : computes all m-faces tessellations of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$ with unit p-order m-hypercubes.

Input :
- $\boldsymbol{N}$: 1-by-d array of integers, $\boldsymbol{N}(i) = N_i$,
- m : integer, $0 \leqslant m < d$,
- p : positive integer.

Output :
- $\mathcal{Q}^m_{p,\mathbf{N}}$: array of tessellations of all m-faces of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$. Its length is $E_{m,d} = 2^{d-m}\binom{d}{m}$.

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

2.5 Tessellation of a d-orthotope with d-orthotopes

Let $\mathcal{O}_d$ be the d-orthotope $[a_1, b_1] \times \cdots \times [a_d, b_d]$. To obtain a regular tesselation of this orthotope with p-order orthotopes one can use an affine transformation of the p-order cartesian grid $\mathcal{Q}_{p,\mathbf{N}} = [\![0, N_1]\!] \times \cdots \times [\![0, N_d]\!]$ to $\mathcal{O}_d$. Let $\boldsymbol{a} = (a_1, \ldots, a_d)$ and $\boldsymbol{b} = (b_1, \ldots, b_d)$ be two vectors of $\mathbb{R}^d$. Let $\boldsymbol{N} \leftarrow [N_1, \ldots, N_d]$. The tessellation with p-order orthotope of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$ is obtained by

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

To obtain the tessellation of the orthotope $\mathcal{O}_d$ we only have to apply the affine transformation:

$$\mathbf{q}(i,:) \leftarrow \boldsymbol{a}(i) + \big((\boldsymbol{b}(i) - \boldsymbol{a}(i))/\boldsymbol{N}(i)\big) * \mathbf{q}(i,:), \quad \forall i \in [\![1, d]\!].$$

This operation is done by the function boxMapping given in Algorithm 4.

Algorithm 4 Function boxMapping : mapping points of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$ to the d-orthotope $[a_1, b_1] \times \cdots \times [a_d, b_d]$

Input :
- $\boldsymbol{N}$: array of d integers, $\boldsymbol{N}(i) = N_i$.
- $\mathbf{q}$: d-by-n_{q} array of integer obtained from $[\mathbf{q}, \mathbf{me}] \leftarrow \text{CGTessHyp}(\boldsymbol{N}, p)$
- $\boldsymbol{a}, \boldsymbol{b}$: arrays of d reals, $\boldsymbol{a}(i) = a_i$, $\boldsymbol{b}(i) = b_i$ with $a_i < b_i$

Output :
- $\mathbf{q}$: vertices array of d-by-n_{q} reals.

```
Function q ← boxMapping (q, a, b, N)
  for i ← 1 to d do
    q(i,:) ← a(i) + ((b(i) − a(i))/N(i)) ∗ q(i,:)
  end for
end Function
```

The function OrthTessOrth , which returns the arrays $\mathbf{q}$ and $\mathbf{me}$ corresponding to the regular tessellation of $\mathcal{O}_d$ with p-order d-orthotopes, is presented in Algorithm 5.

Algorithm 5 Function OrthTessOrth : d-orthotope regular tessellation with p-order orthotopes

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

Output :
- $\mathbf{q}$: array of d-by-n_{q} reals, $n_{\mathrm{q}} = \prod_{i=1}^{d}(pN_i + 1)$.
- $\mathbf{me}$: array of 2^d-by-n_{me} integers, $n_{\mathrm{me}} = \prod_{i=1}^{d} N_i$..

```
Function [q, me] ← OrthTessOrth (a, b, N, p)
  [q, me] ← CGTessHyp(N, p)
  q ← boxMapping(q, a, b)
end Function
```

2.6 m-faces tessellations of a d-orthotope

As seen in section 2.5, we only have to apply the function boxMapping to each array $\mathcal{Q}^m_{p,\mathbf{N}}(k).\mathbf{q}$ of the tessellations of the m-faces of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$. This is the object of the function OrthTessFaces given in Algorithm 6.

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

Input :

- $\boldsymbol{a}, \boldsymbol{b}$: arrays of d reals, $\boldsymbol{a}(i) = a_i$, $\boldsymbol{b}(i) = b_i$,
- $\boldsymbol{N}$: array of d integers, $\boldsymbol{N}(i) = N_i$,
- m : integer, $0 \leqslant m < d$,
- p : order, positive integer.

Output :

- $s\mathcal{O}_{\boldsymbol{h}}$: array of the tessellations of each m-faces of the orthotope. Its length is $E_{m,d} = 2^{d-m}\binom{d}{m}$.

```
Function sO_h ← ORTHTESSFACES (a, b, N, m, p)
  sO_h ← CGTESSHYPFACES(N, m, p)
  for k ← 1 to LEN(sO_h) do
    sO_h(k).q ← BOXMAPPING(sO_h(k).q, a, b, N)
  end for
end Function
```

3 Tessellation with high-order d-simplicial elements

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

The basic principle selected here is to start from a tesselation of a cartesian grid with unit hypercubes as obtained in section ??. Then by using the Kuhn's decomposition of an hypercube in simplices, we build in section ?? a tesselation of a cartesian grid with simplices and we explain how to obtain all its m-faces in section 3.5. Finally, ...

3.1 High-order d-simplicial mesh elements in $\mathbb{R}^d$

The reference element of the p-order d-simplicial mesh element in $\mathbb{R}^d$ is the simplex $\widehat{K}$ with vertices denoted by $\{\hat{\mathbf{q}}^0, \ldots, \hat{\mathbf{q}}^d\}$ and such that

$$\hat{\mathbf{q}}^0 = (0, \ldots, 0)^t, \text{ and } \hat{\mathbf{q}}^j = \boldsymbol{e}^{[j]}, \ \forall j \in [\![1, d]\!]$$

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

Let A_p be the subset of multi-index in $\mathbb{N}^d$ defined by

$$A_p = \left\{\boldsymbol{\alpha} \in \mathbb{N}^d : \ |\boldsymbol{\alpha}| \leqslant p\right\} \tag{18}$$

From Lemma 11 of Appendix C, The cardinality of A_p denoted by N_p is

$$N_p = C_{d+p}^p = \frac{(d+p)!}{d!p!}.$$

The N_p regular nodes of the reference element $\widehat{K}$ are

$$\boldsymbol{x}^{\boldsymbol{\alpha}} = \frac{\boldsymbol{\alpha}}{p}, \quad \forall \boldsymbol{\alpha} \in A_p \tag{19}$$

and they contains the $d+1$ vertices of $\widehat{K}$

$$\boldsymbol{x}^{\boldsymbol{\alpha}} = \frac{\boldsymbol{\alpha}}{p}, \quad \forall \boldsymbol{\alpha} \in \{0, p\}^d \text{ such that } \sum_{j=1}^{d} \boldsymbol{\alpha}_j \leqslant p. \tag{20}$$

In Figure 6 and Figure 7, nodes numbering is represented respectively for the 2-simplicial and 3-simplicial reference elements of order 2 and 3.

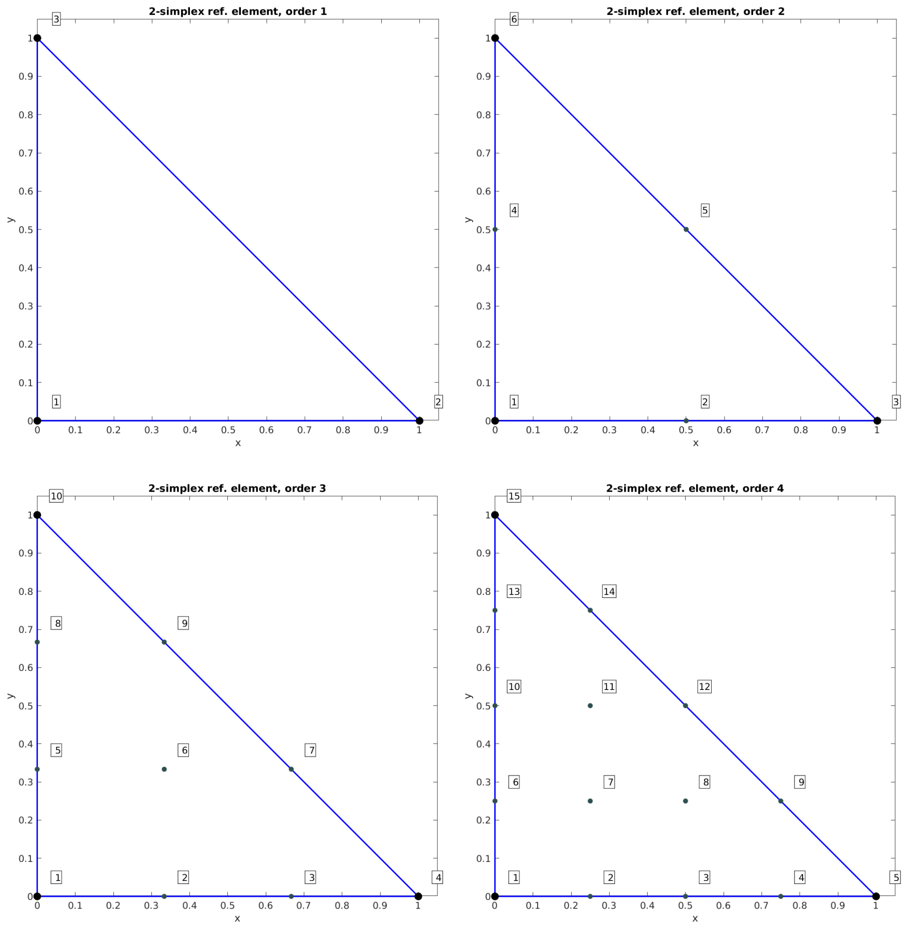

Figure 8: Nodes numbering of the unit 2-simplicial element in $\mathbb{R}^2$: order 1 (top left), order 2 (top right), order 3 (bottom left) and order 4 (bottom right)

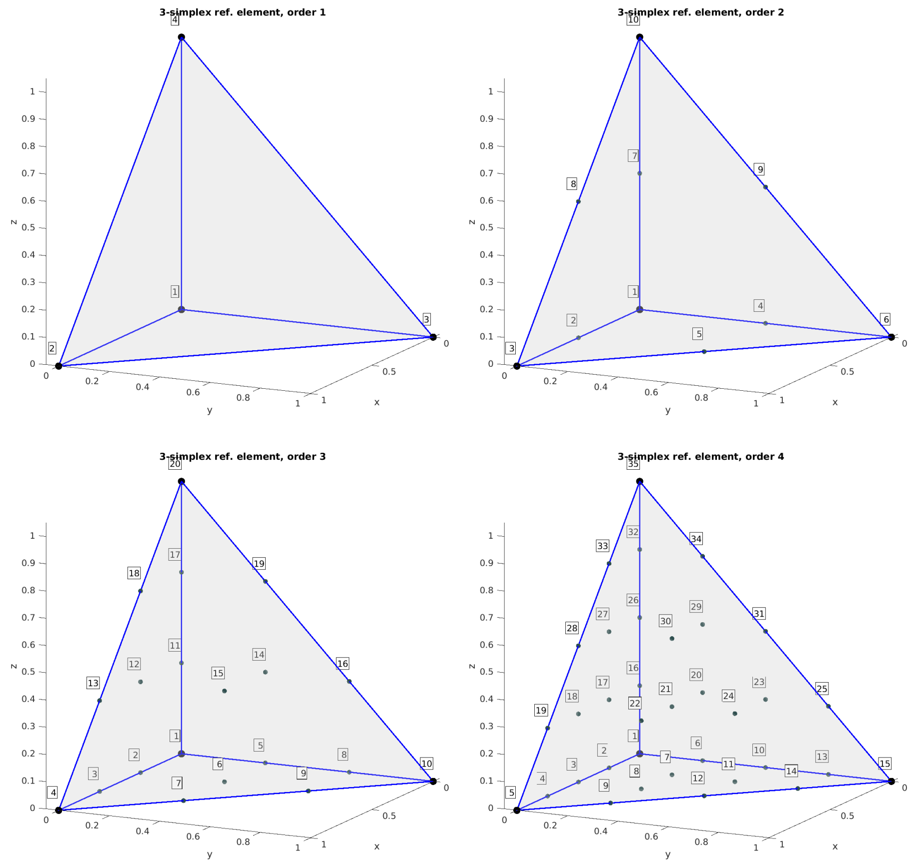

Figure 9: Nodes numbering of the unit 3-simplicial element in $\mathbb{R}^3$: order 1 (top left), order 2 (top right), order 3 (bottom left) and order 4 (bottom right)

In Algorithm 7, the NodesSimRef function returns nodes of the reference p-order d-simplex in $\mathbb{R}^d$ with nodes numbering described by Figures 8 and 9. They are obtained by selecting the nodes of the reference p-order d-orthotope $\widehat{\mathrm{H}}$ (see section 2.1) which are in the reference simplex $\widehat{K}$.

Algorithm 7 Function NodesSimRef : returns nodes of the reference p-order d-simplex in $\mathbb{R}^d$.

Input :
- d : space dimension, positive integer
- p : order, positive integer

Output :
- $\mathbf{q}$: vertices array of d-by-n_{q} reals with $n_{\mathrm{q}} = \frac{(d+p)!}{d!p!}$.

Function $\mathbf{q} \leftarrow$ NodesSimRef (d, p)
  $\mathbf{q} \leftarrow$ CartesianGridPoints$(p * $ Ones$(1, d))$
  $\mathbf{I} \leftarrow$ Find(Sum$(\mathbf{q}, 1) \leqslant p)$
  $\mathbf{q} \leftarrow \mathbf{q}(:, \mathbf{I})/p$
end Function

3.2 Kuhn's decomposition of a d-hypercube

Kuhn's subdivision (see [2, 10, 11]) 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 $\widehat{\mathrm{H}} = [0,1]^d$ be the unit d-hypercube in $\mathbb{R}^d$. Let $\mathbf{e}^{[1]}, \dots, \mathbf{e}^{[d]}$ be the standard unit basis vectors of $\mathbb{R}^d$ and denote by S_d the permutation group of $[\![1,d]\!]$. For all $\pi \in S_d$, the simplex K_π has for vertices $\{\boldsymbol{x}_\pi^{[0]}, \dots, \boldsymbol{x}_\pi^{[d]}\}$ defined by*

$$\boldsymbol{x}_\pi^{[0]} = (0,\dots,0)^t, \quad \boldsymbol{x}_\pi^{[j]} = \boldsymbol{x}_\pi^{[j-1]} + \mathbf{e}^{[\pi(j)]}, \ \forall j \in [\![1,d]\!]. \tag{21}$$

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

$$\mathcal{K}(\widehat{\mathrm{H}}) = \{K_\pi \mid \pi \in S_d\} \tag{22}$$

*is called the **Kuhn's subdivision** of $\widehat{\mathrm{H}}$ and its cardinality is $d!$.*

For example, we give in Figure 10 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 **q** and the connectivity array **me** are given by (préciser comment **me** est ordonné):

- for $d = 2$,

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

- for $d = 3$,

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

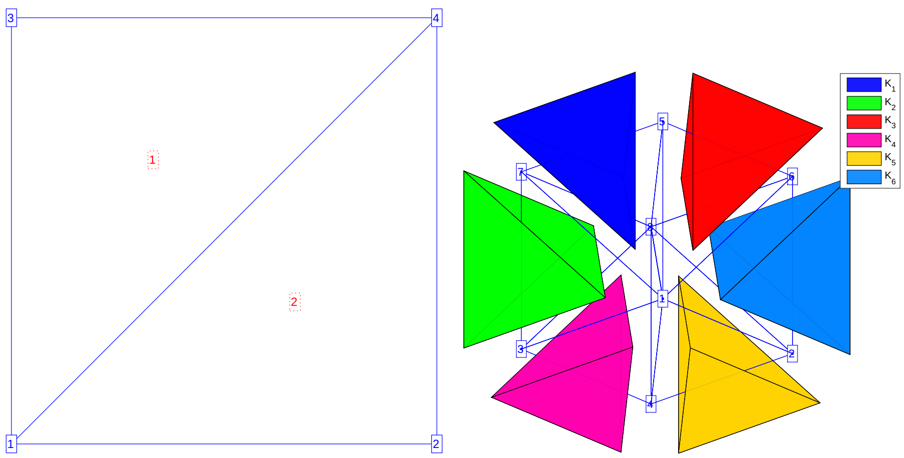

Figure 10: Kuhn's subdivision

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

$$\boldsymbol{x}^{[0]} = (0, \ldots, 0)^t, \quad \boldsymbol{x}^{[j]} = \boldsymbol{x}^{[j-1]} + \boldsymbol{e}^{[j]}, \ \forall j \in [\![1, d]\!]. \tag{23}$$

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 (21) can be derived from the reference simplex K_{ref} by

$$\boldsymbol{x}_\pi^{[j]} = \pi(\boldsymbol{x}^{[j]}), \quad \forall j \in [\![0, d]\!]. \tag{24}$$

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

$$\pi(K_{\text{ref}}) = K_\pi \tag{25}$$

Lemma 8 ([2], Lemma 4.1) *The* ***Kuhn's subdivision*** $\mathcal{K}(\widehat{\mathrm{H}})$ *of the unit d-hypercube $\widehat{\mathrm{H}}$ has the following properties:*

1. 0^d *and* 1^d *are common vertices of all elements* $K_\pi \in \mathcal{K}(\widehat{\mathrm{H}})$.
2. $\mathcal{K}(\widehat{\mathrm{H}})$ *is a consistent/conforming triangulation of* $\widehat{\mathrm{H}}$.
3. $\mathcal{K}(\widehat{\mathrm{H}})$ *is compatible with translation, i.e., for each vector* $\boldsymbol{v} \in [\![0, 1]\!]^d$ *the union of* $\mathcal{K}(\widehat{\mathrm{H}})$ *and* $\mathcal{K}(\boldsymbol{v} + \widehat{\mathrm{H}})$ *is a consistent/conforming triangulation of the set* $\widehat{\mathrm{H}} \cup (\boldsymbol{v} + \widehat{\mathrm{H}})$.
4. *For any affine transformation* $\mathcal{F}$*, the Kuhn's triangulation of* $\mathcal{F}(\widehat{\mathrm{H}})$ *is defined by* $\mathcal{K}(\mathcal{F}(\widehat{\mathrm{H}})) \stackrel{\text{def}}{=} \mathcal{F}(\mathcal{K}(\widehat{\mathrm{H}}))$.

To explicitly obtain a Kuhn's triangulation $\mathcal{K}(\widehat{\mathrm{H}})$ of the unit d-hypercube $\widehat{\mathrm{H}}$ 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 $\mathbf{q}^{\text{ref}}$ be the d-by-$(d+1)$ array of vertex coordinates of reference d-simplex K^{ref} :

$$\mathbf{q}^{\text{ref}} = \left(\boldsymbol{x}^{[0]} \,\vdots\, \boldsymbol{x}^{[1]} \,\vdots\, \ldots \quad \ldots \,\vdots\, \boldsymbol{x}^{[d]} \right) = \begin{pmatrix} 0 & 1 & \ldots & \ldots & 1 \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}$$

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. The array $\mathbf{P}$ is obtained by using the function PERMS defined in Appendix A.2. We use (24) and (25) to build the vertices of K_π. So the j-th vertex of K_π is given by

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

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

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

If the k-th d-simplex has 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 8, we give the function KuhnTriangulation which returns the points array $\mathbf{q}$ and the connectivity array $\mathbf{me}$ where all the d-simplices have a positive orientation.

Algorithm 8 Kuhn's triangulation of the unit d-hypercube $[0,1]^d$ with $d!$ simplices (positive orientation)

Input :
  d : space dimension

Output :
  $\mathbf{q}$: vertices array of d-by-2^d integers.
  $\mathbf{me}$: connectivity array of $(d+1)$-by-$d!$ integers

Function $[\mathbf{q}, \mathbf{me}] \leftarrow$ KuhnTri (d)
  $\mathbf{q} \leftarrow$ CartesianGridPoints(Ones$(1,d)$)
  $\mathbf{q}^{\text{ref}} \leftarrow \begin{pmatrix} 0 & 1 & \dots & \dots & 1 \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}$ ▷ a d-by-$(d+1)$ array
  $\mathbf{P} \leftarrow$ perms$(1:d)$ ▷ see Appendix A.2
  $\mathbf{me} \leftarrow \mathbb{0}_{d+1,d!}$
  $\boldsymbol{a} \leftarrow [2^0, 2^1, \dots, 2^{d-2}, 2^{d-1}]$
  for $k \leftarrow 1$ **to** $d!$ **do**
    for $j \leftarrow 1$ **to** $d+1$ **do**
      $\mathbf{me}(j,k) \leftarrow$ Dot$(\boldsymbol{a}, \mathbf{q}^{\text{ref}}(\mathbf{P}(:,k),j)) + 1$
    end for
    if Det$([\mathbf{q}(:,\mathbf{me}(:,k));$ Ones$(1,d+1)]) < 0$ **then**
      $t \leftarrow \mathbf{me}(1,k)$, $\mathbf{me}(1,k) \leftarrow \mathbf{me}(d+1,k)$, $\mathbf{me}(d+1,k) \leftarrow t$
    end if
  end for
end Function

3.3 Kuhn's decomposition of a d-hypercube by p-order simplices

We just sawn in section 3.2 the Kuhn's decomposition of the unit d-hypercube by 1-order simplices. To obtain the same decomposition with p-order simplices, $p > 1$, we must build a node array $\mathbf{q}$ and a connectivity array $\mathbf{me}$ with respectively dimensions d-by-$(p+1)^d$ and C^p_{d+p}-by-$d!$.

In Figures 11 and 12, the Kuhn's decomposition of the unit d-hypercube with 2-order simplices and 3-order simplices is represented respectively in dimension 2 and 3.

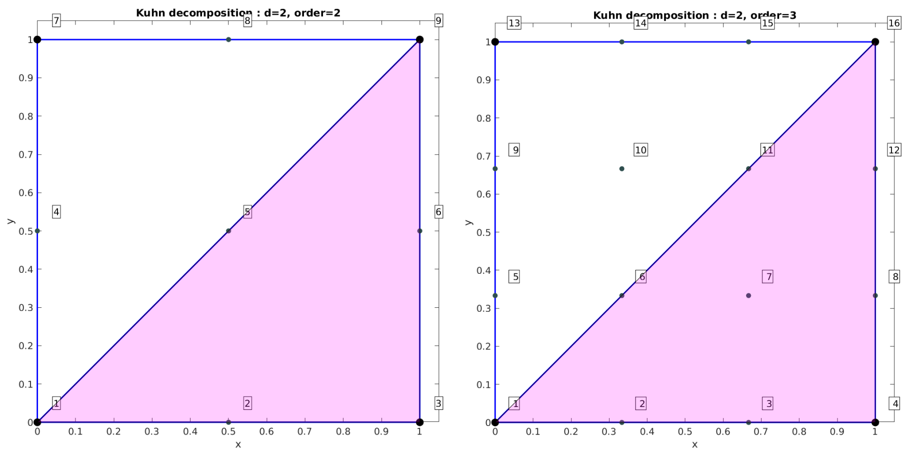

Figure 11: Kuhn's decomposition of the unit square by 2-order simplices (left) and 3-order simplices (right). The first element in the decomposition is colored.

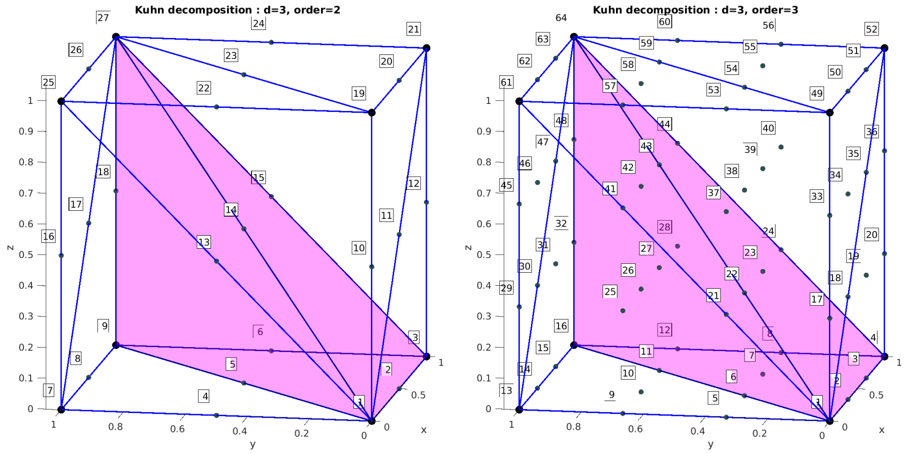

Figure 12: Kuhn's decomposition of the unit cube by 2-order simplices (left) and 3-order simplices (right). The first element in the decomposition is colored.

For example, with $d = 3$ and $p = 2$ (i.e. Figure 12, left graphic), the nodes array is

$$\mathbf{q} \stackrel{\mathsf{def}}{=} \frac{1}{2}\begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

and the connectivity array is

$$\mathbf{me} \stackrel{\text{def}}{=} \begin{bmatrix} 27 & 1 & 1 & 27 & 27 & 1 \\ 15 & 2 & 4 & 23 & 17 & 10 \\ 3 & 3 & 7 & 19 & 7 & 19 \\ 18 & 11 & 5 & 24 & 26 & 13 \\ 6 & 12 & 8 & 20 & 16 & 22 \\ 9 & 21 & 9 & 21 & 25 & 25 \\ 14 & 14 & 14 & 14 & 14 & 14 \\ 2 & 15 & 17 & 10 & 4 & 23 \\ 5 & 24 & 18 & 11 & 13 & 26 \\ 1 & 27 & 27 & 1 & 1 & 27 \end{bmatrix}$$

The nodes array is the one from the p-order d-orthotope mesh element given by (6) in section 2.1 and can be obtained from the CartesianGridPoints function, by using

$$\mathbf{q} \leftarrow (1/p) * \text{CartesianGridPoints}(p * \text{Ones}(1, d))$$

For building the connectivity array **me** one needs to define the mapping function from the unit reference d-simplex to a d-simplex. Let $\widehat{K}$ be the unit reference d-simplex with vertices denoted by $\{\widehat{\mathbf{q}}^0, \ldots, \widehat{\mathbf{q}}^d\}$ defined in section 3.1. Let $K \subset \mathbb{R}^d$ be a non-degenerate d-simplex and and $\{\mathbf{q}^{[i]}\}_{i=0}^d$ its vertices. The affine map/transformation $\mathcal{F}_K$ from the unit reference d-simplex $\widehat{K} \subset \mathbb{R}^d$ to $K \subset \mathbb{R}^d$ is given by

$$\boldsymbol{q} = \mathbb{A}_K \hat{\mathbf{q}} + \mathbf{q}^{[0]} = \mathcal{F}_K(\hat{\mathbf{q}}). \tag{26}$$

where $\mathbb{A}_K \in \mathcal{M}_{d,d}(\mathbb{R})$ is defined by

$$\mathbb{A}_K = \left(\mathbf{q}^{[1]} - \mathbf{q}^{[0]} \,\middle\vdots\, \cdots \,\middle\vdots\, \mathbf{q}^{[d]} - \mathbf{q}^{[0]} \right) \tag{27}$$

To build this function we only have to know the vertices of the K simplex.

From the KuhnTri function, we build the vertices array $\mathbf{q}^{\text{kt}}$ and the associated connectivity array $\mathbf{me}^{\text{kt}}$:

$$[\mathbf{q}^{\text{kt}}, \mathbf{me}^{\text{kt}}] \leftarrow \text{KuhnTri}(d).$$

Thereafter, for each l-th simplex of the Kuhn's decomposition, $l \in [\![1, d!]\!]$, we can build its vertices array

$$\mathbf{Q} \leftarrow \mathbf{q}^{\text{kt}}(:, \mathbf{me}^{\text{kt}}(:, l))$$

and then the matrix of the mapping function from $\widehat{K}$ to the l-th simplex is given by:

$$\mathbb{A} \leftarrow \mathbf{Q}(:, 2 : d+1) - \text{RepTile}(\mathbf{Q}(:, 1), 1, d).$$

From the NodesSimRef function given in Algorithm 7 we obtain nodes of the reference p-order d-simplex in $\mathbb{R}^d$:

$$\mathbf{q}^{\text{ref}} \leftarrow \text{NodesSimRef}(d, p).$$

So, by using mapping function (26), we obtain p-order nodes of the l-th simplex:

$$\mathbf{q}^{\mathrm{nod}} \leftarrow \mathbb{A} * \mathbf{q}^{\mathrm{ref}} + \mathbf{Q}(:,1)$$

Now to build the (p-order) connectivity array **me**, we must obtain their indices in the (p-order) nodes array **q**. From (3), we deduce that the multi-indices associated with nodes array are

$$\mathbf{inod} \leftarrow p * \mathbf{q}^{\mathrm{nod}}.$$

Then, from (5), the index $\mathbf{me}(j,l)$ of the j-th nodes of the l-th simplex in **q** array is

$$\mathbf{me}(j,l) \leftarrow \mathcal{L}_p(\mathbf{inod}(:,j)) \stackrel{\mathsf{def}}{=} 1 + \sum_{s=1}^{d} (p+1)^{s-1}\mathbf{inod}(s,j).$$

Let $\boldsymbol{\beta} \in \mathbb{N}^d$ such that $\boldsymbol{\beta}_s = (p+1)^{s-1}$ for all s in $[\![1,d]\!]$. By using the CGBETA function given in Algorithm 1 we obtain the 1-by-d array

$$\boldsymbol{\beta} \leftarrow \textsc{CGbeta}(p * \textsc{Ones}(1,d))$$

Then we have

$$\mathbf{me}(j,l) \leftarrow 1 + \textsc{Dot}\big(\boldsymbol{\beta}, \mathbf{inod}(:,j)\big).$$

Finally, using matricial product gives

$$\mathbf{me}(:,l) \leftarrow 1 + \boldsymbol{\beta} * \mathbf{inod}$$

A complete function is given in Algorithm 9.

Algorithm 9 Kuhn's triangulation of the unit d-hypercube $[0,1]^d$ with $d!$ p-order simplices (positive orientation)

Input :
- d : space dimension, positive integer
- p : order, positive integer

Output :
- **q** : vertices array of d-by-$(p+1)^d$ integers.
- **me** : connectivity array of C^p_{d+p}-by-$d!$ integers

Function $[\mathbf{q}, \mathbf{me}] \leftarrow \textsc{KuhnTriOrder}\ (d, p)$
  $[\mathbf{q}^{\mathrm{kt}}, \mathbf{me}^{\mathrm{kt}}] \leftarrow \textsc{KuhnTri}(d)$
  if $p == 1$ **then**
    $\mathbf{q} \leftarrow \mathbf{q}^{\mathrm{kt}}$, $\mathbf{me} \leftarrow \mathbf{me}^{\mathrm{kt}}$, **return**
  end if
  $\mathbf{q} \leftarrow \textsc{CartesianGridPoints}(p * \textsc{Ones}(1,d))/p$
  $\mathbf{q}^{\mathrm{ref}} \leftarrow \textsc{NodesSimRef}(d,p)$
  $\boldsymbol{\beta} \leftarrow \textsc{CGbeta}(p * \textsc{Ones}(1,d))$
  for $l \leftarrow 1$ **to** $d!$ **do**
    $\mathbf{Q} \leftarrow \mathbf{q}^{\mathrm{kt}}(:, \mathbf{me}^{\mathrm{kt}}(:,l))$
    $\mathbb{A} \leftarrow \mathbf{Q}(:, 2:d+1) - \textsc{RepTile}(\mathbf{Q}(:,1), 1, d)$
    $\mathbf{q}^{\mathrm{nod}} \leftarrow \mathbb{A} * \mathbf{q}^{\mathrm{ref}} + \mathbf{Q}(:,1)$
    $\mathbf{me}(:,l) \leftarrow 1 + \boldsymbol{\beta} * (p * \mathbf{q}^{\mathrm{nod}})$
  end for
end Function

From this tesselation of the unit reference d-hypercube, we will see how to get a regular tessellation of a cartesian grid with p-order simplices.

3.4 Cartesian grid tesselation with p-order simplices

Let $\mathcal{Q}_{p,\mathbf{N}}$ be the d-dimensional cartesian grid tessellated with p-order unit d-orthotopes and defined in section 2.2. As before, so as not to confuse notations, we denote by $\mathcal{Q}_{p,\mathbf{N}}.\mathbf{q}$ and $\mathcal{Q}_{p,\mathbf{N}}.\mathbf{me}$ respectively the nodes and connectivity arrays of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$. There are $N_h = \prod_{i=1}^{d} N_i$ unit hypercubes in this tessellation.

Let $\mathcal{I} = [\![0, N_1[\![\times \ldots \times [\![0, N_d[\![$. By using the CartesianGridPoints function, one can build the d-by-N_h array:

$$\mathcal{I} \leftarrow \textsc{CartesianGridPoints}()\mathbf{N} - 1).$$

We have

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

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

From Lemma 8, the triangulation

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

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

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

It thus remains to calculate the connectivity array $\mathbf{me}$ of $\mathcal{T}_{p,\mathbf{N}}$ also denoted by $\mathcal{T}_{p,\mathbf{N}}.\mathbf{me}$. This is a C_{d+p}^{p}-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 with p-order simplices for the reference hypercube $[0,1]^d$ can be obtained from the function KuhnTriangulation :

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

Let $\boldsymbol{\imath} \in \mathcal{I}$ and $r = \mathcal{H}(\boldsymbol{\imath})$ where $\mathcal{H}$ is defined by (12). Let $l \in [\![1, d!]\!]$ and $k = d!(r-1) + l$. We choose to store the l-th simplex of $\mathcal{K}(\mathrm{H}_{\boldsymbol{\imath}})$ in $\mathbf{me}(:, k)$.

Let $j \in [\![1, C_{d+p}^{p}]\!]$. The j-th nodes of the l-th p-order simplex of $\mathcal{K}(\mathrm{H}_{\boldsymbol{\imath}})$ is stored in $\mathbf{q}(:, \mathbf{me}(j,k))$ and its coordinates are 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 $\mathbf{me}(j,k)$. From (9), we obtain

$$\begin{aligned} \mathbf{me}(j,k) &= \mathcal{G}_p\big(p(\boldsymbol{\imath} + \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l)))\big) \\ &= 1 + \langle \boldsymbol{\beta}, p(\boldsymbol{\imath} + \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l)))\rangle \\ &= 1 + p\,\langle \boldsymbol{\beta}, \boldsymbol{\imath}\rangle + p\,\langle \boldsymbol{\beta}, \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l))\rangle \end{aligned}$$

where $\boldsymbol{\beta}$ is defined in (10) and can be computed as a 1-by-d array by

$$\boldsymbol{\beta} \leftarrow \textsc{CGbeta}(p * \mathbf{N}).$$

Let **iBase** be the 1-by-N_h array given by

$$\mathbf{iBase} \leftarrow 1 + p\left\langle \boldsymbol{\beta}, \text{CartesianGridPoints}(\mathbf{N}-1)\right\rangle.$$

Then the array **me** is given $\forall l \in [\![1, d!]\!]$, $\forall j \in [\![1, d+1]\!]$, $\forall r \in [\![1, N_h]\!]$, by

$$\mathbf{me}(j, d!(r-1)+l) \leftarrow \mathbf{iBase}(r) + p\left\langle \boldsymbol{\beta}, \mathbf{q}_{\mathrm{K}}(:, \mathbf{me}_{\mathrm{K}}(j,l))\right\rangle.$$

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

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

We give in Algorithm **??** the function CGTriangulation which computes the triangulation of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$.

Algorithm 10 Function CGTessSim : computes the tessellation of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$ with p-order simplices

---

Input :
- $\boldsymbol{N}$: array of d positive integers, $\boldsymbol{N}(i) = N_i$.
- p : order, positive integer.

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

```
Function [q, me] ← CGTessSim (N, p)
  q ← CartesianGridPoints(p * N)/p
  β ← CGbeta(p * N)
  iBase ← 1 + p⟨β, CartesianGridPoints(N − 1)⟩
  [q_K, me_K] ← KuhnTriOrder(d, p)
  Idx ← d! * [0 : (N_h − 1)]
  for l ← 1 to d! do
    Idx ← Idx + 1
    for j ← 1 to C^p_{d+p} do
      me(j, Idx) ← iBase + p * β * q_K(:, me_K(j, l))
    end for
  end for
end Function
```

3.5 m-faces tessellations of a cartesian grid with p-order simplices

Let $\mathcal{Q}_{p,\mathbf{N}}$ be the d-dimensional cartesian grid defined in section **??**. As before, we denote by $\mathcal{T}_{p,\mathbf{N}}.\mathbf{q}$ and $\mathcal{T}_{p,\mathbf{N}}.\mathbf{me}$ respectively the nodes and connectivity arrays of the tessellation of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$ with p-order d-simplices obtained from CGTriOrder function and described in Algorithm 10.

Let $m \in [\![0, d[\![$ and $k \in [\![1, E_{m,d}]\!]$ where $E_{m,d}$ is the number of m-faces defined in (1). We want to determine $\mathcal{T}_{p,\boldsymbol{N}}^{m}(k)$, the tessellation 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 2.3. We denote by

- $\mathcal{T}^m_{p,\boldsymbol{N}}(k).\mathbf{q}$, the (local) nodes array
- $\mathcal{T}^m_{p,\boldsymbol{N}}(k).\mathbf{me}$, the (local) connectivity array
- $\mathcal{T}^m_{p,\boldsymbol{N}}(k).\text{toGlobal}$, the global indices such that

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

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

The only difference with the construction of $\mathcal{Q}^m_{p,\mathbf{N}}(k)$ given in section 2.4 is on the $\mathbf{me}^w$ array. For $\mathcal{Q}^m_{p,\mathbf{N}}(k)$, we had

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

whereas for $\mathcal{T}^m_{p,\boldsymbol{N}}(k)$ we must have instead

$$[\mathbf{q}^w, \mathbf{me}^w] \leftarrow \text{CGTessSim}(\boldsymbol{N}(\mathbf{idnc}), p)$$

So only one line changes in the Algorithm 3 to obtain the new one given in Algorithm **??** where the function CGTessSimFaces computes $\mathcal{T}^m_{p,\boldsymbol{N}}(k)$, $\forall k \in 2^{d-m}n_c$.

The line

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

is replaced by

$$[\mathbf{q}^w, \mathbf{me}^w] \leftarrow \text{CGTessSim}(\boldsymbol{N}(\mathbf{idnc}), p)$$

Algorithm 11 Function CGTessSimFaces : computes all m-faces tessellations of the cartesian grid $\mathcal{Q}_{\mathbf{N}}$ with p-order m-simplices

Input :
- $\boldsymbol{N}$: array of d integers, $\boldsymbol{N}(i) = N_i$.
- m : integer, $0 \leqslant m < d$,
- p : positive integer.

Output :
- $\mathcal{T}^m_{p,\boldsymbol{N}}$: array of triangulations of all m-faces comming from the cartesian grid triangulation $\mathcal{T}_{\boldsymbol{N}}$. The length of $\mathcal{T}^m_{p,\boldsymbol{N}}$ is $E_{m,d} = 2^{d-m}\binom{d}{m}$ (number of m-faces).

```
Function T^m_{p,N} ← CGTessSimFaces (N, m, p)
  β ← CGBeta(p ∗ N)
  if m == 0 then
    Q ← Diag(N) ∗ CartesianGridPoints(Ones(1, d))
    for k ← 1 to 2^d do
      T^m_{p,N}(k).q ← Q(:, k)
      T^m_{p,N}(k).me ← 1
      T^m_{p,N}(k).toGlobal ← 1 + ⟨β, Q(:, k)⟩
    end for
  else
    n_c ← binom(d, m)
    L ← Combs([[1, d]], d − m)
    S ← CartesianGridPoints(Ones(1, d − m))
    k ← 1
    for l ← 1 to n_c do
      idc ← L(l, :)
      idnc ← [[1, d]]\idc
      [q^w, me^w] ← CGTriangulation(N(idnc), p)
      n^l_q ← ∏_{s=1}^{m} (N(idnc(s)) + 1)                ▷ or length of q^w
      for r ← 1 to 2^{d−m} do
        T^m_{p,N}(k).q(idnc, :) ← q^w
        T^m_{p,N}(k).q(idc, :) ← (N(idc)^t .∗ S(:, r)) ∗ Ones(1, n^l_q)
        T^m_{p,N}(k).me ← me^w
        T^m_{p,N}(k).toGlobal ← 1 + pβ^t ∗ T^m_{p,N}(k).q
        k ← k + 1
      end for
    end for
  end if
end Function
```

3.6 d-orthotope tessellation with d-simplices

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 2.5 while taking as a starting point the cartesian grid triangulation.

Algorithm 12 Function ORTHTRIANGULATION : regular tessellation with simplices of a d-orthotope

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

Output :
- $\mathbf{q}$: vertices array with d-by-n_{q} reals.
- $\mathbf{me}$: connectivity array with $(d+1)$-by-n_{me} integers.

```
Function [q, me] ← OrthTriangulation (N, a, b)
  [q, me] ← CGTriangulation(N)
  q ← boxMapping(q, a, b, N)
end Function
```

3.7 m-faces tessellations of a d-orthotope with d-simplices

As seen in section 2.5, we only have to apply the function BOXMAPPING to each vertices array $\mathcal{T}^m_{p,\boldsymbol{N}}(k).\mathbf{q}$ corresponding to the k-th m-faces tessellations of the cartesian grid $\mathcal{Q}_{p,\mathbf{N}}$. This is the object of the function ORTHTRIFACES given in Algorithm 13.

Algorithm 13 Function ORTHTRIFACES : computes the conforming tessellations with simplices of all m-faces of the d-orthotope $[a_1, b_1] \times \cdots \times [a_d, b_d]$

Input :
- $\boldsymbol{N}$: array of d integers, $\boldsymbol{N}(i) = N_i$.
- $\boldsymbol{a}, \boldsymbol{b}$: arrays of d reals, $\boldsymbol{a}(i) = a_i$, $\boldsymbol{b}(i) = b_i$
- m : integer, $0 \leqslant m < d$

Output :
- $\mathcal{T}^m_{\boldsymbol{N}}$: array of the tessellations with simplices of all m-faces of the orthotope. Its length is $E_{m,d} = 2^{d-m}\binom{d}{m}$.

```
Function T^m_N ← OrthTriFaces (N, a, b, m)
  T^m_N ← CGTessSimFaces(N, m)
  for k ← 1 to len(T^m_N) do
    T^m_N(k).q ← boxMapping(T^m_N(k).q, a, b, N)
  end for
end Function
```

The codes in Matlab, Octave and Python, referenced as `fc_hypermesh`, can be obtained on

`http://www.math.univ-paris13.fr/~cuvelier/software/`

The Python package `fc_hypermesh` is also available on PyPI [9].

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 $\mathbb{A}$,
TRANSPOSE($\mathbb{A}$)	transpose of $\mathbb{A}$,
SUM($\mathbb{A}$, dim)	sums along dimension dim,
PROD($\mathbb{A}$, dim)	product along 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,
ONES(m, n)	m-by-n array/matrix of ones,
ZEROS(m, n)	m-by-n array/matrix of zeros,
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 $\mathbb{A}$,
RESHAPE($\mathbb{A}, m, n$)	returns the m-by-n array/matrix whose elements are taken columnwise from $\mathbb{A}$.
FIND($\boldsymbol{x}$)	returns a vector of indices of nonzero elements of a vector $\boldsymbol{x}$.

A.2 Combinatorial functions

PERMS($\boldsymbol{V}$)	where $\boldsymbol{V}$ is an array of length n. Returns a $n!$-by-n array containing all permutations of $\boldsymbol{V}$ elements. The lexicographical order is chosen.
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 combinations of n elements taken k at a time. The lexicographical order is chosen.

B Computational costs

All the algorithms of this paper were implemented under Matlab [5], Octave [4] and Python [6]. In each language, the `OrthMesh` class is available which contains a regular and conforming tessellations of a d-orthotope and all its m-faces with high-order orthotopes or simplicial elements ($0 \leqslant m < d$).

In this section, computational costs of the `OrthMesh` constructor are presented for tessellations of the orthotope $[-1;1]^d$ with p-order orthotopes and simplices where $d \in [\![2,5]\!]$ and $p \in [\![1,3]\!]$. In each direction, the orthotope is subdivized in N intervals and so there is $n_\text{q} = (pN+1)^d$ nodes in the associated tessellation. If the orthotope is tessellated with orthotopes, then it contains $n_\text{me} = N^d$ orthotope elements. Otherwise the orthotope is tessallated with simplices and it contains $n_\text{me} = d!N^d$ elements.

The computations were done on a computer with `Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz` processor and 63Go of RAM under `Ubuntu 18.04.3 LTS` (`x86_64`).

B.1 Tessellation with p-order d-orthotopes

In this section, the computational costs of the `OrthMesh` constructor with p-order d-orthotopes are given with $d \in [\![2,5]\!]$ and $p \in [\![1,3]\!]$.

B.1.1 order $p = 1$

Under Matlab, Octave and Python, the computational costs of the `OrthMesh` constructor to tesselate the $[-1,1]^d$ orthotope with 1-order orthotopes are given in tables 6 to 9, respectively for $d = 2$ to $d = 5$.

N	n_q	n_me	Python	Matlab	Octave
1000	1 002 001	1 000 000	0.220 (s)	0.514 (s)	0.328 (s)
2000	4 004 001	4 000 000	0.614 (s)	0.334 (s)	0.767 (s)
3000	9 006 001	9 000 000	1.379 (s)	0.664 (s)	1.546 (s)
4000	16 008 001	16 000 000	2.389 (s)	1.120 (s)	2.547 (s)
5000	25 010 001	25 000 000	3.701 (s)	1.755 (s)	3.918 (s)

Table 6: Tessellation of $[-1,1]^2$ with n_me orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_me	Python	Matlab	Octave
40	68 921	64 000	0.188 (s)	0.500 (s)	0.478 (s)
60	226 981	216 000	0.265 (s)	0.132 (s)	0.587 (s)
80	531 441	512 000	0.322 (s)	0.155 (s)	0.651 (s)
100	1 030 301	1 000 000	0.411 (s)	0.199 (s)	0.749 (s)
120	1 771 561	1 728 000	0.553 (s)	0.301 (s)	0.909 (s)
140	2 803 221	2 744 000	0.738 (s)	0.379 (s)	1.171 (s)
160	4 173 281	4 096 000	1.022 (s)	0.532 (s)	1.487 (s)
180	5 929 741	5 832 000	1.336 (s)	0.721 (s)	1.918 (s)

Table 7: Tessellation of $[-1,1]^3$ with orthotopes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
10	14 641	10 000	0.293 (s)	0.579 (s)	1.106 (s)
20	194 481	160 000	0.394 (s)	0.231 (s)	1.238 (s)
25	456 976	390 625	0.457 (s)	0.258 (s)	1.341 (s)
30	923 521	810 000	0.594 (s)	0.355 (s)	1.515 (s)
35	1 679 616	1 500 625	0.830 (s)	0.493 (s)	1.814 (s)

Table 8: Tessellation of $[-1, 1]^4$ with n_{me} orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
2	243	32	0.531 (s)	0.745 (s)	2.883 (s)
4	3 125	1 024	0.512 (s)	0.374 (s)	2.868 (s)
6	16 807	7 776	0.634 (s)	0.365 (s)	3.024 (s)
8	59 049	32 768	0.584 (s)	0.374 (s)	3.053 (s)
10	161 051	100 000	0.624 (s)	0.440 (s)	3.133 (s)
12	371 293	248 832	0.770 (s)	0.493 (s)	3.659 (s)

Table 9: Tessellation of $[-1, 1]^5$ with n_{me} orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

B.1.2 order $p = 2$

Under Matlab, Octave and Python, the computational costs of the `OrthMesh` constructor to tesselate the $[-1, 1]^d$ orthotope with 2-order orthotopes are given in tables 10 to 13, respectively for $d = 2$ to $d = 5$.

N	n_q	n_{me}	Python	Matlab	Octave
1000	4 004 001	1 000 000	0.388 (s)	0.618 (s)	0.579 (s)
2000	16 008 001	4 000 000	1.426 (s)	0.741 (s)	1.810 (s)
3000	36 012 001	9 000 000	3.199 (s)	1.535 (s)	3.915 (s)
4000	64 016 001	16 000 000	5.523 (s)	2.710 (s)	6.730 (s)
5000	100 020 001	25 000 000	8.700 (s)	4.048 (s)	10.161 (s)

Table 10: Tessellation of $[-1, 1]^2$ with n_{me} 2-order orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
40	531 441	64 000	0.272 (s)	0.538 (s)	0.614 (s)
60	1 771 561	216 000	0.389 (s)	0.228 (s)	0.807 (s)
80	4 173 281	512 000	0.610 (s)	0.372 (s)	1.111 (s)
100	8 120 601	1 000 000	1.041 (s)	0.643 (s)	1.755 (s)
120	13 997 521	1 728 000	1.655 (s)	1.012 (s)	2.618 (s)
140	22 188 041	2 744 000	2.549 (s)	1.517 (s)	3.829 (s)
160	33 076 161	4 096 000	3.719 (s)	2.238 (s)	5.399 (s)
180	47 045 881	5 832 000	5.176 (s)	3.073 (s)	7.456 (s)

Table 11: Tessellation of $[-1, 1]^3$ with n_{me} 2-order orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
10	194 481	10 000	0.360 (s)	0.607 (s)	1.234 (s)
20	2 825 761	160 000	0.688 (s)	0.480 (s)	1.781 (s)
25	6 765 201	390 625	1.319 (s)	0.875 (s)	2.999 (s)
30	13 845 841	810 000	2.332 (s)	1.579 (s)	4.541 (s)
35	25 411 681	1 500 625	3.956 (s)	2.743 (s)	7.088 (s)

Table 12: Tessellation of $[-1, 1]^4$ with 2-order orthotopes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
2	3 125	32	0.514 (s)	0.780 (s)	2.898 (s)
4	59 049	1 024	0.569 (s)	0.396 (s)	3.054 (s)
6	371 293	7 776	0.682 (s)	0.487 (s)	3.658 (s)
8	1 419 857	32 768	0.862 (s)	0.646 (s)	4.165 (s)
10	4 084 101	100 000	1.384 (s)	1.143 (s)	5.055 (s)
12	9 765 625	248 832	2.534 (s)	2.072 (s)	7.092 (s)

Table 13: Tessellation of $[-1, 1]^5$ with n_{me} 2-order orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

B.1.3 order $p = 3$

Under Matlab, Octave and Python, the computational costs of the `OrthMesh` constructor to tesselate the $[-1, 1]^d$ orthotope with 3-order orthotopes are given in tables 14 to 17, respectively for $d = 2$ to $d = 5$.

N	n_q	n_{me}	Python	Matlab	Octave
500	2 253 001	250 000	0.248 (s)	0.509 (s)	0.398 (s)
1000	9 006 001	1 000 000	0.751 (s)	0.430 (s)	1.047 (s)
2000	36 012 001	4 000 000	2.771 (s)	1.367 (s)	3.553 (s)
3000	81 018 001	9 000 000	6.173 (s)	2.960 (s)	7.627 (s)
4000	144 024 001	16 000 000	11.061 (s)	5.108 (s)	13.254 (s)

Table 14: Tessellation of $[-1,1]^2$ with n_{me} 3-order orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
40	1 771 561	64 000	0.365 (s)	0.601 (s)	0.760 (s)
60	5 929 741	216 000	0.740 (s)	0.468 (s)	1.352 (s)
80	13 997 521	512 000	1.440 (s)	0.928 (s)	2.515 (s)
100	27 270 901	1 000 000	2.757 (s)	1.688 (s)	4.271 (s)
120	47 045 881	1 728 000	4.751 (s)	2.800 (s)	6.937 (s)
140	74 618 461	2 744 000	7.340 (s)	4.327 (s)	10.592 (s)

Table 15: Tessellation of $[-1,1]^3$ with n_{me} 3-order orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
5	65 536	625	0.326 (s)	0.597 (s)	1.183 (s)
10	923 521	10 000	0.430 (s)	0.280 (s)	1.424 (s)
20	13 845 841	160 000	1.974 (s)	1.528 (s)	4.420 (s)
25	33 362 176	390 625	4.208 (s)	3.347 (s)	8.257 (s)
30	68 574 961	810 000	8.524 (s)	6.173 (s)	15.318 (s)

Table 16: Tessellation of $[-1,1]^4$ with n_{me} 3-order orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
3	100 000	243	0.604 (s)	0.784 (s)	3.392 (s)
5	1 048 576	3 125	0.778 (s)	0.565 (s)	4.972 (s)
7	5 153 632	16 807	1.468 (s)	1.279 (s)	6.439 (s)
9	17 210 368	59 049	3.515 (s)	3.005 (s)	9.897 (s)
10	28 629 151	100 000	5.451 (s)	4.776 (s)	13.088 (s)

Table 17: Tessellation of $[-1,1]^5$ with n_{me} 3-order orthotopes and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

B.2 Tessellation with p-order d-simplices

In this section, the computational costs of the `OrthMesh` constructor with p-order d-simplices are given with $d \in [\![2, 5]\!]$ and $p \in [\![1, 3]\!]$.

B.2.1 order $p = 1$

Under Matlab, Octave and Python, the computational costs of the `OrthMesh` constructor to tesselate the $[-1, 1]^d$ orthotope with 1-order simplices are given in tables 18 to 21, respectively for $d = 2$ to $d = 5$.

N	n_q	n_{me}	Python	Matlab	Octave
1000	1 002 001	2 000 000	0.272 (s)	0.599 (s)	0.422 (s)
2000	4 004 001	8 000 000	0.770 (s)	0.634 (s)	1.005 (s)
3000	9 006 001	18 000 000	1.636 (s)	1.159 (s)	2.158 (s)
4000	16 008 001	32 000 000	2.804 (s)	2.146 (s)	3.681 (s)
5000	25 010 001	50 000 000	4.358 (s)	3.285 (s)	5.719 (s)

Table 18: Tessellation of $[-1, 1]^2$ with n_{me} simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019aand Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
40	68 921	384 000	0.245 (s)	0.568 (s)	0.482 (s)
60	226 981	1 296 000	0.346 (s)	0.222 (s)	0.639 (s)
80	531 441	3 072 000	0.468 (s)	0.399 (s)	0.901 (s)
100	1 030 301	6 000 000	0.672 (s)	0.659 (s)	1.306 (s)
120	1 771 561	10 368 000	0.989 (s)	0.980 (s)	1.886 (s)
140	2 803 221	16 464 000	1.420 (s)	1.543 (s)	2.736 (s)
160	4 173 281	24 576 000	2.023 (s)	2.161 (s)	3.785 (s)
180	5 929 741	34 992 000	3.341 (s)	3.279 (s)	5.522 (s)

Table 19: Tessellation of $[-1, 1]^3$ with n_{me} simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
10	14 641	240 000	0.372 (s)	0.631 (s)	1.148 (s)
20	194 481	3 840 000	0.854 (s)	0.512 (s)	1.742 (s)
25	456 976	9 375 000	1.600 (s)	1.134 (s)	2.670 (s)
30	923 521	19 440 000	2.915 (s)	2.090 (s)	4.360 (s)
35	1 679 616	36 015 000	5.278 (s)	4.088 (s)	7.075 (s)

Table 20: Tessellation of $[-1, 1]^4$ with n_{me} simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_{q}	n_{me}	Python	Matlab	Octave
2	243	3 840	0.571 (s)	0.817 (s)	3.032 (s)
4	3 125	122 880	0.556 (s)	0.393 (s)	3.031 (s)
6	16 807	933 120	0.777 (s)	0.504 (s)	3.354 (s)
8	59 049	3 932 160	1.110 (s)	0.870 (s)	3.924 (s)
10	161 051	12 000 000	2.307 (s)	1.988 (s)	5.927 (s)
12	371 293	29 859 840	5.122 (s)	5.744 (s)	11.461 (s)

Table 21: Tessellation of $[-1,1]^5$ with n_{me} simplices and n_{q} nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

B.2.2 order $p = 2$

Under Matlab, Octave and Python, the computational costs of the `OrthMesh` constructor to tesselate the $[-1,1]^d$ orthotope with 2-order simplices are given in tables 22 to 25, respectively for $d = 2$ to $d = 5$.

N	n_{q}	n_{me}	Python	Matlab	Octave
1000	4 004 001	2 000 000	0.461 (s)	0.838 (s)	0.691 (s)
2000	16 008 001	8 000 000	1.584 (s)	1.547 (s)	2.202 (s)
3000	36 012 001	18 000 000	3.482 (s)	3.087 (s)	4.953 (s)
4000	64 016 001	32 000 000	6.059 (s)	5.577 (s)	8.619 (s)
5000	100 020 001	50 000 000	9.779 (s)	8.637 (s)	13.367 (s)

Table 22: Tessellation of $[-1,1]^2$ with n_{me} 2-order simplices and n_{q} nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_{q}	n_{me}	Python	Matlab	Octave
40	531 441	384 000	0.323 (s)	0.661 (s)	0.723 (s)
60	1 771 561	1 296 000	0.520 (s)	0.449 (s)	1.071 (s)
80	4 173 281	3 072 000	0.926 (s)	0.861 (s)	1.785 (s)
100	8 120 601	6 000 000	1.710 (s)	1.636 (s)	3.108 (s)
120	13 997 521	10 368 000	2.777 (s)	2.470 (s)	4.896 (s)
140	22 188 041	16 464 000	4.165 (s)	4.139 (s)	7.407 (s)
160	33 076 161	24 576 000	6.150 (s)	6.166 (s)	11.051 (s)
180	47 045 881	34 992 000	9.107 (s)	8.655 (s)	16.412 (s)

Table 23: Tessellation of $[-1,1]^3$ with n_{me} 2-order simplices and n_{q} nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
10	194 481	240 000	0.458 (s)	0.722 (s)	1.332 (s)
20	2 825 761	3 840 000	1.910 (s)	1.525 (s)	3.407 (s)
25	6 765 201	9 375 000	4.185 (s)	3.611 (s)	7.224 (s)
30	13 845 841	19 440 000	8.654 (s)	7.781 (s)	13.525 (s)
35	25 411 681	36 015 000	15.931 (s)	14.635 (s)	24.130 (s)

Table 24: Tessellation of $[-1, 1]^4$ with n_{me} 2-order simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
2	3 125	3 840	0.568 (s)	0.856 (s)	3.188 (s)
4	59 049	122 880	0.711 (s)	0.450 (s)	3.340 (s)
6	371 293	933 120	1.168 (s)	0.825 (s)	4.268 (s)
8	1 419 857	3 932 160	2.515 (s)	2.412 (s)	6.637 (s)
10	4 084 101	12 000 000	6.511 (s)	7.725 (s)	15.118 (s)
12	9 765 625	29 859 840	15.856 (s)	21.417 (s)	34.776 (s)

Table 25: Tessellation of $[-1, 1]^5$ with n_{me} 2-order simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

B.2.3 order $p = 3$

Under Matlab, Octave and Python, the computational costs of the `OrthMesh` constructor to tesselate the $[-1, 1]^d$ orthotope with 3-order simplices are given in tables 26 to 29, respectively for $d = 2$ to $d = 5$.

N	n_q	n_{me}	Python	Matlab	Octave
500	2 253 001	500 000	0.306 (s)	0.643 (s)	0.503 (s)
1000	9 006 001	2 000 000	0.796 (s)	0.789 (s)	1.398 (s)
2000	36 012 001	8 000 000	3.087 (s)	2.891 (s)	5.003 (s)
3000	81 018 001	18 000 000	6.799 (s)	6.304 (s)	11.251 (s)
4000	144 024 001	32 000 000	12.154 (s)	11.017 (s)	19.655 (s)

Table 26: Tessellation of $[-1, 1]^2$ with n_{me} 3-order simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
40	1 771 561	384 000	0.441 (s)	0.759 (s)	0.976 (s)
60	5 929 741	1 296 000	0.977 (s)	0.920 (s)	2.021 (s)
80	13 997 521	3 072 000	2.011 (s)	1.927 (s)	4.069 (s)
100	27 270 901	6 000 000	3.703 (s)	3.699 (s)	7.350 (s)
120	47 045 881	10 368 000	6.285 (s)	6.672 (s)	12.161 (s)
140	74 618 461	16 464 000	9.982 (s)	9.543 (s)	18.968 (s)

Table 27: Tessellation of $[-1,1]^3$ with n_{me} 3-order simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
5	65 536	15 000	0.419 (s)	0.681 (s)	1.312 (s)
10	923 521	240 000	0.598 (s)	0.450 (s)	1.694 (s)
20	13 845 841	3 840 000	4.833 (s)	5.139 (s)	10.122 (s)
25	33 362 176	9 375 000	11.597 (s)	12.917 (s)	23.813 (s)
30	68 574 961	19 440 000	23.132 (s)	25.455 (s)	44.084 (s)

Table 28: Tessellation of $[-1,1]^4$ with n_{me} 3-order simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

N	n_q	n_{me}	Python	Matlab	Octave
3	100 000	29 160	0.790 (s)	0.928 (s)	3.555 (s)
5	1 048 576	375 000	1.374 (s)	1.117 (s)	5.003 (s)
7	5 153 632	2 016 840	3.652 (s)	4.072 (s)	9.261 (s)
9	17 210 368	7 085 880	11.049 (s)	16.948 (s)	25.328 (s)
10	28 629 151	12 000 000	18.305 (s)	28.634 (s)	43.364 (s)

Table 29: Tessellation of $[-1,1]^5$ with n_{me} 3-order simplices and n_q nodes. Computational times in seconds for Python 3.8.1, Matlab 2019a and Octave 5.1.0.

C Some combinatorial reminders

Let $k \in \mathbb{N}$ and $n \in \mathbb{N}$, with $n \geqslant k$. The binomial coefficient is written by C_n^k or $\binom{n}{k}$ and its given by

$$C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

There are some usefull identities:

$$C_n^k = C_n^{n-k}, \tag{28}$$

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k, \qquad \textit{Pascal's formula} \tag{29}$$

$$\sum_{i=k}^{n} C_i^k = C_{n+1}^{k+1}, \qquad \textit{hockey-stick formula} \tag{30}$$

Let $d \in \mathbb{N}^*$ and $m \in \mathbb{N}$. An element $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_d)$ of $\mathbb{N}^d$ is called a **multi-index**.

Lemma 9 *Let $d \in \mathbb{N}^*$ and $m \in \mathbb{N}$. We consider in $\mathbb{N}^d$ the set*

$$B_m = \{\boldsymbol{\alpha} \in \mathbb{N}^d : \ |\boldsymbol{\alpha}| = m\}, \tag{31}$$

where $|\boldsymbol{\alpha}| = \sum_{j=1}^{d} \boldsymbol{\alpha}_j$. Then the cardinality of B_m denoted by $\mathrm{card}(B_m)$ is given by

$$\mathrm{card}(B_m) = C_{d+m-1}^m. \tag{32}$$

Indeed this corresponds to placing m identical balls into d distinct boxes where more than one ball in a box is possible.

Theorem 10 (Theorem 1.8, page 6, [1]) *The number of ways to distribute m identical objects into d distinct boxes, with empty boxes allowed, and multiple occupancy allowed is given by C_{m+d-1}^m.*

Lemma 11 *Let $d \in \mathbb{N}^*$ and $m \in \mathbb{N}$. Let A_m be the subset of $\mathbb{N}^d$ defined by*

$$A_m = \{\boldsymbol{\alpha} \in \mathbb{N}^d : \ |\boldsymbol{\alpha}| \leqslant m\}, \tag{33}$$

Then the cardinality of A_m is

$$\mathrm{card}(A_m) = C_{d+m}^m. \tag{34}$$

Proof: The family of sets $B_0, \dots, B_m$ is a partition of A_m and so we have

$$\begin{aligned} \mathrm{card}(A_m) &= \sum_{j=0}^{m} \mathrm{card}(B_j) = \sum_{j=0}^{m} C_{d+j-1}^j \\ &= C_{d+m-1}^m + \sum_{j=0}^{m-1} C_{d+j-1}^j \end{aligned}$$

From (28), we have $C_{d+j-1}^j = C_{d+j-1}^{d-1}$ and so

$$\mathrm{card}(A_m) = C_{d+m-1}^m + \sum_{j=0}^{m-1} C_{d+j-1}^{d-1}.$$

From the *hockey-stick formula* (30) with $k = d-1$ and $n = d+m-2$ we deduce

$$C_{d+m-1}^d = \sum_{i=d-1}^{d+m-2} C_i^{d-1} = \sum_{j=0}^{m-1} C_{d+j-1}^{d-1}$$

From (28), we have $C_{d+m-1}^d = C_{d+m-1}^{m-1}$. Then we get

$$\mathrm{card}(A_m) = C_{d+m-1}^m + C_{d+m-1}^{m-1}.$$

Finally, using *Pascal's formula* (29) gives (34). □

List of algorithms

List of Tables

References

[1] R.M. Beekman. *An Introduction to Number Theoretic Combinatorics*. Lulu.com, 2017.

[2] Jürgen Bey. Simplicial grid refinement: on freudenthal's algorithm and the optimal number of congruence classes. *Numerische Mathematik*, 85(1):1–29, 2000.

[3] H.S.M. Coxeter. *Regular Polytopes*. Dover books on advanced mathematics. Dover Publications, 1973.

[4] F. Cuvelier. fc_hypermesh: a object-oriented Octave package to mesh any d-orthotopes (hyperrectangle in dimension d) and their m-faces with high order simplices or orthotopes. `http://www.math.univ-paris13.fr/~cuvelier/software/`, 2017. User's Guide.

[5] F. Cuvelier. fc_hypermesh: a object-oriented Matlab toolbox to mesh any d-orthotopes (hyperrectangle in dimension d) and their m-faces with high order simplices or orthotopes. `http://www.math.univ-paris13.fr/~cuvelier/software/`, 2019. User's Guide.

[6] F. Cuvelier. fc_hypermesh: a object-oriented Python package to mesh any d-orthotopes (hyperrectangle in dimension d) and their m-faces with high order simplices or orthotopes. `http://www.math.univ-paris13.fr/~cuvelier/software/`, 2019. User's Guide.

[7] F. Cuvelier and G. Scarella. Vectorized algorithms for regular tessellations of d-orthotopes and their faces. http://www.math.univ-paris13.fr/ cuvelier/docs/reports/HyperMesh/HyperMesh_0.0.4.pdf, 2016.

[8] François Cuvelier and Gilles Scarella. Vectorized algorithms for regular tessellations of d-orthotopes and their faces. *HAL archives ouvertes*, November 2017. preprint.

[9] Python Software Foundation. Pypi, the python package index. `https://pypi.python.org/`, 2003–.

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

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