Proceedings of the Federated Conference on Computer Science and Information Systems pp. 157–163
DOI: 10.15439/2019F16
ISSN 2300-5963 ACSIS, Vol. 18

Integration of Polynomials over n-Dimensional Simplices

Abdenebi ROUIGUEB*, Mohamed MAIZA†, Abderahmane TKOURT† and Imed CHERCHOUR *
*Ecole Militaire Polytechnique
Data Fusion and Analysis Laboratory, Algiers, Algeria
Email: rouigueb.abdenebi@gmail.com
†Ecole Militaire Polytechnique
Modeling and Optimization Techniques Laboratory, Algiers, Algeria,
Email: m_maiza@esi.dz

***Abstract*—Integrating an arbitrary polynomial function f of degree D over a general simplex in dimension n is well-known in the state of the art to be NP-hard when D and n are allowed to vary, but it is time-polynomial when D or n are fixed. This paper presents an efficient algorithm to compute the *exact* value of this integral. The proposed algorithm has a time-polynomial complexity when D or n are fixed, and it requires a reasonable time when the values of D and n are less than 10 using widely available standard calculators such as desktops.**

I. Introduction

THE integral evaluation of polynomial functions over n-dimensional polytopes is essential in many applications. Particularly, it can be used to calculate the probability of a given interval of variables expressed as a polytope and when a polynomial function is used to fit the multivariate probability density function.

In dimension n, efficient integrating formulas may be set up for some types of polytopes having specific or regular shapes such as hyper-cubes and hyper-parallelepipeds. On the other hand, integrating an arbitrary polynomial function f of degree D over an *arbitrary* general convex polytope is a hard task.

For the simple case $f = 1$ (when $D = 0$), it turns into volume computing. Even volume computing of polytopes of varying dimension was proven to be NP-hard [2]. Hence, one can conclude that integrating of polynomials over convex polytopes is NP-hard as well, see [1] for more details.

Usually, integration over a general convex polytope is achieved by partitioning it into a finite set of simplices. Then the whole integral value can be obtained by summing the integration results of f over the resulting simplices. The computational complexity of this approach depends on the: i) triangulation algorithm complexity; ii) number of simplices; iii) integration algorithm complexity of f over a general simplex.

It has been proven that finding the smallest triangulation is NP-hard [3]. Furthermore, the number of simplices seems to increase exponentially with the dimension for all the known triangulation algorithms. Considering these challenges, it would be suitable to use a fast triangulation algorithm, and to focus on the improvement of the integration algorithm over general simplices.

The bad news is that even integrating f over a general simplex Δ is shown to be NP-hard [1], but the good news is that integration can be carried out within an acceptable time for some applications when n and D values are not too high (≤ 10), and time-polynomial for some specific types of f and Δ. For instance, this problem is polynomial time when n or D are fixed [1]. Moreover, useful efficient formulas are also given when f is quadratic and cubic and Δ is affinely symmetric [6], when Δ is the standard (unit) simplex and f is expressed as a product of linear forms, etc.

In [5], interesting integration formulas of arbitrary odd degree function for the n-simplex are derived using combinatorial methods. The key idea consists in employing the known integration formula over the standard simplex [8] by performing an appropriate mapping to a $n+1$ variable space. This method involves $C(n+D+1, D)$ iterations (C: combinations number). In [7], a quite similar idea by finding a suitable transformation to the standard simplex is investigated. Based on these two last studies, in this work, we want to propose a new practical algorithm for integrating a high degree (odd and even) polynomial over a general simplex, where the aim is to further accelerate the original problem transforming to another equivalent integration problem over the standard simplex.

The rest of this paper is organized as follows. The next section presents the problem statement. Then, our main contributions are described in section III. Before conclusion, complexity analysis and some experimental results are given in section IV.

II. Problem statement

Let $\Delta \in \Re^n$ be a general n-simplex and $f \in Q[x_1, ..., x_n]$ be a multivariate polynomial function of degree D with real coefficients. Commonly, f is represented as a sum of M monomial terms, $f = \Sigma_{i=1}^{M} w_i \; x_1^{\alpha_1^{(i)}} ... x_n^{\alpha_n^{(i)}}$, where w_i and $\alpha_j^{(i)} \in N$ ($\alpha_1^{(i)} + ... + \alpha_n^{(i)} \leq D$) correspond respectively to the coefficient and variables powers for each monomial $i = 1..M$.

In this study, we consider the problem of the evaluation of the multiple definite integral of f over Δ, which we denote by $I_{f\Delta}$, and it is given by the formula:

$$I_{f\Delta} = \int_{\Delta} f \; dx_1....dx_n. \tag{1}$$

We aim at providing new practical methods that compute efficiently the *exact* value of $I_{f\Delta}$ when the coefficients of f and the vertices of Δ are rational numbers as discussed in [1], or compute the *numerical* value when floating-point numbers are used instead. In this second case, no approximations are made and the total error in the evaluation is due only to the floating-point numbers representation precision.

III. Integration over polytopes

A. Integration over the standard simplex

The standard simplex, denoted by Δ_s, is the polytope defined by the $n+1$ following vertices: $v_0 = (0,0,...,0)'$, $v_1 = (1,0...,0)'$, $v_2 = (0,1,...,0)'$,..., $v_n = (0,0,...,1)'$. The integral of f over Δ_s is expressed as follows:

$$I_{f\Delta_s} = \int_{x_1=0}^{1} \int_{x_2=0}^{1-x_1} \dots \int_{x_n=0}^{1-x_1\dots-x_{n-1}} f dx_1 \dots dx_n. \tag{2}$$

The integral of one monomial $x^{\langle\alpha\rangle} = x_1^{\alpha_1} x_2^{\alpha_2} ... x_n^{\alpha_n}$ can be computed efficiently using the Stroud formula [8]:

$$I_{x^{\langle\alpha\rangle}\Delta_s} = \frac{\alpha_1!...\alpha_n!}{(n+\alpha_1+...+\alpha_n)!}. \tag{3}$$

Then, $I_f\Delta_s$ can be computed by summing the integral of all f monomial terms, $I_{f\Delta_s} = \Sigma_{i=1}^{M} w_i I_{x_i^{\langle\alpha\rangle}\Delta_s}$.

Note that dynamic programming can accelerate considerably the computation. e.g. factorial terms can be computed and stored just once and used many times.

B. Integration over a general simplex

The majority of simplices obtained by triangulation are not standard. Integrating f over a general simplex Δ is an NP-hard problem of varying dimension and degree [1], but it can be solved within an acceptable time for moderate dimension and degree ($n \leq 10$, $D \leq 10$) by the computing means which are nowadays available. To evaluate $I_{f\Delta}$, a good option would be to find an affine change of variables from the original space $[x_1, ..., x_n]$ to a new space $[y_1, ..., y_n]$ such that

$$I_{\Delta} = \int_{\Delta} f \; dx_1....dx_n = \int_{\Delta_s} h \; dy_1....dy_n, \tag{4}$$

where h is a polynomial function with the same degree as f, Δ_s is the standard simplex. After that, one can utilize formula 3. It is particularly noteworthy that for an non-empty volume simplex Δ, this change of variables is always possible. We will show how to determine h terms in the rest of this section.

Let the vertices of Δ be $v_0 = (v_1^0, ..., v_n^0)'$, $v_1 = (v_1^1, ..., v_n^1)'$, ..., $v_n = (v_1^n, ..., v_n^n)'$. We propose to find an affine transformation $T : \Delta_s \to \Delta$ that maps the standard simplex Δ_s in the y space to the general simplex Δ in the x space, as shown in Fig 1 example. Thus, T maps each vertex of Δ_s to a distinct vertex of Δ, the order of vertices is not important. Formally, T maps a given point $\mathbf{y}$ to the point $\mathbf{x}$ given by $\mathbf{x} = A\mathbf{y} + B$ where A is an invertible $n \times n$ matrix that defines the combination effect of rotation, scaling and shearing, and B is a translation vector.

The correspondence between vertices of Δ_s and vertices of Δ yields the following linear system $V = A \times V_s + B$, which is given by

$$\begin{pmatrix} v_1^0 \, v_1^1 \dots v_1^n \\ \vdots \; \vdots \; \ddots \; \vdots \\ v_n^0 \, v_n^1 \cdots v_n^n \end{pmatrix} = \begin{pmatrix} A_{1,1} \dots A_{1,n} \\ \vdots \quad \ddots \quad \vdots \\ A_{n,1} \cdots A_{n,n} \end{pmatrix} \times \begin{pmatrix} 0 1 \cdots 0 \\ \vdots \vdots \ddots \vdots \\ 0 0 \cdots 1 \end{pmatrix} + \begin{pmatrix} B_1 \\ \vdots \\ B_n \end{pmatrix} \tag{5}$$

, where columns of matrices V and V_s represent the vertices coordinates of Δ and Δ_s, respectively. The unique solution of the equations system (5) is:

$$B = \left(v_1^0, \dots, v_n^0\right)' \; and \;\; A_{i,j} = v_i^j - B_i \;\; (\forall \; i,j \in 1...n), \tag{6}$$

which can be computed in linear time. Affine transformations are known to preserve betweenness, $\mathbf{x} = T(\mathbf{y})$ lies inside Δ if and only if $\mathbf{y}$ is inside Δ_s. To compute $I_{f\Delta}$, we propose to perform the change of variables $\mathbf{x} = A \times \mathbf{y} + B$ where the i^{th} component of $\mathbf{x}$ is

$$\begin{aligned} x_i &= A_{i,1} \times y_1 + \ldots + A_{i,n} \times y_n + B_i \\ &= \Sigma_{j=1}^{n} A_{i,j} \times y_j + B_i. \end{aligned} \tag{7}$$

Hence, we have:

$$\begin{aligned} I_{f\Delta} &= \int_{\Delta} f(x_1, ..., x_n) dx_1 ... dx_n \\ &= \int_{\Delta_s} f(\Sigma_{j=1}^{n} A_{1,j} \times y_j + B_1, ..., \Sigma_{j=1}^{n} A_{n,j} \times y_j + B_n) \\ &\quad |det(Jac_{T(y_1,...,y_n)})| \;\; dy_1 ... dy_n, \end{aligned} \tag{8}$$

where $|det(Jac_{T(y_1,...,y_n))})| = |det(A)|$ is the absolute value of the determinant of the Jacobian of T .

For the simple case of a single monomial term, $f = x^{\langle\alpha\rangle} = x_1^{\alpha_1} ... x_n^{\alpha_n}$ we have:

$$\begin{aligned} I_{x^{\langle\alpha\rangle}\Delta} = \int_{\Delta_u} &(\Sigma_{j=1}^{n} A_{1,j} y_j + B_1)^{\alpha_1} ... (\Sigma_{j=1}^{n} A_{n,j} y_j + B_n)^{\alpha_n} \\ &|det(A)| \;\; dy_1 ... dy_n. \end{aligned} \tag{9}$$

Let $P_{x_i} = \Sigma_{j=1}^{n} A_{i,j} y_j + B_i$ be the dense polynomial of degree one in the n-dimensional space of y variables. The desired polynomial function h is then equal to $|det(A)| f(P_{x_i}, ..., P_{x_n})$.

Therefore, determining h coefficients can be carried out according to the expression of f by performing a series of additions and multiplications of dense intermediate polynomials of degree $d \leq D$ (D=degree of f).

For example, as illustrated in Fig. 1, to integrate $f = x_1 x_2^3 + x_1^2 x_2 + x_2^2 + 2x_1 x_2 + x_1 + 2$ over the simplex Δ defined by

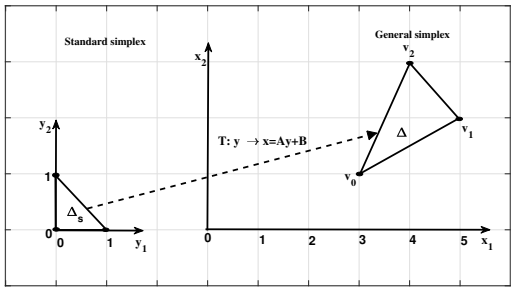


Fig. 1. 2-Dimensional affine transformation example.

the vertices $V = \{v_0 = (3,1)', v_1 = (5,2)', v_2 = (4,3)'\}$, by using equations (6,7), we obtain:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},\ B = \begin{bmatrix} 3 \\ 1 \end{bmatrix},\ \begin{matrix} p_{x_1} = 2y_1 + y_2 + 3 \\ p_{x_2} = y_1 + 2y_2 + 1 \end{matrix}$$

Determining h depends indeed on the regular expression used to denote f; in other words, it depends on the position of brackets and operators '*', '+' in f expression. For this example, h can be built efficiently according to the scheme shown in Fig. 2:(a) when f is expressed as a sum of monomials, or according the scheme of Fig. 2:(b) when f is factorized as follows $f = (x_1x_2+1)*(x_2^2+x_1+2)$. As you can see in Fig. 2:(a), terms x_1x_2 and x_2^2 are computed just once, and used to compute the four first terms of $f : x_1x_2^3, x_1^2x_2, x_2^2, 2x_1x_2$

For this approach, integration efficiency depends conjointly on : i) the factorization tree of f and ii) the complexity of addition and multiplication of intermediate polynomials. Finding the optimal factorization tree for an arbitrary dense polynomial is a difficult combinational problem because the search space of all possible factorization schemes is large; e.g. using straight-line-programs [1] may be useful. In this study, our main contribution is focused on the second point; on the proposition of new methods to accelerate furthermore addition and particularly multiplication of intermediate polynomials involved during the construction of h.

C. Discussion

In the proposed method, computing $I_{f\Delta}$ is achieved by performing a suitable change of variables from x to y space such that equation (4) holds. And then, integrating the monomial terms of h over the standard simplex Δ_s using formula (3) in a polynomial time. h terms are determined by accomplishing a long sequence of addition and multiplication of *dense* intermediate polynomials of degree up to D. Almost all obtained polynomials are dense because the matrix transformation A is often not sparse for general simplices resulting from the triangulation process. The big part of the computational complexity is due to multiplication rather than addition of intermediate polynomials. Indeed, the multiplication $P = P_1 \times P_2$ of two polynomials represented as a sum of monomial terms can be carried out in two steps: i) distribution and monomial multiplication (a Cartesian product) and ii) simplification of

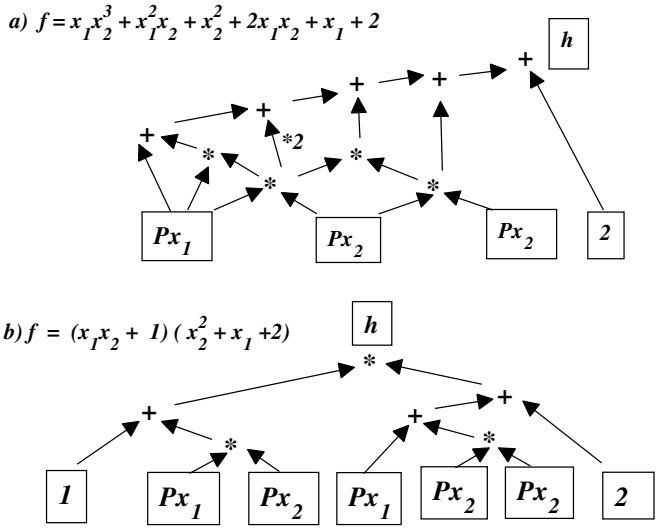


Fig. 2. Example of variables change according to 2 different factorizations.

result terms having the same degree. It should be emphasized that the simplification (step ii) requires more of computational complexity than the distribution and multiplication (step i). Let k_1,k_2 be sizes of P_1, P_2, respectively. Consequently, we need $k_1 \times k_2$ elementary operations of multiplication in step i), but we need $k_1 \times k_2 \times C$ operations where C is the cost of the simplification of a given monomial term m. C is equal to the cost of search plus the cost of insertion of term m within the structure of P.

Therefore, it is interesting to improve monomial simplification by the proposition of suitable polynomial representation and efficient algorithms for multiplication.

D. Proposition 1: accelerating variables change

The physical memory can in fact be seen as a continuous sequence of cells and all intermediate polynomials terms are stored of course in that space. During the variables change process, we need to find efficiently many times the location of a monomial given their variable powers, hence the order of monomials in memory is important.

As all the polynomials obtained are dense, we propose to represent a given polynomial P as a triplet (n,d,W) where n is the dimension, d is its degree ($d \leq D$), and W is a vector of floats containing the monomial coefficients of P. The size of W is then equal to $(n+d)!/(n!d!)$. It is a good idea to save only the monomial coefficients into a compact structure according to a *particular order* and to not save variables powers (α vectors). The desired order must allow a fast mapping in both directions between variables powers vector and the monomial position in the structure of P. Consequently, space complexity will be reduced because variables powers (α vectors) are not saved.

The memory position (index) of the coefficient of a given monomial $x^{\langle\alpha\rangle} = x_1^{\alpha_1}...x_n^{\alpha_n}$ must be calculated efficiently based on the values of n, d,α and the chosen order. To this end, we consider that $wx_1^{\alpha_1}...x_n^{\alpha_n}$ is ordered before (on the

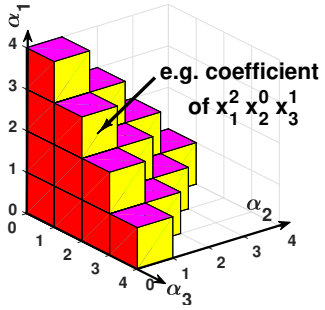


Fig. 3. VOIS structure example (D=3, n=3).

Procedure 1 Mapping powers to index($Pow2Ind$)

Input: α monomial powers vector
degree d and dimension n of the polynomial
Output: Ind (the corresponding index memory of $x^{\langle\alpha\rangle}$)
$Ind \leftarrow 0$;
$j \leftarrow n-1$; $i \leftarrow d$; start cell in Pascal square
$k \leftarrow 0$; variable index in vector α
while $\alpha \neq (0,...,0)$ **do**

if $\alpha[k] > 0$ **then**
$Ind \leftarrow Ind + Pascal[j,i]$; increment Ind
$i \leftarrow i-1$; shift left in Pascal
$\alpha[k] \leftarrow \alpha[k]-1$;
else
$j \leftarrow j-1$; shift up in Pascal
$k \leftarrow k+1$; shift right in α
end if
end while

left) of $w' x_1^{\alpha'_1}...x_n^{\alpha'_n}$ if $[\alpha_1...\alpha_n] < [\alpha'_1...\alpha'_n]$, which can be evaluated recursively as follows:

$$[\alpha_i...\alpha_n] < [\alpha'_i...\alpha'_n] if \begin{cases} \alpha_i < \alpha'_i \ or \ (\alpha_i = \alpha'_i \ and \\ [\alpha_{i+1}...\alpha_n] < [\alpha'_{i+1},...,\alpha'_n]) \end{cases}$$

We propose to represent the coefficients W of a dense polynomial of degree d and n variables as a Virtual Ordering Integer Simplex which we refer to as VOIS structure in this paper. The coefficient of the monomial $x_1^{\alpha_1}...x_n^{\alpha_n}$ is stored in the cell $(\alpha_1,...,\alpha_n)$ of the VOIS structure. Fig 3 illustrates an example for d=3 and n=3. The coefficients of P are stored in the physical memory as a flat vector according to the VOIS order as follows: $[w_0, w_1, ...w_k]$ where $k = (n+d)!/n! * d!)$ is the size of P, as illustrated in Fig 4.

For polynomial multiplication, we need both to handle monomials powers $(\alpha_1,...,\alpha_n)$, and to access directly to their memory location. The principal aim of the VOIS structure is to speed up conversion between monomial powers vectors and theirs corresponding memory indices.

The proposed functions that map powers vector to the corresponding memory index ($pow2ind$) and the inverse mapping ($ind2pow$) are displayed in Algorithms 1 and 2, respectively.

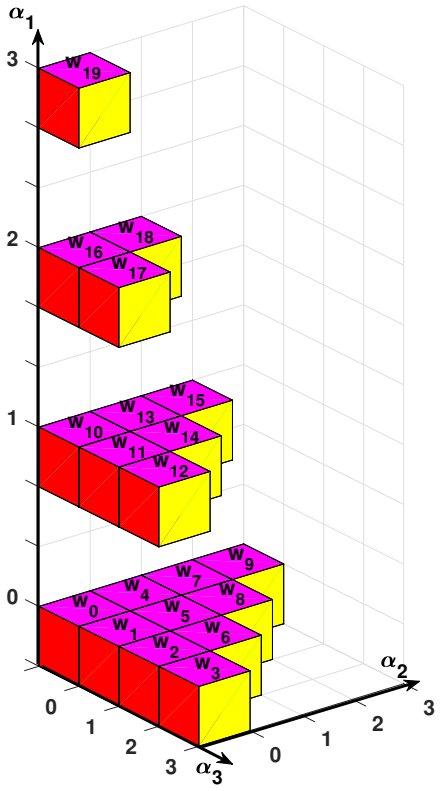
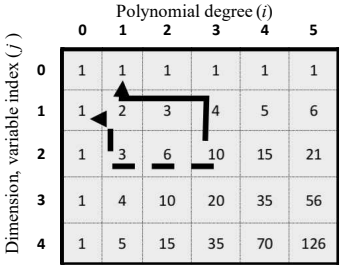


Fig. 4. Memory indices of monomials coefficients (example: D=3, n=3).

Procedure 2 Mapping index to powers ($Ind2Pow$)

Input: Ind monomial powers vector
degree d and dimension n of the polynomial
Output: α monomial powers vector
$\alpha \leftarrow [0,...,0]$;
$j \leftarrow n-1$; $i \leftarrow d$;
$k \leftarrow 0$;
while $Ind \neq 0$ **do**

if $Ind \geq Pascal[j,i]$ **then**
$\alpha[k] \leftarrow \alpha[k]+1$;
$Ind \leftarrow Ind - Pascal[j,i]$; decrement Ind
$i \leftarrow i-1$; shift left in Pascal
else
$j \leftarrow j-1$; shift up in Pascal
$k \leftarrow k+1$; shift right in α
end if
end while



	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1	2	3	4	5	6
2	1	3	6	10	15	21
3	1	4	10	20	35	56
4	1	5	15	35	70	126

a) The Pascal square

Ind	*j*	*i*	*k*	α	Iteration
18	2	3	1	[0,0,0]	Initial.
08	2	2	1	[1,0,0]	(1)
02	2	1	1	[2,0,0]	(2)
02	1	1	2	[2,0,0]	(3)
00	1	0	2	**[2,1,0]**	(4)

b) *IndToPow()* trace example

Ind	*j*	*i*	*k*	α	Iteration
00	2	3	0	**[0,2,1]**	Initial.
00	1	3	1	[0,2,1]	(1)
04	1	2	1	[0,1,1]	(2)
07	1	1	1	[0,0,1]	(3)
07	0	1	1	[0,0,1]	(4)
08	0	0	1	[0,0,0]	(5)

c) *PowToInd() trace example*

Fig. 5. Conversin examples between powers and memory indices.

Fig. 5 shows an example of conversion between powers and memory indices. The first row and column are set to one in the Pascal square, and the remaining cells are filled recursively as follows $Pascal(j,i) = Pascal(j-1,i) + Pascal(j,i-1)$. For both algorithms of conversion, we start from the cell $Pascal(n-1,d)$. The run trace of $Ind2Pow$ (resp. $Pow2Ind$) function from $Ind = 18$ (resp. $\alpha = [0,2,1]$) to $\alpha = [2,1,0]$ (resp. 8) is depicted on the sub-figure 5-b) (resp. sub-figure 5-c). The time complexity of conversion is linear, at most $(n+d)$ iterations are required for conversion. These two conversion algorithms are conceived in order to speed up polynomial addition and especially multiplication as shown in Algorithm 3 (note that addition is more efficient since only two *not nested* for-loops are required to add terms of P_1 to P and then terms of P_2 to P).

E. Proposition 2: optimization of time at the expense of memory complexity

When f is represented as a sum of terms, performing the change of variables of all monomials separately, one by one, is certainly a non-efficient way since several monomials differ slightly from each other. On the other hand, finding the optimal factorization tree is still an open research area.

As a compromise, in this study, we apply a factorization scheme using the Dynamic Programming principle. In this sense, monomials with low degrees are transformed first. To compute $T(x^{\langle\alpha\rangle})$, we need just to multiply a monomial of degree $\alpha-1$ which is supposed already computed by the one-degree polynomial of a given variable P_{x_i}. For example, the twenty monomials terms of a polynomial with $n = 3$ and $d = 3$ are computed according to the order shown in Figure 6. Implicitly, the construction of h is achieved using a straight-line-program [1]. Indeed, for this example, the computation

Procedure 3 Polynomial multiplication ($PolMul$)

Input: P_1, P_2 two polynomials with degrees d_1, d_2, respectively, and the same dimension n
Output: P //the product $P = P_1 \times P_2$
$sz_1 \leftarrow (n+d_1)!/(n!d_1!)$; //size of P_1
$sz_2 \leftarrow (n+d_2)!/(n!d_2!)$; //size of P_2
$d \leftarrow d_1 + d_2$; $sz \leftarrow sz \leftarrow (n+d)!/(n!d!)$ // degree and size of P
$P \leftarrow ZerosPolynomial$; // a vector of sz zeros.
for $i_1 = 0$ **to** sz_1 **do**
  $\alpha_1 \leftarrow Ind2Pow(i_1, d_1, n)$
  for $i_2 = 0$ **to** sz_2 **do**
    $\alpha_2 \leftarrow Ind2Pow(i_2, d_2, n)$
    $\alpha \leftarrow \alpha_1 + \alpha_2$ // add the powers vector α_1 and α_2
    $i \leftarrow Pow2Ind(\alpha, d, n)$
    $P[i] \leftarrow P[i] + P_1[i_1] \times P_2[i_2]$;
  end for
end for

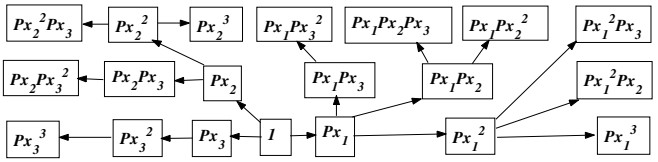


Fig. 6. Factorization tree for the change of variables.

order is as follows: $w_0+x_3(w_1+x_3(w_2+x_3(w_3)))+x_2(w_4+x_3(w_5+x_3(w_6))+x_2(w_7+x_2(w_8)))+x_1(...)$, where w_i are real coefficients.

During the whole variable transforming process for the adopted factorization, we have to multiply several times monomials with degree $d < D$ with a polynomial of degree one. Instead of calling the functions, $Ind2Pow$ and $Pow2Ind$, we propose to save the monomial indices of the multiplication output polynomial P in a vector.

See algorithm 4, if we try to multiply a polynomial P_1 with degree d_1 with a polynomial P_2 with $d_2 = 1$ for the first time then we save the sequence of i values (line $Ix_{d_1}[m] \leftarrow i$; $m \leftarrow m+1$; in the if-clause), else we reuse the indices already saved when polynomials with the same degrees are multiplied (line $i \leftarrow Ix_{d_1}[m]$; $m \leftarrow m+1$; in the else-clause).

IV. EXPERIMENTATION, DISCUSSION AND PERSPECTIVES

The key contribution of this work consists in the proposition of the VOIS structure and in the development of the corresponding mapping functions $Ind2Pow$ and $Pow2Ind$. This solution has sped up mostly polynomial multiplication and consequently the whole integration process.

At each iteration in the second for-loop of algorithm 3, the coefficient product $P_1[i_1]P_2[i_2]$ is computed and added to the monomial term of degree $\alpha = \alpha_1 + \alpha_2$ in P structure. Table I presents a comparison of our method with other possible alternatives in terms of time and space complexity needed to find, insert or update the monomial $x^{\langle\alpha\rangle}$ in the

TABLE I
DEFINING CHARACTERISTICS OF FIVE EARLY DIGITAL COMPUTERS

#	Method	Search time	Insertion time	Space complexity
1	Sorted linked list	O(X) //check all elements	O(1)	X *(n+1) //Coefficient and powers (α)
2	Sorted dynamic array	O(Log(X)) //dichotomic search	O(X) //shift elements right	X *(n+1) //Coefficient and powers (α)
3	Static pre-allocated array	O(Log(X)) //dichotomic search	O(1)	X *(n+1) //Coefficient and powers (α)
4	**VOIS without indices saving**	**O(n+d)** //Pow2Ind	**O(1)**	**X** //only coefficients
5	**VOIS with indices saving**	**O(1)** //direct access	**O(1)**	**X + [((n+1)d)/(d+n)]*X** //coefficients + indices trace

where $X = (n + d)!/(n!d!)$ is equal to the number of all polynomial terms of $P(n, d)$.

Procedure 4 Polynomial multiplication with indices saving($PolMul$)

Input: P_1, P_2 two polynomials with degrees d_1, $d_2 = 1$, in dim. n
Output: $P = P_1 \times P_2$
$P \leftarrow ZerosPolynomial$; $m \leftarrow 0$;
if Ix_{d_1} is empty **then**
  for $i_1 = 1$ **to** $Size$ of P_1 **do**
    $\alpha_1 \leftarrow Ind2Pow(i_1, d_1, n)$
    for $i_2 = 1$ **to** $Size$ of P_2 **do**
      $\alpha_2 \leftarrow Ind2Pow(i_2, d_2, n)$; $\alpha \leftarrow \alpha_1 + \alpha_2$;
      $i \leftarrow Pow2Ind(\alpha, d, n)$
      $P[i] \leftarrow P[i] + P_1[i_1] \times P_2[i_2]$;
      $Ix_{d_1}[m] \leftarrow i$; $m \leftarrow m + 1$; // save i sequence
    end for
  end for
else
  for $i_1 = 1$ **to** $Size$ of P_1 **do**
    for $i_2 = 1$ **to** $Size$ of P_2 **do**
      $i \leftarrow Ix_{d_1}[m]$; $m \leftarrow m + 1$; //direct access
      $P[i] \leftarrow P[i] + P_1[i_1] \times P_2[i_2]$;
    end for
  end for
end if

output polynomial P of multiplication. One can see that our proposition (methods 4 and 5) overcomes incontestably the three first classical methods in terms of the trade-off between time and space complexity.

Another finding was that the global complexity of our algorithm for the integration over a general simplex is polynomial in time when the degree D or the dimension n are fixed. The proof of this result is as follows. In our proposition, all polynomial multiplications are carried out between a polynomial p_1 ($d1 \leq D$) with a one degree polynomial P_2 ($d_2 = 1$).

For a fixed degree D, the complexity of multiplication $P_1 \times P_2$ is: CompMul= size of P_1 * size of P_2 * O(n+d), where $O(n + d)$ is the search-insertion time complexity of a one monomial term (see row 4 in Table I). Hence, we have:

$$CompMul = (d_1 + n)!/(d_1!n!) * (n + 1) * O(n + d)$$
$$\leq (D + n)!/(D!n!) * (n + 1) * O(n + D)$$
$$\leq (D + n)^D/(D!) * (n + 1) * O(n + D)$$
$$\leq O(n^D) * O(n) * O(n + D)$$
$$\leq O(n^{D+2})$$

, which is polynomial with a varying n. The complexity of polynomial addition is also polynomial in time; it is lesser than the multiplication complexity. According to the used polynomial factorization, we need at most to compute $(D+n)!/(D!n!)$ multiplications and $(D+n)!/(D!n!)$ additions which is polynomial for a fixed D. The total complexity of integrating is then polynomial because the composition of two polynomial functions is also polynomial.

The permutation between D and n in the given proof allows us to conclude that the complexity is polynomial for a fixed number of variables. This result agrees with results given recently in [1].

Table II shows the measured integration time using the proposed VOIS method for some examples of polynomials and simplices generated randomly when varying D and n. Experiment are carried out on a standard computer.The best integration results for less than a second are highlighted in bold and the worst results exceeding 10 hours are not displayed. One can notice that time-complexity increases very fast as an exponential when augmenting together D and n, but it increases with a lower rate when either D or n are low. We also notice that saving indices allows reducing time by a factor nearby 5.

V. CONCLUSION

In this paper, we have proposed an integration method of high dimensional polynomial functions with high degree over a general simplex by performing an affine change of variables to the standard simplex where efficient formulas are already known.

The suggested variables change turns into making addition and multiplication operations over intermediate polynomials many times. To this end, we have proposed a compact data structure for polynomial representation, that we have called VOIS, in order to optimize the different operations involved during the polynomial transforming from the general simplex integration problem to an equivalent standard simplex integration problem.

For a fixed degree (and a varying dimension) or for a fixed dimension (and a varying degree) the integration computational complexity of our algorithm over a general simplex

TABLE II
TIME OF INTEGRATION OF A DENSE POLYNOMIAL OVER A GENERAL SIMPLEX

Degree	Method1: VOIS with indices saving (sec)								Method2: VOIS without indices saving (sec)							
	$n=2$	$n=4$	$n=6$	$n=7$	$n=8$	$n=9$	$n=12$	$n=15$	$n=2$	$n=4$	$n=6$	$n=7$	$n=8$	$n=9$	$n=12$	$n=15$
$D=2$	**0**	**0**	**0.001**	**0.004**	**0.027**	**0.227**	287.81	-	**0**	**0**	**0.001**	**0.004**	**0.026**	**0.225**	291.71	-
$D=4$	**0**	**0.001**	**0.005**	**0.015**	**0.054**	**0.258**	297.77	-	**0**	**0.015**	**0.027**	**0.045**	**0.126**	**0.447**	293.87	-
$D=6$	**0**	**0.005**	**0.095**	**0.348**	1.061	3.245	333.65	-	**0**	**0.013**	**0.348**	1.339	4.509	13.64	520.79	-
$D=7$	**0**	**0.011**	**0.340**	1.334	4.446	16.25	666.91	-	**0**	**0.033**	1.278	5.648	21.69	74.25	2133.6	-
$D=8$	**0**	**0.023**	**0.966**	4.607	19.19	74.17	2889.1	-	**0**	**0.080**	4.199	4.199	93.76	363.2	-	-
$D=9$	**0**	**0.050**	2.67	14.66	71.50	324.45	-	-	**0.001**	**0.178**	12.23	72.281	361.4	1584.1	-	-
$D=12$	**0.001**	**0.318**	39.12	322.59	10534	-	-	-	**0.002**	1.248	185.2	1534.5	10835	-	-	-
$D=15$	**0.002**	1.441	365.66	18928	-	-	-	-	**0.004**	6.135	1727.3	19647	-	-	-	-
$D=20$	**0.005**	11.25	34579	-	-	-	-	-	**0.012**	50.09	34415	-	-	-	-	-

In experiments, only one core of the Processor Intel i7 3.7 GHz is used.

is polynomial. However, when varying both dimension and degree, the complexity in experiments that we have carried out seems to increase exponentially. In this last case, we recall that integration of a general polynomial function over a general simplex is shown to be NP hard [1].

A second aspect not fully examined in this work relates to the representation of the input polynomial; more precisely to the regular expression used to represent the polynomial that we want to integrate. For instance, we have found in some experiments that representing the polynomial function as a product of lower-degree polynomials, if it is possible, is more efficient than using a dense form expressed as a sum of monomial terms. We recommend orienting future works on the problem of determining the optimal factorization tree of polynomial functions in relation to integration performances.

REFERENCES

[1] V. Baldoni and N. Berline and J. A. De Loera and M. Köppe and M. Vergne, "How to Integrate a Polynomial over a Simplex," *Math. Comput. J.,* vol. 80, 2011, pp. 297–325.

[2] M. E. Dyer and A. M. Frieze, "Frieze, On the complexity of computing the volume of a polyhedron," *SIAM J. Comput.* vol. 17, no. 5, 1961, pp. 967-974.

[3] J. A. De Loera, J. Rambau, and F. Santos, *Triangulations: Structures and algorithms,* Book manuscript, 2008.

[4] A. H. Stroud, *Approximate Calculation of Multiple Integrals.* Prentice-Hall, Englewood Cliffs, NJ, 1971.

[5] A. Grundmann and H. M. Moller, "Invariant Integration Formulas for the n-Simplex by Combinatorial Methods," *SIAM J. Numer. Anal.* vol. 17, no. 5, 1961, pp. 282-290.

[6] P. C. Hammer and A. H. Stroud, "Numerical integration over simplexes," *Math Tables other Aids Comput.* vol. 10, 1956, pp. 137-139.

[7] F. Bernardini, "Integration of polynomials over n-dimensional polyhedra," *Computer-Aided Design* vol. 23, no. 11, 1991, pp. 51-58.

[8] A. H. Stroud, *Approximate Calculation of Multiple Integrals,* Prentice-Hall, Englewood Cliffs, NJ, 1971.
a product of linear forms