KARATSUBA AND TOOM-COOK METHODS FOR MULTIVARIATE POLYNOMIALS

Marco Bodrato and Alberto Zanoni

Abstract. Karatsuba and Toom-Cook are well-known methods used to efficiently multiply univariate polynomials and long integers. For multivariate polynomials, asymptotically good approaches like Kronecker's trick combined with FFT become truly effective only when the degree is above some threshold. In this paper we analyze Karatsuba and some of Toom-Cook methods for multivariate polynomials, considering density in a different way with respect to Kronecker, and present some algorithms for fast multivariate polynomial multiplication in practical cases, when degrees are not huge. A fast sparse polynomial multiplication algorithm is also proposed.

2000 *Mathematics Subject Classification*: 11A05, 11A25, 11K65, 11Y70

1. Introduction

Subquadratic multiplication methods for univariate polynomials were first introduced in [8], [11], [5] by applying an evaluate-multiply-interpolate (EMI) scheme. For brevity, in this paper we indicate the classical Toom-Cook k-way splitting method with Toom-k (Karatsuba corresponds to Toom-2). Besides the schoolbook method, multivariate polynomial multiplication was treated e.g. by Moenck [9] by either:

- recursing on every variable with an approach having as basic case the classical one.
- using the Kronecker trick to reduce to the univariate case, and then applying univariate algorithms (typically, FFT).

Canny et al. [3] describe an algorithm, working in characteristic 0, based on the generic EMI schema, whose complexity is expressed in terms of the number of product coefficients (called T in the paper). By considering interpolation points $(p_1^{e_1}, \ldots, p_n^{e_n})$ with p_i different primes, $0 \leq e_i < T$ and the complexity of involved matrix operations, they obtain $O(M(T) \log T)$, where $M(T)$ is univariate polynomial multiplication complexity.
Pan [10], by using the EMI schema, is able to lower Kronecker+FFT method complexity, while Cantor and Kaltofen [4] provide an algorithm with $O(N \log N)$-multiplications and $O(N \log N \log\log N)$-sums over completely generic algebras. We also point out the studies made by Fateman on representing polynomials as long integers [7] and its testing [6] for sparse polynomials multiplication, in which different CAS are compared.
The authors studied in [2] the problem concerning optimal inversion sequences for interpolation phase of classical univariate Toom methods, also for unbalanced cases. To the best of our knowledge, it seems that multivariate Toom methods were not considered deeply, may be due to the presence of better asymptotic methods. In this work we describe in full detail Karatsuba, Toom-2.5 and Toom-3 method for multivariate polynomials, and moreover provide as a byproduct an algorithm for fast sparse polynomial multiplication.

2. Multivariate polynomial multiplication

We introduce some notation and terminology concerning multivariate polynomials. We use multindexes notation: for n-variate polynomials, let $X = (x_1, \ldots, x_n)$ be a set of variables, $\mathbb{N}^n$ the set of multindexes, and $\mathcal{T} = \{X^\alpha \mid \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n\}$ the set of terms. We will also indicate generic terms with $T_i, T_j \in \mathcal{T}$, when it is not necessary to put in evidence the multiexponents.
Let $\mathbf{R}$ be a commutative ring and $\overline{X} = X \cup \{x_0\}$, where x_0 is a new variable. We indicate the homogenization of a polynomial $p(X) \in \mathbf{R}[X]$ with $\overline{p}$ or $p(\overline{X}) \in \mathbf{R}[\overline{X}]$.

2.1 *Approaches based on different density definitions*

Definition 1 *Let* $p(X) = \sum_{\alpha \in s(p)} p_\alpha X^\alpha \in \mathbf{R}[X]$, *with* $p_\alpha \neq 0$ *for all* $\alpha \in s(p) \subset \mathbb{N}^n$. *We call the index set* $s(p)$ *its* support, *and* $^{\#}p$ *the support cardinality.*

The support $s(p)$ can be identified with the set of terms of p. We may consider $^{\#}p$ as the cardinality of the data set which is necessary to represent p in the terms basis.

Definition 2 *Let p be as above. We call* total degree *the number* $\deg_1(p) = \deg(p) = \max_{\alpha \in s(p)} \left\{ \sum_{i=1}^{n} \alpha_i \right\}$ *and* max degree *the number* $\deg_\infty(p) = \max_{\alpha \in s(p)} \{\alpha_i\}$.

A polynomial p can be represented in different ways, which we define below: as a triangle, as a square, or sparsely.

Definition 3 *We say p has a* **square** *representation (is* square*) when is considered as* $p(X) = \sum_{|\alpha| \le \deg_\infty(p)} p_\alpha X^\alpha$ *so that all p_α with $|\alpha| \le \deg_\infty(p)$ (even 0 ones) are memorized.*

Definition 4 *We say p has a* **triangular** *representation (is* triangular*) when is considered as* $p(X) = \sum_{|\alpha| \le \deg(p)} p_\alpha X^\alpha$ *so that all p_α with $|\alpha| \le \deg(p)$ (even 0 ones) are memorized.*

Definition (1) refers to polynomials with sparse representation, while definitions (3) and (4) to two different dense representations. Figure 1 shows what happens in the bidimensional case for $\deg_1(p) = \deg_\infty(p) = 2$.

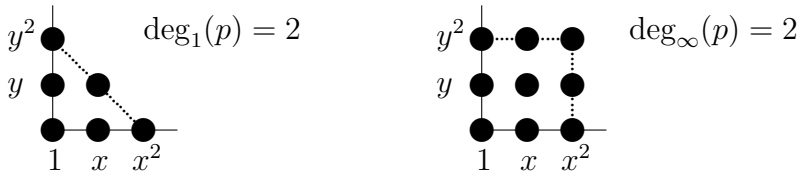


Figure 1: Polynomial supports representations

The triangular representation fits particularly well with homogenization. Infact, if p is triangular then $\overline{p}$ can be written as

$$\overline{p}(\overline{X}) = \sum_{|\overline{\alpha}| = \deg(p)} p_\alpha \overline{X}^{\overline{\alpha}} \qquad \text{where} \quad \overline{\alpha} = (\deg(p) - |\alpha|, \alpha_1, \cdots, \alpha_n)$$

Viceversa, if we start from a homogeneous dense polynomial, we can easily recover from it a triangular one by dehomogenizing it.

There are cases for which one of the dense representations is better than the other one, meaning that there are less zero coefficients. The square representation was already studied: we will analyze here the triangular representation, which is interesting for many real world applications.

Homogeneous n-variate polynomials study leads to consider dihedrons having dimension up to $(n-1)$. When we use the suffix "m-dimensional" we refer to homogeneous polynomials in $m+1$ variables. In this case, classical Toom methods will be called unidimensional, not univariate (they become bivariate after homogenization).

We indicate with $[p]$ the vector of the coefficients of p in the chosen representation.

2.2 *New methods for triangular density*

A general description of Toom algorithm follows: it can be described in five steps.

Splitting : Define $Y = X^\beta$, where β is an appropriate multindex. Rewrite $\widehat{a}(X), \widehat{b}(X) \in \mathbf{R}[X]$ as $a(Y), b(Y) \in \mathbf{R}[X][Y]$, having (polynomial) coefficients in $\mathbf{R}[X]$, each term of it with multiexponents componentwise strictly less than β. Finally, homogenize a and b with respect to y_0, obtaining $\overline{a}, \overline{b}$.

Traditionally Toom-k algorithm requires balanced operands, so that $d_a = \deg(\overline{a}) = \deg(\overline{b}) = d_b = k-1$, but we can easily generalize to unbalanced ones. We assume commutativity, hence $d_a \geq d_b > 1$. We call $\ell_p = \binom{\deg(p)+n}{n}$ the number of coefficients of a triangular polynomial p in n variables.

Evaluation : We want to compute $\overline{c} = \overline{a}\overline{b}$, whose degree is $d = d_a + d_b$, so we need $\ell = \binom{d+n}{n}$ evaluation points $P_{d,n} = \{\mathbf{p_1}, \ldots, \mathbf{p}_\ell\}$ where $\mathbf{p_i} \in \mathbf{R}[\mathbf{x}]^{\mathbf{n+1}}$ can be a not constant polynomial vector, for each i. We define $D = \max_i\{\max_j\{\deg((\mathbf{p_i})_\mathbf{j})\}\}$.

The evaluation of a single polynomial (for example $\overline{a}$) on the points $\mathbf{p_i}$ can be computed as a matrix by vector multiplication. To do this, we must linearly order the coefficients and the terms of $\overline{a}, \overline{b}, \overline{c}$ as

$$\overline{a}(\overline{Y}) = \sum_{i=1}^{\ell_a} a_i \overline{Y}^{\overline{\alpha}_i} \quad , \quad \overline{b}(\overline{Y}) = \sum_{i=1}^{\ell_b} b_i \overline{Y}^{\overline{\alpha}_i} \quad , \quad \overline{c}(\overline{Y}) = \sum_{i=1}^{\ell} c_i \overline{Y}^{\overline{\alpha}_i}$$

We indicate with E_{n,d,d_a} the resulting $\ell \times \ell_a$ matrix.

$$\overline{a}(P_{d,n}) = E_{n,d,d_a}[\overline{a}] \implies \begin{pmatrix} \overline{a}(\mathbf{p_1}) \\ \overline{a}(\mathbf{p_2}) \\ \vdots \\ \overline{a}(\mathbf{p}_\ell) \end{pmatrix} = \begin{pmatrix} \mathbf{p_1}^{\overline{\alpha}_1} & \mathbf{p_1}^{\overline{\alpha}_2} & \cdots & \mathbf{p_1}^{\overline{\alpha}_{\ell_a}} \\ \mathbf{p_2}^{\overline{\alpha}_1} & \mathbf{p_2}^{\overline{\alpha}_2} & \cdots & \mathbf{p_2}^{\overline{\alpha}_{\ell_a}} \\ \vdots & \vdots & & \vdots \\ \mathbf{p}_\ell^{\overline{\alpha}_1} & \mathbf{p}_\ell^{\overline{\alpha}_2} & \cdots & \mathbf{p}_\ell^{\overline{\alpha}_{\ell_a}} \end{pmatrix} \begin{pmatrix} \overline{a}_1 \\ \overline{a}_2 \\ \vdots \\ \overline{a}_{\ell_a} \end{pmatrix} \quad (1)$$

Recursive multiplication : We compute $\forall i$, $\overline{c}(\mathbf{p_i}) = \overline{\mathbf{a}}(\mathbf{p_i})\overline{\mathbf{b}}(\mathbf{p_i})$, with a total of ℓ multiplications of polynomials whose degree is comparable to that of Y. We have $\deg(\overline{a}(\mathbf{p_i})) \leq \mathbf{D}\mathbf{d_a} + \deg(\mathbf{Y})$, $\deg(\overline{b}(\mathbf{p_i})) \leq \mathbf{D}\mathbf{d_b} + \deg(\mathbf{Y})$, and the result $\deg(\overline{c}(\mathbf{p_i})) \leq \mathbf{D}\mathbf{d} + \mathbf{2}\deg(\mathbf{Y})$. We note that D, d, d_a, d_b are fixed numbers for a chosen implementation, $\deg(Y)$ will instead grow as the operands grow.

Interpolation : This step depends only on the expected degree of the result d, and on the ℓ chosen points $\mathbf{p_i}$, no more on d_a and d_b separately. We want to determine the coefficients of the polynomial $\overline{c}$. We know the values of $\overline{c}$ evaluated at ℓ points, so we face a classical interpolation problem. We need to multiply by the inverse of $A_{d,n}$, a $\ell \times \ell$ matrix.

$$[\overline{c}(P_{d,n})] = A_{d,n}[\overline{c}] \implies \begin{pmatrix} \overline{c}_1 \\ \overline{c}_2 \\ \vdots \\ \overline{c}_\ell \end{pmatrix} = \begin{pmatrix} \mathbf{p_1}^{\overline{\alpha}_1} & \mathbf{p_1}^{\overline{\alpha}_2} & \cdots & \mathbf{p_1}^{\overline{\alpha}_\ell} \\ \mathbf{p_2}^{\overline{\alpha}_1} & \mathbf{p_2}^{\overline{\alpha}_2} & \cdots & \mathbf{p_2}^{\overline{\alpha}_\ell} \\ \vdots & \vdots & & \vdots \\ \mathbf{p}_\ell^{\overline{\alpha}_1} & \mathbf{p}_\ell^{\overline{\alpha}_2} & \cdots & \mathbf{p}_\ell^{\overline{\alpha}_\ell} \end{pmatrix}^{-1} \begin{pmatrix} \overline{c}(\mathbf{p_1}) \\ \overline{c}(\mathbf{p_2}) \\ \vdots \\ \overline{c}(\mathbf{p}_\ell) \end{pmatrix} \quad (2)$$

Recomposition : Compute the final result with one more evaluation: $\widehat{c}(X) = \overline{c}(1, X^\beta)$.

2.3 *Complexity models*

We take into consideration different models to describe the complexity of the presented algorithms. Our landmark is the set of typically available operations in a computer. As well as addition, subtraction, left/right shift (optimized multiplication/division by powers of 2) and division, we will also take into consideration other two specific operations, that could be implemented in an ad-hoc optimized way.

1. Consider the following two equivalent processes (A) and (B)

$$(A)\ \begin{array}{l}\mathbf{T} \leftarrow \mathbf{2}^{|\mathbf{e}|}\mathbf{X};\\ \mathbf{Z} \leftarrow \mathbf{Y} \pm \mathbf{T};\end{array} \qquad ; \qquad (B)\ \ \mathbf{Z} \leftarrow \mathbf{Y} \pm \mathbf{2}^{|\mathbf{e}|}\mathbf{X};$$

   one could write a shift-add function $sa(\mathbf{X}, \mathbf{Y}, \mathbf{e}) \mapsto (\mathbf{Y} + \mathbf{sign}(\mathbf{e})\mathbf{2}^{|\mathbf{e}|}\mathbf{X})$ performing (B) process, which reads $\mathbf{X}$, $\mathbf{Y}$ and updates $\mathbf{Z}$ just once, taking benefit of code locality.

2. Similarly, we may have an ad hoc implemented multiply-by-3-and-add function $ma(\mathbf{X}, \mathbf{Y}) \mapsto (\mathbf{3X} + \mathbf{Y})$, for equivalent processes (A') and (B')

$$(A')\ \begin{array}{l}\mathbf{T} \leftarrow \mathbf{3X};\\ \mathbf{Z} \leftarrow \mathbf{T} + \mathbf{Y};\end{array} \qquad ; \qquad (B')\ \ \mathbf{Z} \leftarrow \mathbf{3X} + \mathbf{Y};$$

In the following, mimicking from [2], we will indicate execution time of every operation according to the following table

Operation		**Time**	**Operation**		**Time**
Addition	$+$	A	Division	$/$	D
Subtraction	$-$	A	Shift-add	sa	$A + \_1\_2$
Shift	$\ll, \gg$	S	Multiply-3-add	ma	$A + \_1\_3$

Usually one has $\_1\_2 \le S \le A \le A + \_1\_2 \le A + \_1\_3$ and $S \le D$. If ma is not available, one can define it as $ma(\mathbf{X}, \mathbf{Y}) = \mathbf{X} + \mathbf{sa}(\mathbf{X}, \mathbf{Y}, \mathbf{1})$, so that its cost becomes $A + (A + \_1\_2)$, and we can consider $\_1\_3 = A + \_1\_2$. If sa is not available, its cost is given by process (A), which gives $A + S$, so that in this case we have $\_1\_2 = S$ and $\_1\_3 = 2A + S$.

3. Multivariate Karatsuba

Karatsuba's idea was generalized in many ways; one of them is the extension to multivariate polynomials. Let two polynomials $\widehat{a}(X), \widehat{b}(X) \in \mathbf{R}[X]$ be given: we want to compute their product $\widehat{c}(X) = \widehat{a}(X)\widehat{b}(X)$ with the transformation techniques described in section 2.2. We obtain two homogeneous polynomials

$$a(\overline{Y}) = \sum_i a_i y_i \qquad ; \qquad b(\overline{Y}) = \sum_i b_i y_i$$

with $a_i, b_i \in \mathbf{R}[X]$, where a_0, b_0 are square, and the remaining ones triangular. All evaluation and interpolation can be condensed in a one-line formula, valid for any characteristic:

$$c(\overline{Y}) = a(\overline{Y})b(\overline{Y}) = \sum_i (a_i b_i)\, y_i^2 + \sum_{i<j} \left((a_i - a_j)(b_j - b_i) + a_i b_i + a_j b_j\right) y_i y_j \quad (3)$$

where any product $a_i b_i$ is computed only once, and recycled for the computation of the coefficients of all terms $y_i y_j$. Another possible formula for the product is the nearly equivalent

$$c(\overline{Y}) = a(\overline{Y})b(\overline{Y}) = \sum_i (a_i b_i)\, y_i^2 + \sum_{i<j} \left((a_i + a_j)(b_i + b_j) - a_i b_i - a_j b_j\right) y_i y_j \quad (4)$$

which is interesting for the following reason: if we use this formula for an univariate polynomial, with the identification $y_i = x^i$, we obtain the Karatsuba generalization given by Weimerskirch and Paar in [12].

3.1 *Application to sparse polynomial multiplication*

As an application of multivariate Karatsuba, we propose an algorithm to treat sparse polynomial multiplication, speeding it up in some cases. Let

$$\widehat{a}(X) = \sum_{T_i \in s(\widehat{a})} a_i T_i \qquad ; \qquad \widehat{b}(X) = \sum_{T_i \in s(\widehat{b})} b_i T_i$$

be (univariate or multivariate) sparse polynomials. Let $I = s(\widehat{a}) \cap s(\widehat{b})$, $S(\widehat{a}) = s(a) \setminus I$, $S(\widehat{b}) = s(\widehat{b}) \setminus I$ and split $\widehat{a}, \widehat{b}$ as follows

$$\widehat{a}(X) = \sum_{T_i \in I} a_i T_i + \sum_{T_i \in S(\widehat{a})} a_i T_i \qquad ; \qquad \widehat{b}(X) = \sum_{T_i \in I} b_i T_i + \sum_{T_i \in S(\widehat{b})} b_i T_i$$

If $^{\#}I > 1$ then their product $c(X)$ can then be computed as follows:

$$\widehat{c}(X) = \widehat{a}(X)\,\widehat{b}(X) = \sum_{T_i \in I} (a_i b_i)\, T_i^2 \tag{5}$$

$$+ \sum_{\substack{T_i, T_j \in I \\ i<j}} ((a_i - a_j)(b_j - b_i) + a_i b_i + a_j b_j)\, T_i T_j \tag{6}$$

$$+ \sum_{\substack{T_i \in I \\ T_j \in S(\widehat{b})}} (a_i b_j)\, T_i T_j + \sum_{\substack{T_i \in S(\widehat{a}) \\ T_j \in I}} (a_i b_j)\, T_i T_j + \sum_{\substack{T_i \in S(\widehat{a}) \\ T_j \in S(\widehat{b})}} (a_i b_j)\, T_i T_j$$

Note that the coefficients in sum (6), which would have been computed as $a_i b_j + a_j b_i$ (2 multiplications) with ordinary coefficient multiplication, can be instead obtained with just one multiplication if all coefficients appearing in sum (5) have already been computed before (see [8]).
Note: the above formula is useful when the products of the coefficients is costly. In general, more multiplications could be saved if a not too expensive criterion to test equality $T_i T_k = T_j T_h$ is available, because in this case it is still possible to apply Karatsuba's idea as follows:

$$\begin{aligned}(a_i T_i + a_j T_j)\cdot(b_h T_h + b_k T_k) &= (a_i b_h) T_i T_h + (a_j b_k) T_j T_k + (a_i b_k + a_j b_h) T_i T_k \\ &= (a_i b_h) T_i T_h + (a_j b_k) T_j T_k + ((a_i + a_j)(b_h + b_k) - a_i b_h - a_j b_k) T_i T_k\end{aligned}$$

4. Multivariate Toom-2.5

Multivariate Toom-2.5 concerns polynomial multiplication with unbalanced operands ($\deg_1(a) = 2$, $\deg_1(b) = 1$). In this case we have ${}^{\#}a = 6$,${}^{\#}b = 3$ and ${}^{\#}c = 10$.

$$\begin{aligned}a(y_0, y_1, y_2) &= a_5 y_1^2 + a_4 y_1 y_2 + a_3 y_2^2 + a_2 y_0 y_1 + a_1 y_0 y_2 + a_0 y_0^2 \\ b(y_0, y_1, y_2) &= b_0 y_0 + b_1 y_1 + b_2 y_2 \\ c(y_0, y_1, y_2) &= (a_5 b_1) y_1^3 + (a_5 b_2 + a_4 b_1) y_1^2 y_2 + (a_4 b_2 + a_3 b_1) y_1 y_2^2 + (a_3 b_2) y_2^3 \\ &\quad + (a_5 b_0 + a_2 b_1) y_0 y_1^2 + (a_4 b_0 + a_2 b_2 + a_1 b_1) y_0 y_1 y_2 + (a_3 b_0 + a_1 b_2) y_2^2 y_0 \\ &\quad + (a_2 b_0 + a_0 b_1) y_0^2 y_1 + (a_1 b_0 + a_0 b_2) y_0^2 y_2 + (a_0 b_0) y_0^3;\end{aligned}$$

4.1 *The bidimensional case in characteristic different from 2*

Ordering $s(c)$ as $(y_0^3, y_1 y_0^2, y_1^2 y_0, y_1^3, y_1^2 y_2, y_1 y_2^2, y_2^3, y_2^2 y_0, y_2 y_0^2, y_0 y_1 y_2)$, here follows a corresponding partial table from which to choose 10 linearly independent lines.

	y_0	y_1	y_2	**Evaluations**	**Matrix line**
1	1	0	0	a_0b_0	1 0 0 0 0 0 0 0 0 0
2	1	1	0	$(a_5+a_2+a_0)(b_0+b_1)$	1 1 1 1 0 0 0 0 0 0
3	1	-1	0	$(a_5-a_2+a_0)(b_0-b_1)$	1-1 1-1 0 0 0 0 0 0
4	0	1	0	a_5b_1	0 0 0 1 0 0 0 0 0 0
5	0	1	1	$(a_5+a_4+a_3)(b_1+b_2)$	0 0 0 1 1 1 1 0 0 0
6	0	1	-1	$(a_5-a_4+a_3)(b_1-b_2)$	0 0 0 1-1 1-1 0 0 0
7	0	0	1	a_3b_2	0 0 0 0 0 0 1 0 0 0
8	1	0	1	$(a_3+a_1+a_0)(b_2+b_0)$	1 0 0 0 0 0 1 1 1 0
9	-1	0	1	$(a_3-a_1+a_0)(b_2-b_0)$	-1 0 0 0 0 0 1-1 1 0
10	1	1	1	$(a_5+\cdots+a_0)(b_2+b_1+b_0)$	1 1 1 1 1 1 1 1 1 1
11	1	-1	1	$(a_5-a_4+a_3-a_2+a_1+a_0)(b_0-b_1+b_2)$	1-1 1-1 1-1 1 1-1-1
12	1	-1	-1	$(a_5+a_4+a_3-a_2-a_1+a_0)(b_0-b_1-b_2)$	1-1 1-1-1-1-1 1-1 1

The matrix results to be quite sparse and mainly containing three submatrices interlaced in a toroidal way, corresponding to three instances of unidimensional Toom-2.5. We report gp code for evaluation and interpolation obtained choosing the first 10 lines.

```
A = a0*y0^2  + a2*y1*y0 + a5*y1^2\
  + a1*y0*y2 + a4*y1*y2 \
  + a3*y2^2;
B = b0*y0 + b1*y1  \
  + b2*y2;

\\ Evaluation
W0 = b0 + b1; W9 = b0 - b1;
W3 = a5 + a0;
W6 = W3 - a2; W3 = W3 + a2;

W1 = W3 * W0;     \\ C(1, 1,0)
W2 = W6 * W9;     \\ C(1,-1,0)

W0 = b1 + b2; W9 = b1 - b2;
W3 = a5 + a3
W6 = W3 - a4; W3 = W3 + a4;

W4 = W3 * W0;     \\ C(0,1, 1)
W5 = W6 * W9;     \\ C(0,1,-1)

W0 = b2 + b0; W9 = b2 - b0;
W3 = a3 + a0;
W6 = W3 - a1; W3 = W3 + a1;

W7 = W3 * W0;     \\ C(-1,0,1)
W8 = W6 * W9;     \\ C( 1,0,1)
W0 = W0 + b1;
W3 = W3 + a5 + a4 + a2;

W9 = W3 * W0;     \\ C(1,1,1)
W0 = a0 * b0;     \\ C(1,0,0)
W3 = a5 * b1;     \\ C(0,1,0)
W6 = a3 * b2;     \\ C(0,0,1)

\\ Interpolation (matrix inversion)

W2 = ( W2 + W1 )/2;
W5 = ( W5 + W4 )/2;
W8 = ( W8 + W7 )/2;

W1 = W1 - W3; W4 = W4 - W6; W7 = W7 - W0;

W9 = W9 - W1 - W4 - W7;

W1 = W1 - W2; W4 = W4 - W5; W7 = W7 - W8;
W2 = W2 - W0; W5 = W5 - W3; W8 = W8 - W6;

\\ Product reconstruction

C = W0*y0^3    + W1*y0^2*y1  + W2*y0*y1^2 +W3*y1^3\
  + W8*y0^2*y2 + W9*y0*y1*y2 + W4*y1^2*y2 \
  + W7*y0*y2^2 + W5*y1*y2^2 \
  + W6*y2^3
```

4.2 *The general case in characteristic different from* 2 *and* 3

Toom-2.5 can be generalized to an arbitrary number of variables. If

$$a(\overline{Y}) = \sum_i a_i y_i^2 + \sum_{i<j} a_{ij} y_i y_j \qquad ; \qquad b(\overline{Y}) = \sum_i b_i y_i$$

then their product $c(\overline{Y})$ can be obtained as

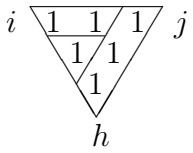
$$\begin{aligned} c(\overline{Y}) = & \sum_i (a_i b_i)\, y_i^3 \\ & + \sum_{i<j} \left(\frac{(a_i + a_j + a_{ij})(b_i + b_j) - (a_i + a_j - a_{ij})(b_i - b_j)}{2} - a_j b_j \right) y_i^2 y_j \\ & + \sum_{i<j} \left(\frac{(a_i + a_j + a_{ij})(b_i + b_j) + (a_i + a_j - a_{ij})(b_i - b_j)}{2} - a_i b_i \right) y_i y_j^2 \\ & + \sum_{i<j<h} \big((a_i + a_j + a_h + a_{ij} + a_{ih} + a_{jh})(b_i + b_j + b_h) \\ & \quad - (a_i + a_j + a_{ij})(b_i + b_j) - (a_j + a_h + a_{jh})(b_j + b_h) \\ & \quad - (a_i + a_h + a_{ih})(b_i + b_h) + a_i b_i + a_j b_j + a_h b_h \big)\, y_i y_j y_h \end{aligned}$$

◇ **Evaluation**

Evaluation phase must compute $\binom{n+2}{3}$ values per factor. In order to obtain good efficiency, the idea is to recycle as much as possible intermediate obtained values. We'll refer to the values concerning $\overline{a}$ evaluation according to their dependency on i, (i,j) or (i,j,h) indexes, respectively, as:

vertices : a_i
sides : evaluated "in 1", $(a_i + a_j + a_{ij})$; "in -1", $(a_i + a_j - a_{ij})$
triangles : $(a_i + a_j + a_h + a_{ij} + a_{ih} + a_{jh})$

Evaluations of $\overline{b}$ and vertices management do not present any difficulty. The computation of (i,j,h) triangle is instead someway tricky. The idea is to obtain it by summing three addends: for example the (j,h) side (evaluated in 1), the "internal point" a_{ih} and a partially evaluated term $ps_{ij} = a_i + a_{ij}$ on side (i,j). This corresponds to the splitting $(a_j + a_h + a_{jh}) + a_{ih} + (a_i + a_{ij})$, and is pictorially represented aside. Similar splittings considering ps_{ih} or ps_{jh} are obviously possible.



Consider the following sequences for sides evaluation in 1 and -1:

$$(I)\quad \begin{aligned} &1) \quad v_1 = a_i + a_j \\ &2) \quad v_2 = v_1 - a_{ij} \\ &3) \quad v_1 = v_1 + a_{ij} \end{aligned} \qquad ; \qquad (II)\quad \begin{aligned} &1) \quad v_1 = a_i + a_{ij} \\ &2) \quad v_1 = v_1 + a_j \\ &3) \quad v_2 = v_1 - (a_{ij} \ll 1) \end{aligned}$$

The first one is optimal (with a cost of $3A$), but it does not contain the term ps_{ij}. We then sometimes have to use the second one (with cost $3A + \_1\_2$) in order to complete the triangle. How many partial terms do we need ? As different splittings are possible, the idea is to use ps_{pq} for as many triangles as possible, as term for the "lower" side (i, j), the "middle" side (i, h), or for the "higher" side (j, h), and accordingly completing the covering of the triangle. This leads to a someway involved code, with which the number of (II) sequences (and therefore of extra $\_1\_2$) is remarkably small.

◇ Interpolation

The key point is the correct generalization of the instruction `W9 = W9 - W7 - W4 - W1`, when operands are as shown aside.

`W1`	(1	1	1	0	0	0	0	0	0	0)
`W4`	(0	0	0	1	1	1	0	0	0	0)
`W7`	(0	0	0	0	0	0	1	1	1	0)
`W9`	(1	1	1	1	1	1	1	1	1	1)

The three subtractions use temporary (`W1`, `W4`, `W7`) unidimensional Toom-2.5 values to obtain a single "1" in `W9` line. This makes possible to use the three (1 1 1) juxtaposable configurations to reduce to 3 the number of operations needed to obtain the coefficients of the terms $y_i y_j y_h$ (with $i < j < h$), instead of the 6 ones needed by blindly applying the inclusion-exclusion principle in equation (). In this case we say that (i, j, h) is *good*.

When $n > 3$, it is not possible to have the above situation for *all* triples (i, j, h). What happens is e.g. the configuration shown aside: a not perfect juxtaposition of the necessary "1", so that some correction must be done – (i, j, h) is *bad*.

`W`p	(1	1	1	0	0	0	0	0	0	0)
`W`q	(0	0	0	1	1	1	0	0	0	0)
`W`r	(1	0	0	0	0	0	0	1	1	0)
`W`s	(1	1	1	1	1	1	1	1	1	1)

To describe the situation, consider the complete graph $G_n = K_{n+1} = (V(G_n), E(G_n))$, where $V(G_n)$ is the set of vertices, $E(G_n)$ the set of edges, and in which every vertex corresponds to a variable. The triples (i, j, h) corresponds to simple circuits of cardinality 3 (*sc*, for brevity). To obtain the three correctly juxtaposable groups (1 1 1), it is necessary that the single "1" to be removed from the groups (1 1 1 1) are properly subtracted. This can be modeled by orienting the edges of G_n. If (i, j, h) is a good *sc* in G_n, then no correction will be necessary (first triangle in figure 2), otherwise one addition will: the successive subtractions of a configuration (1 1 1 1) on one side and

two (1 1 1) on the other two sides should be corrected by the addition of a single (1) in the vertex which was subtracted two times (second triangle in figure 2, where vertex i is doubly subtracted).

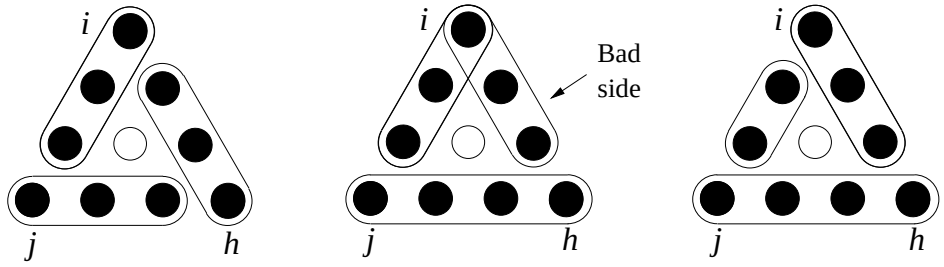


Figure 2: Triangles: good orientations, bad orientations, general solution

The solution consisting in an edges orientation maximizing the number of good triangles is not the best one. Infact, if such a solution is choosed e.g. by defining

Orientation : $i \to j$ iff $(i \not\equiv j \bmod 2)$	Good triangle : $(i \equiv h \not\equiv j \bmod 2)$

it is easily proved that the (maximal) number $GT(n)$ of good triangles is

$$GT(n) = \begin{cases} \dfrac{n(n^2-4)}{24} & \text{if } n \equiv 0 \bmod 2 \\ \dfrac{n(n^2-1)}{24} & \text{if } n \equiv 1 \bmod 2 \end{cases}$$

so that the number of necessary corrections is $\binom{n}{3} - GT(n) = O(n^3)$. A better solution is to use a not optimal inversion sequence for unidimensional Toom-2.5. The idea is to manage three types of configurations: (1 1 1 1) and (1 1 1 0) – or (0 1 1 1) – which are always available at no extra cost, and (0 1 1 0). This means that we need an extra operation on each *side*, not on each triangle. In particular, the configuration (0 1 1 0) on side (i, j) can be used for all triangles (i, j, h) with $h > j$, so that a correction on one single side works for many triangles. The procedure is (third triangle in figure 2):
– Subtract from the "higher" side (j, h) of all triangles the configuration (1 1 1 1)
– Subtract from all sides (i, j) the vertex j, obtaining the configuration $\kappa = (1\ 1\ 1\ 0)$
– Subtract from the "middle" side (i, h) of all triangles the just obtained configuration κ

– Subtract from all sides (i, j) the vertex i, obtaining the configuration $\kappa' = (0\ 1\ 1\ 0)$
– Subtract from the "lower" side (i, j) of all triangles the just obtained configuration κ'

Actually it's not difficult to have also a single good triangle, which should be treated apart, so that the number of needed corrections lowers to $\binom{n-1}{2} - 1 = O(n^2)$. We have

Z) Toom-2.5 (n)	A	S	_1_2
Evaluation	$3\binom{n}{3} + 5\binom{n}{2}$		$\binom{\lceil n/2 \rceil}{2} + \binom{\lfloor n/2 \rfloor}{2}$
Interpolation	$3\binom{n}{3} + 4\binom{n}{2} + \binom{n-1}{2} - 1$	$\binom{n}{2}$	

An implementation in gp code is reported in appendix A. ⋄ **The general case in characteristic** 2 **and** 3

In characteristic 2, 1 and -1 coincide, therefore another value must be chosen, and it can well be x_i, for a fixed i. In [1], Bodrato described the univariate approach. We have

Z_2) Toom-2.5 (n)	A	S	D
Evaluation	$3\binom{n}{3} + 6\binom{n}{2}$	$\binom{n}{2} + 2(n-1)$	
Interpolation	$3\binom{n}{3} + 6\binom{n}{2}$	$2\binom{n}{2}$	$\binom{n}{2}$

An implementation in gp code is reported in appendix B. In characteristic 3 we can use again 1 and -1, as they are different. Considering that $2 \equiv -1 \pmod 3$, it is possible to avoid *all* shifts by slightly modifying Toom-2.5 for integers: the general case is obtained as in section 4.2, adapting the inversion sequence for sides.

5. Multivariate Toom-3

We first treat some particular cases, representing the building blocks for which it is possible to reach full optimality in interpolation phase. For the general case, some extra operation is instead needed. We report, either in the

following sections or in appendices C and D, effective `gp` code implementing the presented algorithms.

5.1 *The bidimensional case*

Bidimensional Toom-3 can be used in two different cases, for balanced or non-balanced polynomials, respectively. Setting $d(p) = \deg_1(p)$ we have

(1)	$a(y_0, y_1, y_2) = a_5y_1^2 + a_4y_1y_2 + a_3y_2^2 + a_2y_1y_0 + a_1y_2y_0 + a_0y_0^2$
$d(a) = d(b) = 2$	$b(y_0, y_1, y_2) = b_5y_1^2 + b_4y_1y_2 + b_3y_2^2 + b_2y_1y_0 + b_1y_2y_0 + b_0y_0^2$
(2)	$a(y_0, y_1, y_2) = a_9y_1^3 + a_8y_1^2y_2 + a_7y_1y_2^2 + a_6y_2^3 + a_5y_1^2y_0$
$d(a) = 3$	$+ a_4y_1y_2y_0 + a_3y_2^2y_0 + a_2y_1y_0^2 + a_1y_2y_0^2 + a_0y_0^3$
$d(b) = 1$	$b(y_0, y_1, y_2) = b_1y_1 + b_2y_2 + b_0y_0$

We need 15 points to determine c. Ordering $s(c)$ similarly as before ($y_0^4, y_0^3y_1, y_0^2y_1^2, y_0y_1^3, y_1^4, y_1^3y_2, y_1^2y_2^2, y_1y_2^3, y_2^4, y_2^3y_0, y_2^2y_0^2, y_2^3y_0, y_0^2y_1y_2, y_0y_1^2y_2, y_0y_1y_2^2$), we obtain the situation presented in figure 3. Note that it is quite easy to partially describe the shape of the generic bidimensional Toom-k matrix $A_{3,k}$ in terms of the optimal classical Toom-k matrix $A_k = A_{2,k}$, for whatever k. Ordering $s(c)$ such that the initial terms are the ones met traveling on the "perimeter" of c

$$s(c) = (\, y_0^{2k-1}, y_0^{2k-2}y_1, \ldots, y_0y_1^{2k-2}, y_1^{2k-1}, y_1^{2k-2}y_2, \ldots, y_1y_2^{2k-2}, \\ y_2^{2k-1}, y_2^{2k-2}y_0, \ldots, y_2y_0^{k-2}, \ldots)$$

and choosing values for y_0, y_1, y_2 similarly as we did above, corresponding to the optimal classical Toom-k method, we have that $A_{3,k}$ will have the below shape.

	y_0	y_1	y_2	**Evaluation (case 1)**	**Matrix line**
1	1	0	0	a_0b_0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2	2	1	0	$(4a_0 + 2a_2 + a_5)(4b_0 + 2b_2 + b_5)$	16 8 4 2 1 0 0 0 0 0 0 0 0 0 0
3	1	1	0	$(a_5 + a_2 + a_0)(b_5 + b_2 + b_0)$	1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
4	-1	1	0	$(a_5 - a_2 + a_0)(b_5 - b_2 + b_0)$	1 -1 1 -1 1 0 0 0 0 0 0 0 0 0 0
5	0	1	0	a_5b_5	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
6	0	2	1	$(4a_5 + 2a_4 + a_3)(4b_5 + 2b_4 + b_3)$	0 0 0 0 16 8 4 2 1 0 0 0 0 0 0
7	0	1	1	$(a_5 + a_4 + a_3)(b_5 + b_4 + b_3)$	0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
8	0	-1	1	$(a_5 - a_4 + a_3)(b_5 - b_4 + b_3)$	0 0 0 0 1 -1 1 -1 1 0 0 0 0 0 0
9	0	0	1	a_3b_3	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
10	1	0	2	$(4a_3 + 2a_1 + a_0)(4b_3 + 2b_1 + b_0)$	1 0 0 0 0 0 0 0 16 8 4 2 0 0 0
11	1	0	1	$(a_3 + a_1 + a_0)(b_3 + b_1 + b_0)$	1 0 0 0 0 0 0 0 1 1 1 1 0 0 0
12	-1	0	1	$(a_3 - a_1 + a_0)(b_3 - b_1 + b_0)$	1 0 0 0 0 0 0 0 1 -1 1 -1 0 0 0
13	1	1	1	$(a_5 + \cdots + a_0)(b_5 + \cdots + b_0)$	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
14	-1	1	1	$(a_5+a_4+a_3-a_2-a_1+a_0)(\langle$idem for $b\rangle)$	1 -1 1 -1 1 1 1 1 1 -1 1 -1 1 -1 -1
15	1	-1	1	$(a_5-a_4+a_3-a_2+a_1+a_0)(\langle$idem for $b\rangle)$	1 -1 1 -1 1 -1 1 -1 1 1 1 1 -1 1 -1

	y_0	y_1	y_2	**Evaluation (case 2)**
1	1	0	0	a_0b_0
2	2	1	0	$(8a_0 + 4a_2 + 2a_5 + a_9)(b_1 + 2b_0)$
3	1	1	0	$(a_9 + a_5 + a_2 + a_0)(b_1 + b_0)$
4	-1	1	0	$(a_9 - a_5 + a_2 - a_0)(b_1 - b_0)$
5	0	1	0	a_9b_1
6	0	2	1	$(8a_9 + 4a_8 + 2a_7 + a_6)(2b_1 + b_2)$
7	0	1	1	$(a_9 + a_8 + a_7 + a_6)(b_2 + b_1)$
8	0	-1	1	$(a_9 - a_8 + a_7 - a_6)(b_2 - b_1)$
9	0	0	1	a_6b_2
10	1	0	2	$(8a_6 + 4a_3 + 2a_1 + a_0)(2b_2 + b_0)$
11	1	0	1	$(a_6 + a_3 + a_1 + a_0)(b_2 + b_0)$
12	-1	0	1	$(a_6 - a_3 + a_1 - a_0)(b_2 - b_0)$
13	1	1	1	$(a_9 + \cdots + a_0)(b_2 + b_1 + b_0)$
14	-1	1	1	$(a_9 + a_8 + a_7 + a_6 - a_5 - a_4 - a_3 + a_2 + a_1 - a_0)(b_1 - b_0 + b_2)$
15	1	-1	1	$(a_9 - a_8 + a_7 - a_6 - a_5 + a_4 - a_3 + a_2 - a_1 - a_0)(b_1 - b_0 - b_2)$

Figure 3: Bidimensional Toom-3: evaluation phase.

$$A_{3,k} = \begin{pmatrix} 1 & & \\ & A_k & \\ & 1 & 0 \\ 0 & & A_k \\ & 1 & \\ & & A_k \\ \cdots & \cdots & \cdots \end{pmatrix}$$

The last lines concern the inner terms T_j of the triangle-shaped support, with $y_i \,|\, T_j \;\forall\, i, j$. The optimal sequence of basic operations needed to invert $A_{3,k}$ (we call it inversion sequence, or IS for short) cannot be straightforwardly deduced from the corresponding IS of A_k, because of the last lines to be treated, but can of course benefit a lot from it.

We report gp code for the two evaluations and for the (common) interpolation.

Balanced Toom-3

```
A = a0*y0^2  + a2*y0*y1 + a5*y1^2\
  + a1*y0*y2 + a4*y1*y2 \
  + a3*y2^2;
B = b0*y0^2  + b2*y0*y1 + b5*y1^2\
  + b1*y0*y2 + b4*y1*y2 \
  + b3*y2^2;

W0 = a0 + a5;        W4 = b0 + b5;
W1 = W0 - a2;        W2 = W4 - b2;
W0 = W0 + a2;        W4 = W4 + b2;

\\ Last 3 rows (internal points).
W10 = W1 + a3;
W14 = a1 - a4;
W13 = W10 + W14;
W12 = W10 - W14;

W10 = W2 + b3;
W14 = b1 - b4;
W11 = W10 + W14;
W10 = W10 - W14;

W14 = W13*W11;     \\ C(1,-1,1)
W13 = W12*W10;     \\ C(-1,1,1)

W10 = W0 + a1 + a4 + a3;
W11 = W4 + b1 + b4 + b3;

W12 = W10*W11;     \\ C(1,1,1)

\\ End first Toom-3 submatrix.
W3 = W1*W2;        \\ C(1,-1,0)
W2 = W0*W4;        \\ C(1, 1,0)

W0 = W0 + a0;   W0 = 2*W0 - a5;
W4 = W4 + b0;   W4 = 2*W4 - b5;

W1 = W0*W4;        \\ C(2,1,0)

\\ Inner rows of second Toom submatrix.
W4 = a5 + a3;        W8 = b5 + b3;
W5 = W4 - a4;        W6 = W8 - b4;
W4 = W4 + a4;        W8 = W8 + b4;

W7 = W5*W6;        \\ C(0,1,-1)
W6 = W4*W8;        \\ C(0,1, 1)

W4 = W4 + a5;   W4 = 2*W4 - a3;
W8 = W8 + b5;   W8 = 2*W8 - b3;
```

Unbalanced Toom-3

```
A = a0*y0^3   +a2*y0^2*y1 +a5*y0*y1^2+a9*y1^3\
  + a1*y0^2*y2+a4*y0*y1*y2+a8*y1^2*y2 \
  + a3*y0*y2^2+a7*y1*y2^2 \
  + a6*y2^3;
B = b0*y0 + b1*y1 \
  + b2*y2;

W4 = a2 + a9;    W3 = a0 + a5;
W0 = W4 + W3;    W1 = W4 - W3;

W6 = b1 - b0; W3 = W1*W6;    \\ C(1,-1,0)

\\ Last 3 lines (I)
W7 = a7 + a4;    W10 = a3 + a1;
W8 = W0 + W7;    W8  = W8 + W10;
W9 = W1 + W7;    W9  = W9 - W10;

W10 = W1 + a7 - a4 - a3 + a1;

\\ Continue evaluation of W1, W2, W3

W5 = b1 + b0;  W2 = W0*W5;   \\ C(1,1,0)

W4  = 2*a0;    W4 += a2;      W4  = 2*W4;
W4 += a5;      W4  = 2*W4;    W4 += a9;

W0 = W5 + b0;  W1 = W0*W4;   \\ C(2,1,0)
W7 = a6 + a8;

\\ Last 3 lines (II)
W0 = W8 + W7;
W5 = W5 + b2;  W12 = W0*W5;  \\ C(1,1,1)

W0 = W10 + W7;
W5 = W6 + b2;  W13 = W0*W5;  \\ C(-1,1,1)

W0 = W9 - W7;
W5 = W6 - b2;  W14 = W0*W5;  \\ C(1,-1,1)

\\ Evaluate W5, W6, W7.
W4 = a7 + a9; W0 = W7 + W4; W5 = W7 - W4;

W6 = b2 - b1;  W7 = W5*W6;   \\ C(0,1,-1)

W5 = b2 + b1;  W6 = W0*W5;   \\ C(0,1,1)

W4  = 2*a9;    W4 += a8;     W4  = 2*W4;
W4 += a7;      W4 = 2*W4;    W4 += a6;

W0 = W5 + b1;  W5 = W0*W4;   \\ C(0,2,1)
```

```
W5 = W4*W8;        \\ C(0,2,1)
                                          \\ Evaluate W9, W10, W11.
\\ Inner rows of second Toom submatrix.    W4  = a3 + a0;
W8 = a0 + a3;        W0  = b0 + b3;       W11 = a6 + a1;
W9 = W8 - a1;        W10 = W0 - b1;       W9  = W4 - W11;
W8 = W8 + a1;        W0  = W0 + b1;       W10 = b0 - b2;

W11 = W9*W10;      \\ C(-1,0,1)           W0  = W4 + W11;W11 = W9*W10; \\ C(1,0,-1)
W10 = W8*W0;       \\ C(1,0,1)            W9  = b2 + b0; W10 = W0*W9;  \\ C(1,0,1)

W8 = W8 + a3;   W8 = 2*W8 - a0;           W4  = 2*a6;    W4 += a3;     W4  = 2*W4;
W0 = W0 + b3;   W0 = 2*W0 - b0;           W4 += a1;      W4  = 2*W4;   W4 += a0;
W9 = W8*W0;        \\ C(1,0,2)
                                          W0 = W9 + b2;  W9 = W0*W4;   \\ C(2,0,1)
W0 = a0*b0;   W4 = a5*b5;   W8 = a3*b3;   W0 = a0*b0;    W4 = a9*b1;   W8 = a6*b2;

        \\ Interpolation: partial Toom-3 submatrices invertion
        W1 = (W1 - W3)/3 ; W5 = (W5 - W7)/3 ; W9  = (W9  - W11)/3;
        W3 = (W2 - W3)/2 ; W7 = (W6 - W7)/2 ; W11 = (W10 - W11)/2;
        W2 =  W2 - W4    ; W6 =  W6 - W8    ; W10 =  W10 - W0    ;
        W1 = (W1 - W2)/2 ; W5 = (W5 - W6)/2 ; W9  = (W9 - W10)/2 ;
        W1 =  W1 - 2*W0  ; W5 =  W5 - 2*W4  ; W9  =  W9 - 2*W8   ;

        \\ Last three rows (partial)
        W14 = (W12 - W14)/2 ; W13 = (W12 - W13)/2;
        W14 =  W14 - W7     ; W12 =  W12 - W6    ;
        W13 =  W13 - W11    ; W12 =  W12 - W10   ;

        \\ Continue Toom-3 submatrices and last 3 lines invertion.
        W2  = W2  - W3   ; W6 = W6  - W7    ; W10 = W10 - W11;

        W12 = W12 - W13;
        W12 = W12 - W2 ;
        W14 = W14 - W12;
        W13 = W13 - W14;
        W14 = W14 - W3 ;

        \\ End of Toom-3 submatrices invertion
        W2 = W2 - W0     ; W6 = W6 - W4     ; W10 = W10 - W8;
        W3 = W3 - W1     ; W7 = W7 - W5     ; W11 = W11 - W9;

        \\ Product reconstruction
        C = W0 *y0^4      + W1 *y0^3   *y1 + W2 *y0^2*y1^2  + W3*y0*y1^3 + W4*y1^4 \
          + W11*y0^3*y2   + W12*y0^2*y2*y1 + W13*y0*y2*y1^2 + W5*y2*y1^3 \
          + W10*y0^2*y2^2 + W14*y0*y2^2*y1 + W6 *y2^2*y1^2 \
          + W9 *y0 *y2^3  + W7 *y2^3*y1 \
          + W8 *y2^4;
```

We underline that the above IS is optimal in the model described in [2]. Note that after the two operations

```
W14 = (W12 - W14)/2;   W13 = (W12 - W13)/2;
```

the last three lines will contain a submatrix with the following configuration, which, inspired by its shape, we name Π.

$$\Pi = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

By choosing which variable set to the value -1 in the interpolating points, it is possible to associate Π to whatever side of the triangle we prefer. This freedom will be crucial for the treatment of the general case.

5.2 *The tridimensional case in characteristic different from 2 and 3*

While Toom-2.5 already reaches its full generality when dimension is 2, Toom-3 attains it from 3, with the geometric setting being a tridimensional tetrahedron (pyramid). The 4 vertices are associated to basic multiplications $a_i b_i$, the 6 sides to unidimensional Toom-3, and the 4 faces to bidimensional Toom-3. To represent the pyramid on a plane we "open" it, identifying some sides and vertices of the "open" faces with equal symbols – see figure 4(a). There is a single inner point, which is not represented.

Each face has 3 internal points: the associated 35×35 matrix $A_{4,3}$ generated by

$$\begin{aligned} P_{4,3} = \{&(1,0,0,0),(2,1,0,0),(1,1,0,0),(1,-1,0,0),(0,1,0,0),(0,2,1,0),\\ &(0,1,1,0),(0,1,-1,0),(0,0,1,0),(1,0,2,0),(1,0,1,0),(-1,0,1,0),\\ &(0,0,0,1),(1,0,0,2),(1,0,0,1),(1,0,0,-1),(0,1,0,2),(0,1,0,1),\\ &(0,1,0,-1),(0,0,1,2),(0,0,1,1),(0,0,1,-1),(1,1,1,0),(1,1,-1,0),\\ &(1,-1,1,0),(1,1,0,1),(1,-1,0,1),(-1,1,0,1),(1,0,1,1),(-1,0,1,1),\\ &(1,0,1,-1),(0,1,1,1),(0,1,1,-1),(0,1,-1,1),(1,1,1,1)\} \end{aligned}$$

with columns indexed by

$$\begin{aligned} s(\bar{c}) = \{&y_0^4, y_1y_0^3, y_1^2y_0^2, y_1^3y_0, y_1^4, y_2y_1^3, y_2^2y_1^2, y_2^3y_1, y_2^4, y_2^3y_0, y_2^2y_0^2, y_2y_0^3, y_3^4, y_3^3y_0,\\ &y_3^2y_0^2, y_3y_0^3, y_3^3y_1, y_3^2y_1^2, y_3y_1^3, y_3^3y_2, y_3^2y_2^2, y_3y_2^3, y_2^2y_1y_0, y_2y_1^2y_0, y_2y_1y_0^2,\\ &y_3y_1^2y_0, y_3y_1y_0^2, y_3^2y_1y_0, y_3y_2y_0^2, y_3^2y_2y_0, y_3y_2^2y_0, y_3^2y_2y_1, y_3y_2^2y_1, y_3y_2y_1^2,\\ &y_3y_2y_1y_0\} \end{aligned}$$

is the following, where zero entries are not indicated (the last line corresponds to the inner point, subblocks to vertices, faces and sides – dihedrons of smaller dimension).

$$A_{4,3} = \begin{pmatrix}
1 &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& &&&& \\
16 & 8 & 4 & 2 & 1 &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& \\
1 & 1 & 1 & 1 & 1 &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& \\
1 & -1 & 1 & -1 & 1 &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& \\
&&&& 1 &&&&& &&&&& &&&&& &&&&& &&&&& &&&&& \\
&&&& 16 & 8 & 4 & 2 & 1 &&&&& &&&&& &&&&& &&&&& &&&&& & \\
&&&& 1 & 1 & 1 & 1 & 1 &&&&& &&&&& &&&&& &&&&& &&&&& & \\
&&&& 1 & -1 & 1 & -1 & 1 &&&&& &&&&& &&&&& &&&&& &&&&& & \\
&&&&& &&& 1 &&&&& &&&&& &&&&& &&&&& &&&&& & \\
1 &&&&& &&& 16 & 8 & 4 & 2 &&&&& &&&&& &&&&& &&&&& &&& \\
1 &&&&& &&& 1 & 1 & 1 & 1 &&&&& &&&&& &&&&& &&&&& &&& \\
1 &&&&& &&& 1 & -1 & 1 & -1 &&&&& &&&&& &&&&& &&&&& &&& \\
&&&&& &&&&& && 1 &&&&& &&&&& &&&&& &&&&& && \\
1 &&&&& &&&&& && 16 & 8 & 4 & 2 &&&&& &&&&& &&&&& &&&& \\
1 &&&&& &&&&& && 1 & 1 & 1 & 1 &&&&& &&&&& &&&&& &&&& \\
1 &&&&& &&&&& && 1 & -1 & 1 & -1 &&&&& &&&&& &&&&& &&&& \\
&&&& 1 &&&&& &&& 1 &&&& 8 & 4 & 2 &&&&& &&&&& &&&&& & \\
&&&& 1 &&&&& &&& 1 &&&& 1 & 1 & 1 &&&&& &&&&& &&&&& & \\
&&&& 1 &&&&& &&& 1 &&&& -1 & 1 & -1 &&&&& &&&&& &&&&& & \\
&&&&& &&& 1 &&&& 16 &&&&& && 8 & 4 & 2 &&&&& &&&&& &&& \\
&&&&& &&& 1 &&&& 1 &&&&& && 1 & 1 & 1 &&&&& &&&&& &&& \\
&&&&& &&& 1 &&&& 1 &&&&& && -1 & 1 & -1 &&&&& &&&&& &&& \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 &&&&& &&&&& & 1 & 1 & 1 &&&&& &&&&& \\
1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 &&&&& &&&&& & 1 & -1 & -1 &&&&& &&&&& \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & 1 & 1 &&&&& &&&&& & -1 & 1 & -1 &&&&& &&&&& \\
1 & 1 & 1 & 1 & 1 &&&&& &&& 1 & 1 & 1 & 1 & 1 & 1 & 1 &&&&& && 1 & 1 & 1 &&&&& && \\
1 & -1 & 1 & -1 & 1 &&&&& &&& 1 & 1 & 1 & 1 & -1 & 1 & -1 &&&&& && 1 & -1 & -1 &&&&& && \\
1 & -1 & 1 & -1 & 1 &&&&& &&& 1 & -1 & 1 & -1 & 1 & 1 & 1 &&&&& && -1 & 1 & -1 &&&&& && \\
1 &&&&& &&& 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 &&&& 1 & 1 & 1 &&&&& && 1 & 1 & 1 &&&& \\
1 &&&&& &&& 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 &&&& 1 & 1 & 1 &&&&& && 1 & -1 & -1 &&&& \\
1 &&&&& &&& 1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 &&&& -1 & 1 & -1 &&&&& && -1 & 1 & -1 &&&& \\
&&&& 1 & 1 & 1 & 1 & 1 &&&& 1 &&&& 1 & 1 & 1 & 1 & 1 & 1 &&&&& &&&&& 1 & 1 & 1 & \\
&&&& 1 & 1 & 1 & 1 & 1 &&&& 1 &&&& -1 & 1 & -1 & -1 & 1 & -1 &&&&& &&&&& 1 & -1 & -1 & \\
&&&& 1 & -1 & 1 & -1 & 1 &&&& 1 &&&& 1 & 1 & 1 & -1 & 1 & -1 &&&&& &&&&& -1 & 1 & -1 & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

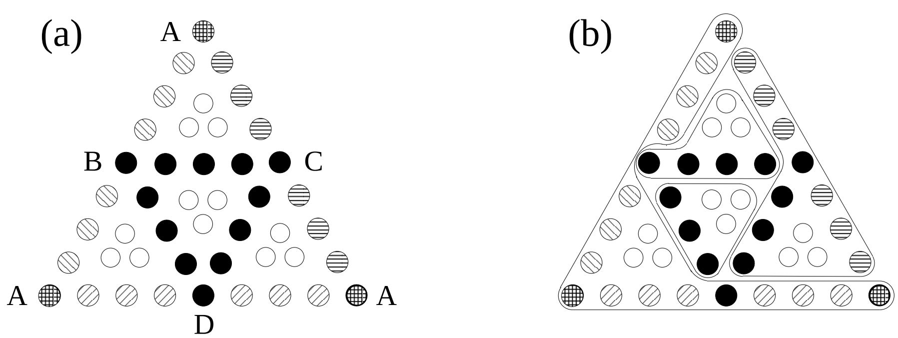


Figure 4: Tridimensional Toom-3 (a) Pyramidal view (b) Inner point isolation: optimal grouping

At least 4 operations are needed to make the last line a singleton one (that is, to isolate the single inner point), corresponding to removing the four groups of three white circles in figure 4. The IS we propose is derived by the optimal

IS of unidimensional and bidimensional Toom-3 methods, and with exactly 4 operations the last line is correctly treated. As with bidimensional Toom-2.5 method, no correction (extra operation) is needed. Due to the code length, we present only the interpolation phase. The matrix line indexes range from 0 to 34, with `Wi` corresponding to the $(i+1)^{\text{th}}$ line. The situation just before the execution of the instruction `W34 -= (W22 + W25 + W28 + W31)` is pictorially described in figure 4(b). Each of the four subtrahends contains the three inner points of the corresponding face and, in decreasing cardinality order, other (8,7,4,3) points on the sides. This way we obtain the $(3+8)+(3+7)+(3+4)+(3+3)=34$ ones to be subtracted. The $(8,7,4,3)$ is not the only optimal decomposition one can obtain: other ones are e.g.

$$(12,4,3,3) \quad , \quad (8,8,3,3) \quad , \quad (7,7,5,3) \quad , \quad (7,7,4,4)$$

It is interesting to note that it was possible to obtain the proposed (8,7,4,3) configuration because the four Π which appear during the inversion have not the same column indexes. This means that no two faces sharing a side s have their Π associated to s. This was possible, because there are 6 sides and only 4 faces, so that there is even a certain degree of freedom in choosing where to let Π appear.

```
\\ Tridimensional Toom-3 interpolation
W23 = (W22 - W23)/2;   W24 = (W22 - W24)/2;
W26 = (W25 - W26)/2;   W27 = (W25 - W27)/2;
W29 = (W28 - W29)/2;   W30 = (W28 - W30)/2;
W32 = (W31 - W32)/2;   W33 = (W31 - W33)/2;

W31 -= W6; W25 -= W17; W28 -= W10;

W1   = (W1 - W3)/3  ; W5   = (W5 - W7)/3  ; W9   = (W9  - W11)/3;
W3   = (W2 - W3)/2  ; W7   = (W6 - W7)/2  ; W11  = (W10 - W11)/2;
W2  -=  W4          ; W6  -=  W8          ; W10 -=  W0;
W1   = (W1 - W2)/2  ; W5   = (W5 - W6)/2  ; W9   = (W9 - W10)/2;
W1  -= 2*W0         ; W5  -= 2*W4         ; W9  -= 2*W8;

W13  = (W13 - W15)/3; W16  = (W16 - W18)/3; W19  = (W19  - W21)/3;
W15  = (W14 - W15)/2; W18  = (W17 - W18)/2; W21  = (W20 - W21)/2;
W14 -=  W0          ; W17 -=  W4          ; W20 -=  W8;
W13  = (W13 - W14)/2; W16  = (W16 - W17)/2; W19  = (W19 - W20)/2;
W13 -= 2*W12        ; W16 -= 2*W12        ; W19 -= 2*W12;

W22 -= W2 ; W24 -= W3;
W14 -= W12;
W25 -= W14; W26 -= W18; W27 -= W15;
W29 -= W11; W28 -= W20; W30 -= W21;

W34 -= (W22 + W25 + W28 + W31);      \\ Inner point: 4 operations
```

```
W22 -= W10; W23 -= W11;
W31 -= W17; W32 -= W18; W33 -= W7;

W22 -= W23; W25 -= W26; W28 -= W29; W31 -= W32;

W2  -= W3 ; W6  -= W7 ; W10 -= W11;
W14 -= W15; W17 -= W18; W20 -= W21;
W17 -= W12; W20 -= W12

W22 -= W6;  W25 -= W2;  W28 -= W14; W31 -= W20;

W24 -= W22; W27 -= W25; W30 -= W28; W33 -= W31;
W23 -= W24; W26 -= W27; W29 -= W30; W32 -= W33;
W24 -= W7 ; W27 -= W3 ; W30 -= W15; W33 -= W21;

W2  -= W0 ; W3  -= W1 ; W6  -= W4 ;
W7  -= W5 ; W10 -= W8 ; W11 -= W9 ;
W15 -= W13; W18 -= W16; W21 -= W19;
```

5.3 *Quadridimensional case in characteristic different from 2 and 3*

In this case we have a four-dimensional tetrahedron, with $\binom{5}{1} = 5$ vertices, $\binom{5}{2} = 10$ sides, $\binom{5}{3} = \binom{5}{2} = 10$ faces and $\binom{5}{4} = 5$ pyramids. Due to the equal number of faces and sides, as in the tridimensional case it is still possible to associate the Π submatrices to the sides such in a way that no two faces sharing a side s both associate their Π to s.

Tag vertices with $\{0, 1, 2, 3, 4\}$ and consider them as numbers modulo 5. There are only two possible types of triangles (i, j, h):

$$\text{(A)}\ \ j = i + 1,\ h = j + 1 \qquad ; \qquad \text{(B)}\ \ j = i + 1,\ h = j + 2$$

We specify below Π associations and edges orientation (figure 5). As configurations (1 1 1 1) are used on sides, we must indicate which vertex the lacking 1 corresponds to (this affects also the evaluation phase, telling to which variable the value 2 is assigned).

	Π	Sides orientation
(A)	(i, h)	$j \to i\ ,\ h \to j\ ,\ h \to i$
(B)	(i, j)	$j \to i\ ,\ h \to j\ ,\ i \to h$

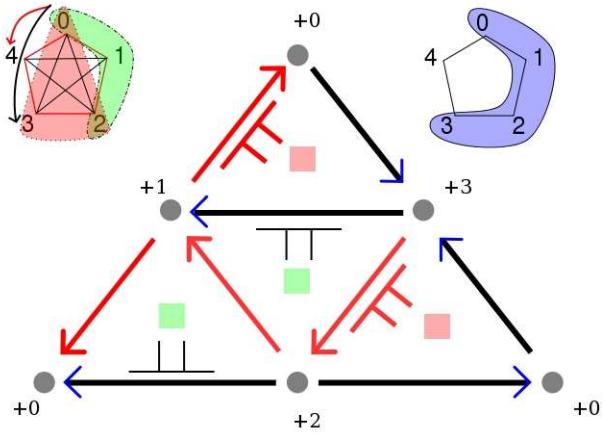


Figure 5: Sides orientation and Π association for quadridimensional Toom-3

After such a careful setting, the 70×70 resulting matrix $A_{4,4}$ can be inverted with an optimal number of operations. An implementation in `gp` code is reported in appendix C.

5.4 *The general balanced case in characteristic different from 2 and 3*

Formulae for quadratic factors get a bit more complicated than Toom-2.5 case, but still manageable. If

$$a(\overline{X}) = \sum_i a_i x_i^2 + \sum_{i<j} a_{ij} x_i x_j \qquad ; \qquad b(\overline{X}) = \sum_i b_i x_i^2 + \sum_{i<j} b_{ij} x_i x_j$$

then the product

$$\begin{aligned} c(\overline{X}) \;=\; & \sum_i (a_i b_i)\, x_i^4 + \sum_{i<j} (a_i b_{ij} + a_{ij} b_i)\, x_i^3 x_j + \sum_{i<j} (a_i b_j + a_{ij} b_{ij} + a_j b_i)\, x_i^2 x_j^2 \\ & + \sum_{i<j} (a_j b_{ij} + a_{ij} b_j)\, x_i x_j^3 + \sum_{i<j<h} (a_i b_{jh} + a_{ij} b_{ih} + a_{ih} b_{ij} + a_{jh} b_i)\, x_i^2 x_j x_h \\ & + \sum_{i<j<h} (a_j b_{ih} + a_{ij} b_{jh} + a_{jh} b_{ij} + a_{ih} b_j)\, x_i x_j^2 x_h \\ & + \sum_{i<j<h} (a_h b_{ij} + a_{ih} b_{jh} + a_{jh} b_{ih} + a_{ij} b_h)\, x_i x_j x_h^2 \\ & + \sum_{i<j<h<k} (a_{ij} b_{hk} + a_{ih} b_{jk} + a_{ik} b_{jh} + a_{jh} b_{ik} + a_{jk} b_{ih} + a_{hk} b_{ij})\, x_i x_j x_h x_k \end{aligned}$$

can be computed by precalculating, for all admissible values of p, q, r, s, the

quantities

$$\begin{array}{ll} W_p = a_p b_p & ;\ W^+_{pq} = (a_p + a_q + a_{pq})(b_p + b_q + b_{pq}) \\ W^{(2)}_{pq} = (a_p + 4a_q + 2a_{pq})(b_p + 4b_q + 2b_{pq}) & ;\ W^-_{pq} = (a_p + a_q - a_{pq})(b_p + b_q - b_{pq}) \end{array}$$

$$\begin{aligned} W^{(1)}_{pqr} &= (a_p + a_q + a_r + a_{pq} + a_{pr} + a_{qr})(b_p + b_q + b_r + b_{pq} + b_{pr} + b_{qr}) \\ W^{(2)}_{pqr} &= (a_p + a_q + a_r - a_{pq} + a_{pr} - a_{qr})(b_p + b_q + b_r - b_{pq} + b_{pr} - b_{qr}) \\ W^{(3)}_{pqr} &= (a_p + a_q + a_r + a_{pq} - a_{pr} - a_{qr})(b_p + b_q + b_r + b_{pq} - b_{pr} - b_{qr}) \end{aligned}$$

$$\begin{aligned} W^{rs}_{pq} = {} & (a_p + a_q + a_r + a_s + a_{pq} + a_{pr} + a_{ps} + a_{qr} + a_{qs} + a_{rs}) \times \\ & (b_p + b_q + b_r + b_s + b_{pq} + b_{pr} + b_{ps} + b_{qr} + b_{qs} + b_{rs}) \end{aligned}$$

The result is

$$\begin{aligned} c(\overline{X}) = {} & \sum_i W_i\, x_i^4 + \sum_{i<j} \left(W^+_{ij} + 2W_j - \frac{1}{6}\left(2W^-_{ij} + 3W_i + W^{(2)}_{ij}\right)\right) x_i^3 x_j \\ & + \sum_{i<j} \left(\frac{W^+_{ij} + W^-_{ij}}{2} - W_i - W_j \right) x_i^2 x_j^2 + \sum_{i<j} \left(\frac{W^{(2)}_{ij} - W^-_{ij} - 3(W^+_{ij} - W_i)}{6} - 2W_j \right) x_i x_j^2 \\ & + \sum_{i<j<h} \left(\frac{W^+_{ij} + W^-_{ij} + W^+_{ih} + W^-_{ih} - W^{(2)}_{ijh} - W^{(3)}_{ijh}}{2} + W^-_{jh} - W_i - W_j - W_h \right) x_i^2 x_j x_h \\ & - \sum_{i<j<h} \left(\frac{W_{ij} + W^-_{ij} + W_{jh} + W^-_{jh} - W^{(1)}_{ijh} - W^{(2)}_{ijh}}{2} - W^+_{ih} + W_i + W_j + W_h \right) x_i x_j^2 x_h \\ & - \sum_{i<j<h} \left(\frac{W_{ih} + W^-_{ih} + W_{jh} + W^-_{jh} - W^{(1)}_{ijh} - W^{(3)}_{ijh}}{2} - W^+_{ij} + W_i + W_j + W_h \right) x_i x_j x_h^2 \\ & + \sum_{i<j<h<k} \Big(W^{hk}_{ij} + W^+_{ij} + W^+_{jh} + W^+_{ih} + W^+_{ik} + W^+_{jk} + W^+_{hk} + \\ & \qquad - (W^{(1)}_{ijh} + W^{(1)}_{ijk} + W^{(1)}_{ihk} + W^{(1)}_{jhk} + W_i + W_j + W_h + W_k) \Big) x_i x_j x_h x_k \end{aligned}$$

◇ Evaluation

The general evaluation procedure in $\binom{n+3}{4}$ carefully chosen points requires not less effort than interpolation. We propose two different evaluations procedures: EV_1 and EV_2. The former is better in a model for which the *ma* operation described in section is not available, while the latter, taking benefit of the reduced number of memory accesses, is preferable when it is.

The general idea for both versions is to build the necessary evaluation values by recycling as much as possible already computed intermediate ones. We'll use $V_3 = \{0, 1, -1, 2\}$ as set of values to instantiate variables with: and

in particular $\{1\}$ for pyramids, $\{0, 1, -1\}$ for faces, the whole V_3 for sides and $\{0, 1\}$ for vertices. We'll once again use a geometrical setting in order to describe evaluation. The a_i and a_{ij} coefficients (and similarly for b) will be seen as vertices and internal points of sides of discrete faces and pyramids, representing interpolation points. To evaluate c in these discrete geometrical structures we then multiply two copies of each of them, one corresponding to a, the other one to b.

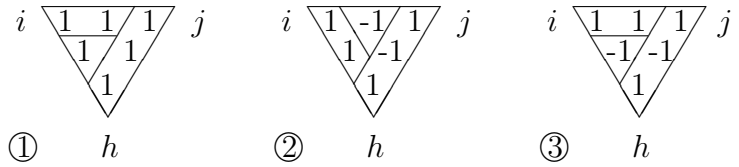
$\boxed{EV_1}$: In order to build the three faces configuration (one consisting of all 1 and two with some -1 in different positions) and the single type of pyramid (made by all 1), we use some temporary values that do not appear in the optimal sequence for sides found by Bodrato [1] when ma is not available, but permit to reduce to $O(n)$ the number of needed extra values. We use the following evaluation sequence on sides, whose cost is $(5A + \_1\_2) + S$, slightly suboptimal with respect to Bodrato's $(5A + \_1\_2)$.

1)	$v_3 = (1\ 0\ 0) + (0\ 1\ 0)$;	$[\ (1\ \ 1\ \ 0) \simeq a_i + a_{ij}$]
2)	$v_1 = v_3 + (0\ 0\ 1)$;	$[\ (1\ \ 1\ \ 1) \simeq a_i + a_{ij} + a_j$]
3)	$v_3 = v_3 + (0\ 1\ 0)$;	$[\ (1\ \ 2\ \ 0) \simeq a_i + 2a_{ij}$]
4)	$v_2 = (0\ 0\ 1) \ll 2$;	$[\ (0\ \ 0\ \ 4) \simeq 4a_j$]
5)	$v_3 = v_3 + v_2$;	$[\ (1\ \ 2\ \ 4) \simeq a_i + 2a_{ij} + 4a_j$]
6)	$v_2 = sa((0\ 1\ 0), v_1, -1)$;	$[\ (1\ \text{-}1\ \ 1) \simeq a_i - a_{ij} + a_j$]

This sequence has the advantage to include the configuration $ps_{ij} \simeq (1\ 1\ 0)$, which will be used to build also faces and pyramids. The tricky point is that the extra operation (step 4) is a shift *on a vertex*. This means that, if we compute the corresponding value and put it apart, we have the chance to recycle this value for all sides having it as second vertex, considering thus only a linear number of extra operations. Note that steps 4) and 5) could be fused into a single step 4-5) $v_3 = sa((0\ 0\ 1), v_3, 2)$, with a sequence cost of $(5A + \_1\_2) + \_1\_2$. To take benefit of both possibilities, we must consider the relationship between the execution time taken by a $\_1\_2$ and by a shift, depending on the implementation. The threshold $t = \lfloor S/(\_1\_2) \rfloor$ indicates when it is convenient to do some (a certain number $\leq t$) $\_1\_2$ operations instead of a single shift. Depending of t, we choose the value of i such that for all sides whose second vertex has index $\leq i$ we use the 4-5) step, while for all other ones we use steps 4) and 5), recycling the (0 0 4) vertex configuration. When sa is not available, we then have to minimize the total number of operations,

and this is achieved considering $t = 2$.

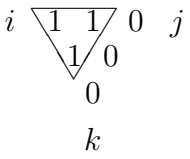
For faces, we obtain the three following configurations by building them as below subdivisions indicate:



① : (1 1 1) on side (j,h) + ps_{ij} + inner point of side (i,h)
② : (1 -1 1) on side (j,h) + ps_{ih} − inner point of side (i,j)
③ : (1 -1 1) on side (j,h) + ps_{ij} − inner point of side (i,h)

They are obtained by analyzing sides starting from the "higher" ones (with bigger vertex indexes), so that just $2 \cdot 3 = 6$ additions/subtractions per factor are needed for faces.

For pyramids the idea is very similar: we use the (supposed already computed) face (j,h,k) configuration ① , the internal point on side (i,h) and the aside shown boxed pf_{ijk} partial (i,j,k) configuration, which is intermediately needed and obtained while computing faces.



Working this way, we build pyramids with just 2 additions per factor. The total complexity is then

A	S	$\_1\_2$
$2\left[2\binom{n}{4} + 6\binom{n}{3} + 5\binom{n}{2}\right]$	$2(n-t)$	$2\left[\binom{n}{2} + \binom{t}{2}\right]$

$\boxed{EV_2}$: We consider the following evaluation sequence for sides, with a total cost of $(4A + \_1\_2 + \_1\_3)$, more efficient when sa, ma are both available:

1) $v_3 = (0\ 1\ 0) + (0\ 0\ 1)$; [$(0\ \ 1\ \ 1) \simeq a_{ij} + a_j$]
2) $v_1 = v_3 + (1\ 0\ 0)$; [$(1\ \ 1\ \ 1) \simeq a_i + a_{ij} + a_j$]
3) $v_2 = sa((0\ 1\ 0), v_1, -1)$; [$(1\ \text{-}1\ \ 1) \simeq a_i - a_{ij} + a_j$]
4) $v_3 = ma(v_3, v_2)$; [$(1\ \ 2\ \ 4) \simeq a_i + 2a_{ij} + 4a_j$]

By using this basic evaluation sequence, it is possible to set up the general one *without any extra operation*. Infact, for what concern faces we obtain configurations ① and ② by means of (supposed already available) proper value of (i, j) side, of $ps'_{ih} = (0\ 1\ 1)$ configuration and of the internal point on side (j, h). For ③ we consider instead the complete side (i, h) together with ps'_{ij} and the remaining middle-point on side (j, h).

This leads to a someway tricky order for values computations, different from the one used for EV_1. For example, almost all pf^+_{ijh} (all but the ones with $h = j + 1$) configurations must be precomputed in advance, and one must be very shrewd in using the result space in order to keep all intermediate needed values. Our implementation, to keep code sufficiently compact, uses only 6 extra temporaries, independently from n. The total complexity is

A	_1_2	_1_3
$2\left[2\binom{n}{4} + 6\binom{n}{3} + 4\binom{n}{2}\right]$	$2\binom{n}{2}$	$2\binom{n}{2}$

◇ Interpolation

Similarly to what happens for general Toom-2.5 method, apart from the particular cases of the precedent sections, it is not possible to have optimal inversion sequences for all pyramids. Infact, when $n > 5$, we have that $\binom{n}{2} < \binom{n}{3}$, and this means that there is at least one side to which we should associate at least two Π.

The general case should then use another inversion sequence on some faces, still permitting to have 4 operations to obtain inner points for pyramids, that costs one addition more. As for Toom-2.5, it is still possible to have a single quadridimensional tetrahedron on which the optimal inversion sequence can be maintained.

Moreover, the threshold t makes a difference here, too. The optimal inversion sequence for one side (classical Toom-3) by Bodrato and Zanoni [2] includes the following operation

$$(2\ 1\ 0\ 0\ 0) - 2(1\ 0\ 0\ 0\ 0)$$

Note that the subtrahend configuration is vertex-depending. We're facing exactly the same situation as in evaluation phase: to use some _1_2 or to

compute a single shift and recycle it for all the sides it can be used with ? Once again, the threshold t enters its game, and both possibilities are used according to its value. We then have

A	S	D	_1_2
$4\binom{n}{4} + 11\binom{n}{3} + \binom{n-1}{3} + 8\binom{n}{2}$	$2\binom{n}{3} + 2\binom{n}{2} + (n-t)$	$\binom{n}{2}$	$\binom{t}{2}$

Appendix D contains `gp` code of two functions performing evaluation and interpolation: `multivariateToom3` uses EV_1, while `multivariateToom3_bis` uses EV_2 evaluation. In order to avoid complications due to indexes treatment, both functions do not include the optimization obtainable considering a single quadridimensional tetrahedron configured as explained in section 5.3. The gain would be very small, and by not considering it we obtain a much cleaner code.

5.5 *The general unbalanced case in characteristic different from 2, 3*

For unbalanced operands, formulae and evaluation procedure change, while interpolation phase does not. We have

$$a(\overline{X}) = \sum_i a_i x_i^3 + \sum_{i\neq j} a_{ij} x_i^2 x_j + \sum_{i<j<h} a_{ijh} x_i x_j x_h \qquad ; \qquad b(\overline{X}) = \sum_i b_i x_i$$

then the product

$$\begin{aligned} c(\overline{X}) = & \sum_i (a_i b_i)\, x_i^4 + \sum_{i<j} (a_i b_j + a_{ij} b_i)\, x_i^3 x_j + \sum_{i<j} (a_{ij} b_j + a_{ji} b_i)\, x_i^2 x_j^2 \\ & + \sum_{i<j} (a_j b_i + a_{ji} b_j)\, x_i x_j^3 + \sum_{i<j<h} (a_{ij} b_h + a_{ih} b_j + a_{ijh} b_i)\, x_i^2 x_j x_h \\ & + \sum_{i<j<h} (a_{ji} b_h + a_{jh} b_i + a_{ijh} b_j)\, x_i x_j^2 x_h + \sum_{i<j<h} (a_{hi} b_j + a_{hj} b_i + a_{ijh} b_h) x_i x_j x_h^2 \\ & + \sum_{i<j<h<k} (a_{ijh} b_k + a_{ijk} b_h + a_{ihk} b_j + a_{jhk} b_i)\, x_i x_j x_h x_k \end{aligned}$$

can be computed by precalculating, for all admissible values of p, q, r, s, the quantities

$$\begin{aligned} W_p &= a_p b_p & ; \; W_{pq}^{+} &= (a_p + a_q + a_{pq} + a_{qp})(b_p + b_q) \\ W_{pq}^{(2)} &= (a_p + 8a_q + 4a_{pq} + 2a_{qp})(b_p + 4b_q) & ; \; W_{pq}^{-} &= (a_p - a_q - a_{pq} + a_{qp})(b_p - b_q) \end{aligned}$$

$$
\begin{aligned}
W^{(1)}_{pqr} &= (a_p + a_q + a_r + a_{pq} + a_{qp} + a_{pr} + a_{rp} + a_{qr} + a_{rq} + a_{pqr})(b_p + b_q + b_r)\\
W^{(2)}_{pqr} &= (a_p - a_q + a_r - a_{pq} + a_{qp} + a_{pr} + a_{rp} + a_{qr} - a_{rq} - a_{pqr})(b_p - b_q + b_r)\\
W^{(3)}_{pqr} &= (a_p + a_q - a_r + a_{pq} + a_{qp} - a_{pr} + a_{rp} - a_{qr} + a_{rq} - a_{pqr})(b_p + b_q - b_r)
\end{aligned}
$$

$$
\begin{aligned}
W^{rs}_{pq} = (&a_p + a_q + a_r + a_s + a_{pq} + a_{qp} + a_{pr} + a_{rp} + a_{ps} + a_{sp} + a_{qr} + a_{rq} +\\
&a_{qs} + a_{sq} + a_{rs} + a_{sr} + a_{pqr} + a_{pqs} + a_{prs} + a_{qrs})(b_p + b_q + b_r + b_s)
\end{aligned}
$$

and express $c(\overline{X})$ as in section 5.4.

The idea for evaluation is very close to the one for the balanced case, and is similarly treated by looking at the longest factor, a, from a geometrical point of view. First of all, notice that, depending on *ma* availability, there are two different optimal evaluation sequences for it. If only *sa* if available we have a cost of $7A + 3\,(\_1\_2)$.

1)	$v_2 = (1\ 0\ 0\ 0) + (0\ 0\ 1\ 0)$;	$[\ (1\ \ 0\ \ 1\ \ 0) \simeq a_i + a_{ji}$]
2)	$v_3 = (0\ 1\ 0\ 0) + (0\ 0\ 0\ 1)$;	$[\ (0\ \ 1\ \ 0\ \ 1) \simeq a_j + a_{ij}$]
3)	$v_1 = v_2 + v_3$;	$[\ (1\ \ 1\ \ 1\ \ 1) \simeq a_i + a_{ij} + a_{ji} + a_j$]
4)	$v_2 = v_2 - v_3$;	$[\ (1\ \text{-}1\ \ 1\ \text{-}1) \simeq a_j - a_{ji} + a_{ji} - a_j$]
5)	$v_3 = sa((0\ 0\ 0\ 1), (0\ 0\ 1\ 0), 1)$;	$[\ (0\ \ 0\ \ 1\ \ 2) \simeq a_{ji} + 2a_j$]
6)	$v_3 = sa(v_3, (0\ 1\ 0\ 0), 1)$;	$[\ (0\ \ 1\ \ 2\ \ 4) \simeq a_{ij} + 2a_{ji} + 4a_j$]
7)	$v_3 = sa(v_3, (1\ 0\ 0\ 0), 1)$;	$[\ (1\ \ 2\ \ 4\ \ 8) \simeq a_i + 2a_{ij} + 4a_{ji} + 8a_j$]

If *ma* is also available, we may also obtain $7A + \_1\_2 + \_1\_3$ with

1)	$\cdots$ 4) as above	
5)	$t = sa((0\ 0\ 0\ 1), (0\ 0\ 1\ 0), 1)$;	$[\ (0\ \ 0\ \ 1\ \ 2) \simeq a_{ji} + 2a_j$]
6)	$v_3 = v_3 + v_1$;	$[\ (1\ \ 2\ \ 1\ \ 2) \simeq a_i + 2a_{ij} + a_{ji} + 2a_j$]
7)	$v_3 = ma(t, v_3)$;	$[\ (1\ \ 2\ \ 4\ \ 8) \simeq a_i + 2a_{ij} + 4a_{ji} + 8a_j$]

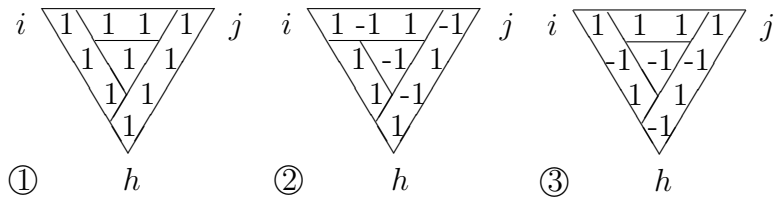
and one must check if $\_1\_3 \leq 2\,(\_1\_2)$ to determine which is the optimal sequence.

In order to optimally treat faces and pyramids, a different evaluation sequence for sides must be considered, whose cost is, luckyly, not much greater: $7A + 2\,(\_1\_2) + \_1\_3$.

1)	$v_3 = (0\ 1\ 0\ 0) + (0\ 0\ 1\ 0)$;	$[\ (0\ \ 1\ \ 1\ \ 0) \simeq a_{ij} + a_{ji}$]
2)	$v_1 = v_1 + (1\ 0\ 0\ 0)$;	$[\ (1\ \ 1\ \ 1\ \ 0) \simeq a_i + a_{ij} + a_{ji}$]
3)	$v_2 = sa((0\ 1\ 0\ 0), v_1, -1)$;	$[\ (1\ \text{-}1\ \ 1\ \ 0) \simeq a_i - a_{ij} + a_{ji}$]
4)	$v_1 = v_1 + (0\ 0\ 0\ 1)$;	$[\ (1\ \ 1\ \ 1\ \ 1) \simeq a_j + a_{ij} + a_{ji} + a_j$]
5)	$v_3 = ma(v_3, v_2)$;	$[\ (1\ \ 2\ \ 4\ \ 0) \simeq a_i + 2a_{ij} + 4a_{ji}$]
6)	$v_2 = v_2 - (0\ 0\ 0\ 1)$;	$[\ (1\ \text{-}1\ \ 1\ \text{-}1) \simeq a_i - a_{ij} + a_{ji} - a_j$]
7)	$v_3 = sa((0\ 0\ 0\ 1), v_3, 3)$;	$[\ (1\ \ 2\ \ 4\ \ 8) \simeq a_i + 2a_{ij} + 4a_{ji} + 8a_j$]

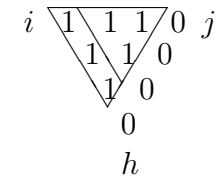
As in EV_1, operation 7 can be split into a shift on the (0 0 0 1) vertex-depending configuration and an addition, with threshold t playing here an important role, too.

Let $ps^*_{ij} = (0\ 1\ 1\ 0)$, $ps^+_{ij} = (1\ 1\ 1\ 0)$ and $ps^-_{ij} = (1\ \text{-}1\ 1\ 0)$ the three useful intermediate partial side configurations. Faces have now an inner point p_{ijh}, and are build as follows, with 3 algebraic sums:



$$\text{①} : ps^*_{ij} + (1\ \ 1\ \ 1\ \ 1) \text{ on side } (j,h) + ps^+_{ih} + p_{ijh}$$
$$\text{②} : ps^*_{ih} - (1\ \text{-}1\ \ 1\ \text{-}1) \text{ on side } (j,h) + ps^-_{ij} - p_{ijh}$$
$$\text{③} : ps^*_{ij} + (1\ \text{-}1\ \ 1\ \text{-}1) \text{ on side } (j,h) + ps^-_{ih} - p_{ijh}$$

The intermediate shown aside addends $pf^*_{ijh} = ps_{ij} + p_{ijh}$ (a partial face triangle) and $pf^\Delta_{ijh} = pf^*_{ijh} + ps^+_{ih}$ (a face minus the "higher" side) are recycled for the pyramids, which have no inner point and are built summing the ① (j,h,k) face configuration, pf^Δ_{ihk}, pf^*_{ijh} and the face inner point p_{ijk}.



The more elaborated evaluation of a and easier of b gives a total complexity of

A	S	_1_2	_1_3
$2\left[2\binom{n}{4} + 6\binom{n}{3} + 5\binom{n}{2}\right]$	$(n-t)$	$\left[\binom{n}{2} + \binom{t}{2}\right]$	$\binom{n}{2}$

An implementation in `gp` code is reported in appendix E.

6. Issues on higher Toom methods

In order to consider Toom-k methods with $k > 3$, it is important to give a closer look to the discrete geometrical structure. The setting for the n-variate case (not counting the homogenizing variable) is the discrete n-dimensional hyper-tetrahedron $\mathcal{HT} = \{\alpha \in \mathbb{N}^n \mid |\alpha| \leq 2k-2\}$ standing for the triangular representation. We define the following quantities in terms of n and k:

$$T = \binom{n+2k-2}{n} \quad ; \quad I = \binom{2k-3}{n} \quad ; \quad F = \binom{n+2k-3}{n-1}$$

They represent, the number of points of $\mathcal{HT}$, the number of its internal points and the number of points on a face, respectively (with $I = 0$ if $2k-3 < n$). In particular, as already observed, $n = 1$ are the classical Toom-k methods, which geometrically correspond to "segments". For $n = 2$ we have triangles, which are analyzed - as we have seen - first considering sides (unidimensional segments) and then internal points. Similarly, the general case is treated considering first the n (hyper-)faces of dimension $n-1$ forming the border, and then the internal points.

Suppose to order all possible $\overline{Y}$-terms forming $\mathcal{HT}$ in a vector as follows

$$L_{n,k} = (T_1, \ldots, T_q, T_{q+1}, \ldots, T_t) \quad \text{so that} \quad \begin{cases} \exists\ j\ :\ y_j \not| \ T_i & 1 \leq i \leq q \\ \forall\ j\ :\ y_j \mid T_i, & i > q \end{cases}$$

and that we already solved all Toom-k up to the $(n-1)$-dimensional one. Then one can obtain a n-dimensional interpolation matrix $A_{n,k}$ with a shape schematically shown here, where $A'_{n-1,k}$ is recursively obtained by setting n lines as "singletons", with a single 1 and 0 elsewhere (corresponding to vertices), and then, taking care of the ordering given by terms indexing columns,

$$A_{n,k} = \left(\begin{array}{c|c} A'_{n-1,k} & 0 \\ \hline \cdots & \cdots \end{array}\right)$$

by $\binom{n}{i}\binom{2k-3}{i}$ lines for every $1 \leq i < n$, corresponding to inner points on lower dimensional dihedrons.

To define the last I lines of the interpolation matrix it is sufficient to evaluate $L_{n,k}$ in I interpolation points with no zero coordinate such that the square submatrix M_I formed by the last I columns of the resulting $T \times I$ matrix is invertible.

Let $V^k_{n-1} = \{v_0, \ldots, v_l\}$ be the set of integers appearing as coordinates in the $(n-1)$-dimensional points used for faces in a multivariate Toom-k method.

In order to make matrix inversion easier, the most natural idea for values to use as coordinates of interpolation points is to consider $v_i \in V_{n-1}^k \setminus \{0\}$ - those already considered for faces. But can we find I points such that $V_n^k = V_{n-1}^k$ and M_I is invertible ? It is not obvious neither if it is always possible nor, if it is, how to determine them. And if it is not possible, when will $V_{n-1}^k \subset V_n^k$ happen ?

No zero coordinate means that $y_i = 0$ never happens for any i (this would mean we're working on a face, and we've already considered all matrix lines given by faces). Moreover, all terms having the same degree, the two points $P_1 = (v_i, \ldots, v_i)$ and $P_2 = (v_j, \ldots, v_j)$ give two linearly dependent lines, so that just one of them may be chosen.

If we need I values, a necessary condition to have $V_{n-1}^k = V_n^k$ is the following one:

$$\binom{2k-3}{n-1} \leq l^n - l + 1$$

6.1 *Considerations on higher Toom methods*

Because of the binomial "nature" of the above quantities, the involved interpolation matrix dimensions grow very fast as n and k grow. For example, for the bidimensional Toom-4 case (triangle) we have 28 points, of which 10 are inner ones. In this case (differently from the case for bidimensional Toom-3) we *must* use the value 2 to obtain a sufficient number of valid inner interpolation points, and this complicates the inversion procedure very much. Already in this case, the number of needed operations to invert the matrix is very likely to hide the benefit of performing less multiplications. Even if asymptotically convenient in theory, we remind that Toom methods are the fastest ones only in a limited range, and when the overhead given by the extra operations is too much, then Kronecker+FFT or some other asymptotically better algorithm is preferred to enter the game. With bigger values of n and k, the situation gets even worse, and we fear that multivariate Karatsuba, Toom-2.5 and Toom-3 are the only generalizations that can really prove to be effective in practical applications.

7. Complexity analysis

Let $T_k^{(2)}(d)$ be the number of multiplications of the bidimensional Toom-k method for a triangle with side length d, and $S_k^{(2)}(d)$ for a square. We want to obtain the explicit formula expressing $T_k^{(2)}(d)$.

One could benefit of the graphical approach: draw $k-1$ vertical and horizontal lines dividing the legs in k equal parts each. The factor triangles are then divided into k small triangles and $k(k-1)/2$ small squares, and the resulting (big) product triangle into $2k-1$ small triangles and $(2k-1)(2k-2)/2 = (2k-1)(k-1)$ small squares (see figure 6 for the case $k=5$).

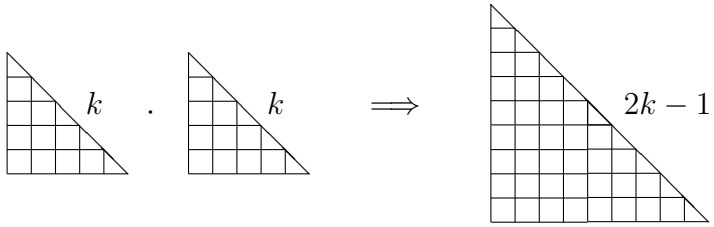


Figure 6: Recursive subdivision process ($k=5$)

Similar considerations for squares tell that two square factors splitted according to k make a big square made of $(2k-1)^2$ small squares. We set $n=2$, $\alpha = 2k-1$, $\beta = k-1$ and $\gamma = \alpha^2 = (2k-1)^2$. For simplicity, suppose $d = k^r$ for a certain $1 < r \in \mathbb{N}$. We have that the recursive formulae become

$$\begin{cases} T_k^{(n)}(d) = \alpha\, T\left(\frac{d}{k}\right) + \alpha\beta S\left(\frac{d}{k}\right) = \alpha\left[T\left(\frac{d}{k}\right) + \beta S\left(\frac{d}{k}\right)\right] \\ S_k^{(n)}(d) = \gamma S\left(\frac{d}{k}\right) \end{cases}$$

Expanding recursively the right sides we obtain

$$\begin{aligned} T_k^{(n)}(d) &= \alpha\left[T_k^{(n)}\left(\frac{d}{k}\right) + \beta S_k^{(n)}\left(\frac{d}{k}\right)\right] \\ &= \alpha\left[\alpha\left[T_k^{(n)}\left(\frac{d}{k^2}\right) + \beta S_k^{(n)}\left(\frac{d}{k^2}\right)\right] + \beta\gamma S_k^{(n)}\left(\frac{d}{k^2}\right)\right] \\ &= \alpha\left[\alpha\, T_k^{(n)}\left(\frac{d}{k^2}\right) + \beta(\alpha+\gamma) S_k^{(n)}\left(\frac{d}{k^2}\right)\right] \\ &= \alpha^2\left[T_k^{(n)}\left(\frac{d}{k^2}\right) + \beta\left(1+\frac{\gamma}{\alpha}\right) S_k^{(n)}\left(\frac{d}{k^2}\right)\right] \\ &= \alpha^2\left[\alpha\left[T_k^{(n)}\left(\frac{d}{k^3}\right) + \beta S_k^{(n)}\left(\frac{d}{k^3}\right)\right] + \beta\left(1+\frac{\gamma}{\alpha}\right)\gamma S_k^{(n)}\left(\frac{d}{k^3}\right)\right] \end{aligned}$$

$$
\begin{aligned}
&= \alpha^3 \left[T_k^{(n)} \left(\frac{d}{k^3} \right) + \beta \Big(\sum_{i=0}^{2} \left(\frac{\gamma}{\alpha} \right)^i \Big) S_k^{(n)} \Big(\frac{d}{k^3} \Big) \right] \\
&\ \ \vdots \\
&= \alpha^e \left[T_k^{(n)} \left(\frac{d}{k^e} \right) + \beta \frac{\left(\frac{\gamma}{\alpha} \right)^e - 1}{\left(\frac{\gamma}{\alpha} \right) - 1} S_k^{(n)} \Big(\frac{d}{k^e} \Big) \right]
\end{aligned}
$$

When $e = r$ we have $d/k^e = 1$, and, considering $T_k^{(n)}(1) = S_k^{(n)}(1) = 1$,

$$
T_k^{(n)}(d) = \alpha^r \left[1 + \beta \, \frac{\left(\frac{\gamma}{\alpha} \right)^r - 1}{\left(\frac{\gamma}{\alpha} \right) - 1} \right] = \alpha^r + \frac{\alpha\beta}{\gamma - \alpha} (\gamma^r - \alpha^r)
$$

Note that $\gamma/\alpha = \alpha$ and $\alpha^r = k^{\log_k \alpha^r} = k^{r \log_k \alpha} = (k^r)^{\log_k \alpha} = d^{\log_k \alpha}$. Substituting the original values for α, β we have

$$
\begin{aligned}
T_k^{(n)}(d) \ &= \alpha^r \Big[1 + \beta \, \frac{\alpha^r - 1}{\alpha - 1} \Big] = \alpha^r \left[1 + (k-1) \frac{\alpha^r - 1}{(2k-1) - 1} \right] \\
&= \alpha^r \left[1 + (k-1) \frac{\alpha^r - 1}{2(k-1)} \right] = \alpha^r \Big[1 + \frac{\alpha^r - 1}{2} \Big] = \alpha^r \Big[\frac{\alpha^r + 1}{2} \Big] \\
&= \frac{(d^2)^{\log_k (2k-1)} + d^{\log_k (2k-1)}}{2} = O((d^2)^{\log_k (2k-1)})
\end{aligned}
$$

Remembering that ${}^{\#}p = O(d^2)$, we found again Toom complexity in terms of input data cardinality. This happens because (see figure 6) the number of squares grows quadratically at each recursion step, while the number of triangles only linearly, and the potential benefit is soon lost. Anyway, we point out the presence of the constant $\frac{1}{2}$.

7.1 *The general case*

The above considerations generalize to the n-dimensional case, with multivariate polynomials $p \in \mathbf{R}[X]$. The geometric idea is similar: consider dihedrons and hypercubes and define a recursive formula. For p dense, ${}^{\#}p = \binom{n+d}{n}$ or ${}^{\#}p = (d+1)^n$ considering total and max degree, respectively, equal to d. In any case ${}^{\#}p = O(d^n)$.

The constants α and $\delta = \alpha\beta$ are in general determined by combinatorial expressions, representing the number of small triangles ("on the hypotenuse")

and of small squares (the inner part) of the product triangle. We have $\delta = \binom{n+2k-3}{n}$, so that

$$\alpha = \binom{n+2k-3}{n-1} \quad , \quad \beta = \frac{\delta}{\alpha} = \frac{2(k-1)}{n} \quad , \quad \gamma = (2k-1)^n$$

The complexity analysis is exactly the same, with these new α, β, γ values. As polynomials in k we have that α has (total) degree $n-1$, while γ has degree n. This means that

$$m = \frac{\alpha\beta}{\gamma - \alpha} = O(1) \implies \begin{aligned} T_k^{(n)}(d) &= O\left(\alpha^r + \frac{\alpha\beta}{\gamma-\alpha}(\gamma^r - \alpha^r)\right) \\ &= O(\alpha^r + \gamma^r - \alpha^r) = O(\gamma^r) \end{aligned}$$

Being

$$\gamma^r = k^{\log_k((2k-1)^n)^r} = k^{r\log_k(2k-1)^n} = (k^r)^{n\log_k(2k-1)} = (d^n)^{\log_k(2k-1)}$$

we find again the complexity of Toom-k method $T_k^{(n)}(d) = O((d^n)^{\log_k(2k-1)})$.

Summing all up, by using "schoolbook" multivariate multiplication method (that we can consider to be Toom-1), every monomial of $a(X)$ must be multiplied with every monomial of $b(X)$. The number of needed multiplications is then

$$\binom{n+d}{d}^2 = \binom{d+n}{n}^2 = \left(\frac{d^n + \cdots}{n!}\right)^2 \simeq \frac{d^{2n}}{(n!)^2} = O((d^n)^2)$$

and $m_{n,1} = (n!)^{-2}$ is the value of the multiplicative constant. Using Toom-k method, we instead have that

$$\begin{aligned} m_{n,k} &= \frac{\beta}{\frac{\gamma}{\alpha} - 1} = \frac{2(k-1)}{n\left[\frac{(2k-1)^n}{\alpha} - 1\right]} \\ &= \frac{2}{n}\frac{(2k+(n-3))(2k+(n-4))\cdots 2k(k-1)}{(n-1)!(2k-1)^{n-1} - (2k+(n-3))(2k+(n-4))\cdots 2k} \end{aligned}$$

Note that the denominator vanishes for $k = 1$, so that the factor $(k-1)$ can be simplified. Some particular expression for small values of n are reported below. It is straightforward to prove that $\lim_{k\to\infty} m_{n,k} = 1/n!$

n	2	3	4	5
$m_{n,k}$	$\frac{1}{2}$	$\frac{2k}{3(4k-1)}$	$\frac{k(2k+1)}{2(24k^2-14k+3)}$	$\frac{4k(k+1)(2k+1)}{5(96k^3-98k^2+43k-6)}$

8. Conclusions

We proposed a generalization of Karatsuba and some of Toom-Cook methods to multivariate polynomials having dense representation with respect to usual degree definition. We presented in full details algorithms computing evaluation and interpolation for Toom-2.5 and Toom-3. They should fit between usual (for not too small degrees) and asymptotically better (for not too big ones) multiplication methods, with a better behavior given by the optimization of the complexity constant, when factor degrees lie inside some interval depending on implementation. We think that higher Toom-Cook methods have small chance to be efficient in practise, because of computational overhead fastly growing as the number of variables and subdivisions grows.

Appendix A. Multivariate Toom-2.5 in characteristic different from 2

```
\\ (C) 2011 Marco Bodrato <http://marco.bodrato.it/>
\\ This code is released under GPL 3.0 licence.

Toom25(n=10)={
 n++;
 U = sum(i=1,n, sum(j=i,n , eval(Str("U",i,"_",j,"*x",i,"*x",j)) ) );
 V = sum(i=1,n  , eval(Str("V",i,"*x",i)) );

 \\ P(xi,xj,xk): P0=(1,0,0); P1=(1,0,1); P2=(1,0,-1); P3=(1,1,1)
 \\ Evaluation:
 sums = 0;shifts = 0;muls = 0;
 Wext = matrix(n, n);
 Wint = vector(n,k,if(k>2,matrix(k,k+1),0));

 for( i=1, n-2,
   forstep( j=i+2, n, 2,
     Wext[i,i]=eval(Str("U",j,"_",j,"+U",i,"_",j));sums++;
     for(k=1,n,
       if(k>j,Wint[k][i,j]=Wext[i,i]+eval(Str("U",j,"_",k));sums++;,
        if((k+i)%2==1,sums++;
          if(k>i,Wint[j][i,k]=Wext[i,i]+eval(Str("U",k,"_",j)),
                 Wint[j][k,i]=Wext[i,i]+eval(Str("U",k,"_",j))))));
     Wext[i,i]+=eval(Str("U",i,"_",i));sums++;
     Wext[j,j] =eval(Str("V",j,"+V",i));sums++;
     for(k=i+1,j-2,k++;
       Wext[k,k] =Wint[j][i,k]+Wext[i,i];sums++;
       Wext[i,j] =Wext[j,j]+eval(Str("V",k));sums++;
       Wint[j][i,k]=Wext[k,k]*Wext[i,j];muls++;);
     Wext[i,j] =Wext[j,j]*Wext[i,i];muls++;
     Wext[i,i]-=2*eval(Str("U",i,"_",j));sums++;shifts++;
     Wext[j,j] =if(j==n&i==n-2,eval(Str("V",i,"-V",j)),eval(Str("V",j,"-V",i)));sums++;
     Wext[j,i] =Wext[j,j]*Wext[i,i];muls++;
 ));
```

```
for( i=1, n,
  forstep( j=i+1, n, 2,
    Wext[j,i] =eval(Str("U",j,"_",j,"+U",i,"_",i));sums++;
    Wext[i,i] =Wext[j,i]+eval(Str("U",i,"_",j));sums++;
    Wext[j,j] =eval(Str("V",j,"+V",i));sums++;
    forstep(k=i+2,j,2,
      Wext[k,k] =Wint[j][i,k]+Wext[i,i];sums++;
      Wext[i,j] =Wext[j,j]+eval(Str("V",k));sums++;
      Wint[j][i,k]=Wext[k,k]*Wext[i,j];muls++;);
    for(k=j+1,n,
      Wext[k,k] =Wint[k][i,j]+Wext[i,i];sums++;
      Wext[i,j] =Wext[j,j]+eval(Str("V",k));sums++;
      Wint[k][i,j]=Wext[k,k]*Wext[i,j];muls++;);
    Wext[i,j] =Wext[j,j]*Wext[i,i];muls++;
    Wext[i,i] =Wext[j,i]-eval(Str("U",i,"_",j));sums++;
    Wext[j,j] =eval(Str("V",j,"-V",i));sums++;
    Wext[j,i] =Wext[j,j]*Wext[i,i];muls++;
  );
  Wext[i,i] =eval(Str("U",i,"_",i,"*V",i));muls++;
);
print("Evaluation required: ",sums," sums, ",shifts," shifts(_1_2),",muls," muls.");
print("Should be: ", binomial(n,2)*5+binomial(n,3)*3, " sums, ",binomial(ceil(n/2),2)
       +binomial(floor(n/2),2)," shifts, ",binomial(n+2,3)," muls.");
print("Naive: ", binomial(n,2)*6+binomial(n,3)*7, " sums, ",0," shifts.");

\\ Interpolation: 6 add, 2 shift, 1 div (2n)
sums=0;shifts=0;

for( i=1, n-3, for( j=i+1, n-1, for( k=j+1, n, Wint[k][i,j] -= Wext[j,k];sums++ )));

for( i=1, n-1, for( j=i+1, n,
  Wext[j,i] =(Wext[j,i] - Wext[i,j])/(-2);sums++;shifts++;
  if( j==n & i==n-2, Wext[i,j] = Wext[i,j] - Wext[i,i];sums++;
                   , \\else
                     Wext[i,j] = Wext[i,j] - Wext[j,j];sums++;
)));
for( i=1, n-3, for( j=i+1, n-1, for( k=j+1, n, Wint[k][i,j] -= Wext[i,k];sums++ )));
if( n-2>0,  Wint[n][n-2,n-1] -= Wext[n-2,n-1] + Wext[n-1,n] + Wext[n-2,n];sums+=3;);

for( i=1, n-3, for( j=i+1, n-1,  Wext[j,i] = Wext[j,i] - Wext[i,i];sums++;
                                 Wext[i,j] = Wext[i,j] - Wext[i,i];sums++; ));

for( i=1, n-3, for( j=i+1, n-1, for( k=j+1, n, Wint[k][i,j] -= Wext[i,j];sums++ )));
for( i=1, n-3, for( j=i+1, n-1, Wext[i,j] = Wext[i,j] - Wext[j,i];sums++ ));

for( i=1, n-1,
  Wext[i,n] = Wext[i,n] - Wext[n,i];sums++;
  if(i==n-2, Wext[n,i] = Wext[n,i] - Wext[n,n];sums++;
             Wext[i,n-1] = Wext[i,n-1] - Wext[n-1,i];sums++;
             Wext[n-1,i] = Wext[n-1,i] - Wext[i,i];sums++;
           ,
             Wext[n,i] = Wext[n,i] - Wext[i,i];sums++;
));
print("Interpolation required: ", sums, " sums, ", shifts, " shifts.");
print("Should be: ", binomial(n-1,2)-(n>2)+binomial(n,2)*4+binomial(n,3)*3,
      " sums, ", binomial(n,2), " shifts.");
```

```
 \\ Recomposition
 W = sum(i=1,n, sum(j=i+1,n, sum(k=j+1,n, Wint[k][i,j]*eval(Str("x",i,"*x",j,"*x",k)) )));
 W+= sum(i=1,n, sum(j=  1,n, if((i==n&j==n-2)||(j==n&i==n-2),
                                Wext[j,i], Wext[i,j])*eval(Str("x",i,"^2*x",j)) ));
 W == U*V
}
```

Appendix B. Multivariate Toom-2.5 in characteristic 2

```
\\ (C) 2011 Marco Bodrato <http://marco.bodrato.it/>
\\ This code is released under GPL 3.0 licence.

Toom25gf2(n=10,t=0)={
 n++; if(t<2|t>n,t=n);

 U = sum(i=1,n, sum(j=i,n , eval(Str("U",i,"_",j,"*x",i,"*x",j)) ) )*Mod(1,2);
 V = sum(i=1,n  , eval(Str("V",i,"*x",i)) )                          *Mod(1,2);

 \\ P(xi,xj,xk): P0=(1,0,0); P1=(1,0,1); P2=(1,0,x+1); P3=(1,1,1)
 \\ Evaluation:
 sums = 0;shifts = 0;addlsh = 0;muls = 0;
 Wext = matrix(n, n);
 Wint = vector(n,k,if(k>2,matrix(k,k+1),0));

 for(i=1, n-1, for( j=i+1, n,
     Wext[j,i]=eval(Str("U",j,"_",j,"+U",i,"_",j));sums++));
 forstep(j=n,1,-1,
   if(j>t,
     Wext[j,j]=x*eval(Str("V",j));shifts++;
     Wext[1,2]=x^2*eval(Str("U",j,"_",j));shifts++;
   );
   for(i=1,j-1,
     Wext[j,i]+=eval(Str("U",i,"_",i));sums++;
     Wext[2,2] =eval(Str("V",i,"+V",j));sums++;
     for(k=i+1,j-1,
       Wext[1,1]=Wext[j,i]+eval(Str("U",k,"_",j))+Wext[k,i];sums+=2;
       Wext[i,j]=Wext[2,2]+eval(Str("V",k));sums++;
       Wint[j][i,k]=Wext[1,1]*Wext[i,j]*Mod(1,2);muls++;
     );
     Wext[i,j] =Wext[2,2]*Wext[j,i]*Mod(1,2);muls++;
     if(j>t,
       Wext[2,2]+=Wext[j,j];
       Wext[1,1] =Wext[j,i]+x*eval(Str("U",i,"_",j))+Wext[1,2];
       sums+=3;addlsh+=1;
     ,\\else
       Wext[2,2]+=x*eval(Str("V",j));
       Wext[1,1] =Wext[j,i]+x*eval(Str("U",i,"_",j,"+x*U",j,"_",j));
       sums+=3;addlsh+=3;
     );
     Wext[j,i] =Wext[2,2]*Wext[1,1]*Mod(1,2);muls++;
   );
   Wext[j,j] =eval(Str("U",j,"_",j,"*V",j))*Mod(1,2);muls++;
 );
```

```
 print("Evaluation required: ",sums," sums, (",shifts,"+",addlsh,") shifts, ",
        muls, " muls.");
 print("Should be: ", binomial(n,2)*6+binomial(n,3)*3, " sums, (",2*(n-t),"+",
       binomial(n,2)+2*binomial(t,2), ") shifts, ",binomial(n+2,3)," muls.");
 print("Naive: ",binomial(n,2)*9+binomial(n,3)*7, " sums, ", 3*binomial(n,2), " shifts.");

 \\ Interpolation: 6 add, 2 shift, 1 div (2n)
 sums=0;shifts=0;addlsh=0;muls=0;Sdivs=0;Smuls=0;

 for( i=1, n-2, for( j=i+1, n-1, for( k=j+1, n,  Wint[k][i,j] -= Wext[i,j];sums++ )));

 for( i=1, n-1, for( j=i+1, n, Wext[j,i] =(Wext[j,i] - Wext[i,j])/(x);sums++;shifts++;
                               Wext[i,j] = Wext[i,j] - Wext[i,i];sums++; ));

 for( i=1, n-2, for( j=i+1, n-1, for( k=j+1, n,  Wint[k][i,j] -= Wext[j,k];sums++ )));

 for( j=2, n, if(j>t,T=Wext[j,j]*(x); shifts++); for( i=1, j-1,
   Wext[j,i] =(Wext[j,i] + Wext[i,j])/(x+1);sums++;Sdivs++;
   Wext[j,i] = Wext[j,i] - if(j>t,T,addlsh++;Wext[j,j]*(x));sums++;
   Wext[i,j] = Wext[i,j] - Wext[j,j];sums++;
 ));

 for( i=1, n-2, for( j=i+1, n-1, for( k=j+1, n, Wint[k][i,j] -= Wext[i,k];sums++ )));

 for( i=1, n-1, for( j=i+1, n,  Wext[i,j] = Wext[i,j] - Wext[j,i];sums++ ));

 print("Interpolation required: ", sums, " sums, (", shifts,"+",addlsh,
       ") shifts+1_2, ", Sdivs, " Sdivs.");
 print("Should be: ", binomial(n,2)*6+binomial(n,3)*3, " sums, (",
        binomial(n,2)+n-t,"+",binomial(t,2),") shifts+1_2, ", binomial(n,2), " Sdivs.");

 \\ Recomposition
 W = sum(i=1,n, sum(j=i+1,n, sum(k=j+1,n , Wint[k][i,j]*eval(Str("x",i,"*x",j,"*x",k)) )));
 W+= sum(i=1,n, sum(j=  1,n, Wext[i,j]*eval(Str("x",i,"^2*x",j)) ));

 W == U*V
}
```

Appendix C. Quadridimensional Toom-3 in characteristic different from 2 and 3

```
\\ (C) 2011 Marco Bodrato and Alberto Zanoni
\\ This code is released under GPL 3.0 licence.

QuadridimensionalToom3() =
{
 local(a,b, V,Sp,Sm,S2, Fp,Fm1,Fm2, T, i,j,h,k, add = 0, shifts = 0, div = 0, add2 = 0);

 \\\\\\\\\\\\ Evaluation \\\\\\\\\\\\\
 a = matrix(5, 5, i,j, if ( i < j, eval(Str("a", i,j)),
                        if (i==j, eval(Str("a", i)), eval(Str("a", j,i)))));
 b = matrix(5, 5, i,j, if ( i < j, eval(Str("b", i,j)),
                        if (i==j, eval(Str("b", i)), eval(Str("b", j,i)))));
 V  = vector(5,i,a[i,i]*b[i,i]);
 Sp = matrix(5, 5, i, j, (a[i,i]+a[j,j]+a[i,j])*(b[i,i]+b[j,j]+b[i,j]));
```

```
Sm = matrix(5, 5, i, j, (a[i,i]+a[j,j]-a[i,j])*(b[i,i]+b[j,j]-b[i,j]));
S2 = matrix(5, 5);
for (i = 1, 5, for (d = 1,2,
 j = i+d - if ((i+d) > 5, 5, 0);           \\ Sides: distance 1
 S2[i,j] = (4*a[i,i]+a[j,j]+2*a[i,j])*(4*b[i,i]+b[j,j]+2*b[i,j]); ));

Fp  = vector(5, i, matrix(5,5,j,h,(a[i,i]+a[j,j]+a[h,h]+a[i,j]+a[i,h]+a[j,h])*
                                  (b[i,i]+b[j,j]+b[h,h]+b[i,j]+b[i,h]+b[j,h])));
Fm1 = vector(5, i, matrix(5,5));
Fm2 = vector(5, i, matrix(5,5));    T = vector(5);

for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);        \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);        \\ Distance 2

   \\ To put pi on side (+0,+2) the -1 goes in position +0 and +2 for green triangles:
   \\ 3 possible cases (index rotation)
   Fm1[i][j,h] = (a[i,i]+a[j,j]+a[h,h]-a[i,j]-a[i,h]+a[j,h])*
                 (b[i,i]+b[j,j]+b[h,h]-b[i,j]-b[i,h]+b[j,h]);
   Fm2[i][j,h] = (a[i,i]+a[j,j]+a[h,h]+a[i,j]-a[i,h]-a[j,h])*
                 (b[i,i]+b[j,j]+b[h,h]+b[i,j]-b[i,h]-b[j,h]);

   h = i+3 - if ((i+3) > 5, 5, 0);                 \\ Distance 3

   \\ To put pi on side (+0,+1) the -1 goes in position +0 and +1 for red triangles:
   \\ 3 possible cases (index rotation)
   Fm1[i][j,h] = (a[i,i]+a[j,j]+a[h,h]-a[i,j]-a[i,h]+a[j,h])*
                 (b[i,i]+b[j,j]+b[h,h]-b[i,j]-b[i,h]+b[j,h]);
   Fm2[i][j,h] = (a[i,i]+a[j,j]+a[h,h]-a[i,j]+a[i,h]-a[j,h])*
                 (b[i,i]+b[j,j]+b[h,h]-b[i,j]+b[i,h]-b[j,h]);
);
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);        \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);        \\ Distance 2
            k = i+3 - if ((i+3) > 5, 5, 0);        \\ Distance 3
 T[i] = (a[i,i]+a[j,j]+a[h,h]+a[k,k]+a[i,j]+a[i,h]+a[i,k]+a[j,h]+a[j,k]+a[h,k])*
        (b[i,i]+b[j,j]+b[h,h]+b[k,k]+b[i,j]+b[i,h]+b[i,k]+b[j,h]+b[j,k]+b[h,k]);
);
\\\\\\\\\\\\ Interpolation \\\\\\\\\\\\\

\\ 1. Decoupling on faces.
for( i=1,5,     j = i+1 - if ((i+1) > 5, 5, 0);   \\ Sides: distance 1
  for (d = 2,3, h = i+d - if ((i+d) > 5, 5, 0);   \\ Sides: distance 2 and 3
  Fm1[i][j,h] = (Fp[i][j,h] - Fm1[i][j,h])/2; add++; shifts++;
  Fm2[i][j,h] = (Fp[i][j,h] - Fm2[i][j,h])/2; add++; shifts++;));

\\ 2. Remove complete sides.
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);       \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);       \\ Distance 2
 Fp[i][j,h] -= Sp[j,h];  add++;);

\\ 3. Work on sides...
for( i=1,5, for (d = 1,2,
  j = i+d - if ((i+d) > 5, 5, 0);        \\ Sides: distance 1
  S2[i,j] = (S2[i,j] - Sm[i,j])/3;  Sm[i,j] = (Sp[i,j] - Sm[i,j])/2;
  add++; div++; add++; shifts++;
  Sp[i,j] -= V[j]; S2[i,j] = (S2[i,j] - Sp[i,j])/2 - 2*V[i];
  add += 3; shifts++; add2++; ));
```

```
\\ 4. Inner points of faces.
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);       \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);       \\ Distance 2

   Fm1[i][j,h] -= Sm[i,j]; Fm2[i][j,h] -= Sm[j,h]; add++; add++;
   h = i+3 - if ((i+3) > 5, 5, 0);                \\ Distance 3

   Fm1[i][j,h] -= Sm[h,i]; Fm2[i][j,h] -= Sm[j,h]; Fp[i][j,h] -= Sp[h,i]; add += 3;);

\\ 5. Work on pyramids.
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);       \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);       \\ Distance 2
            k = i+3 - if ((i+3) > 5, 5, 0);       \\ Distance 3
 T[i] -= (Fp[h][k,i] + Fp[j][h,k]); add++; add++;);

\\ 6. Remove part of sides.
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);       \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);       \\ Distance 2
 Fp[i][j,h] -= Sp[i,j];  add++;
            h = i+3 - if ((i+3) > 5, 5, 0);       \\ Distance 3
 Fp[i][j,h] -= Sp[j,h];  add++;);

\\ 7. End working on pyramids.
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);       \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);       \\ Distance 2
            k = i+3 - if ((i+3) > 5, 5, 0);       \\ Distance 3
 T[i] -= (Fp[i][j,h] + Fp[i][j,k]); add++; add++;);

\\ 8. Continue working on sides.
for( i=1,5, for (d = 1,2, j = i+d - if ((i+d) > 5, 5, 0);
  Sp[i,j] -= Sm[i,j];    add++;));

\\ 9. Continue working on faces.
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);      \\ Distance 1
            h = i+3 - if ((i+3) > 5, 5, 0);      \\ Distance 3
  Fp[i][j,h] -= Sp[i,j]; add++;);

\\ 10. Continue sides.
for( i=1,5, for (d = 1,2, j = i+d - if ((i+d) > 5, 5, 0);
  Sp[i,j] -= V[i];       add++; ));

\\ 11. End faces.
for( i=1,5, j = i+1 - if ((i+1) > 5, 5, 0);      \\ Distance 1
            h = i+2 - if ((i+2) > 5, 5, 0);      \\ Distance 3
  Fp[i][j,h] -= Sp[i,h]; add++;);

for( i=1,5,     j = i+1 - if ((i+1) > 5, 5, 0);  \\ Sides: distance 1
  for (d = 2,3, h = i+d - if ((i+d) > 5, 5, 0);  \\ Sides: distance 2 and 3
 Fp [i][j,h] -= Fm1[i][j,h];
 Fm2[i][j,h] -= Fp [i][j,h];
 Fm1[i][j,h] -= Fm2[i][j,h];

 Fm2[i][j,h] -= if ( d == 2, Sm[i,h], Sm[i,j]); add += 4; ));

\\ 12. End sides.
```

```
 for( i=1,5, for (d = 1,2, j = i+d - if ((i+d) > 5, 5, 0);
   Sm[i,j] -= S2[i,j]; add++;));

 print("--------------- Final situation -----------------------------");
 print("Vertices = ", V);

 for( i=1,5, for (d = 1,2,
  j = i+d - if ((i+d) > 5, 5, 0);
  print("Side(+)[", i, ",", j, "] = ",Sp[i,j], "      Side(-)[", i, ",", j, "] = ",
        Sm[i,j],"      Side(2)[", i, ",", j, "] = ",S2[i,j])));

 for( i=1,5,     j = i+1 - if ((i+1) > 5, 5, 0);
   for( d = 2,3, h = i+d - if ((i+d) > 5, 5, 0);
     print("F+  [", i, ",", j, ",", h, "] = ",Fp[i][j,h]); ));

 for( i=1,5,     j = i+1 - if ((i+1) > 5, 5, 0);
   for( d = 2,3, h = i+d - if ((i+d) > 5, 5, 0);
     print("F-1 [", i, ",", j, ",", h, "] = ",Fm1[i][j,h]); ));

 for( i=1,5,     j = i+1 - if ((i+1) > 5, 5, 0);
   for( d = 2,3, h = i+d - if ((i+d) > 5, 5, 0);
     print("F-2 [", i, ",", j, ",", h, "] = ",Fm2[i][j,h]); ));
 for( i=1,5, print("T[",i,"] = ", T[i]));

 print("Expected:");
 print("add = ", 10*8+10*11+5*4,"  shifts = ",10*2+10*2,"  div = ",10, "  _1_2 = ",10);
 print("add = ", add, "  shifts = ",shifts,"  div = ",div, "  _1_2 = ",add2);
}
```

Appendix D. Multivariate Toom-3 in characteristic different from 2 and 3

The function `multivariateToom3` implements multivariate Toom-3 method when *ma* is not available, while `multivariateToom3_bis` when it is. The (common) interpolation section is detailed only for the first function.

```
multivariateToom3(n=5, t=0) =
{
 local(a,b, Sp,Sm,S2, Fp,Fm1,Fm2, T, i,j,h,k, Eadd = 0, Eshifts = 0,
       Eadd2=0, add = 0, shifts = 0, div = 0, add2 = 0);

 if(t<1|t>n,t=n);
 \\\\\\\\\\\\ Evaluation \\\\\\\\\\\\\
 a  = matrix(n, n, i,j, if ( i < j, eval(Str("a", i,j)),eval(Str("a", i))));
 b  = matrix(n, n, i,j, if ( i < j, eval(Str("b", i,j)),eval(Str("b", i))));

 Sp = matrix(n, n);   Fp  = vector(n, i, matrix(n,n));
 Sm = matrix(n, n);   Fm1 = vector(n, i, matrix(n,n));
 S2 = matrix(n, n);   Fm2 = vector(n, i, matrix(n,n));
 T = matrix(n,n,i,j,matrix(n,n));

 \\ Evaluate first partial sides.
 for( i=1, n-1, for( j = i+1, n,
```

```
  Sp[i,j] = a[i,i] + a[i,j];    Sm[i,j] = b[i,i] + b[i,j]; Eadd += 2;
  \\ Evaluate then some useful partial results: a[j,j] + a[j,h] + a[j,k]
  \\ and similarly for b. Put them in the free slots Fm1, Fm2.
  for( h = j+1, n,
    Fm2[i][j,h] = Sp[i,j] + a[i,h];  Fm1[i][j,h] = Sm[i,j] + b[i,h]; Eadd += 2;)));

\\ Start evaluation.
forstep( k=n, 2, -1,   if (k>t,T1 = a[k,k]<<2; T2 = b[k,k]<<2; Eshifts += 2;);
  forstep( h = k-1, 1, -1,
    T3 = Sp[h,k] + a[h,k]; T4 = Sm[h,k] + b[h,k]; Eadd += 2;
    Fp[1][n,n] = Sp[h,k] + a[k,k]; S2[h,k] = Fp[1][n,n] - a[h,k]<<1; \\ Some sides in 1
    Fp[n][n,n] = Sm[h,k] + b[k,k]; Sp[h,k] = Fp[n][n,n] - b[h,k]<<1; \\ and in -1
    Eadd += 4; Eadd2 += 2;

    forstep( j = h-1, 1, -1,    \\ Faces in (+1)
      Fm2[j][h,k] += Fp[1][n,n] ; Fm1[j][h,k] += Fp[n][n,n]; Eadd += 2;
      forstep( i = j-1, 1, -1,  \\ Pyramids.
        Sm[h,k]     = Fm2[j][h,k] + Fm2[i][j,k] + a[i,h];
        Fp[j][h,k]  = Fm1[j][h,k] + Fm1[i][j,k] + b[i,h];
        T[i,j][h,k] = Sm[h,k]*Fp[j][h,k]; Eadd += 4; );

      Fp[j][h,k] = Fm2[j][h,k]*Fm1[j][h,k];

    \\ Evaluate faces with -1 in two different positions.
      Sm[h,k] = S2[h,k] + Sp[j,h] - a[j,k];  Fm1[j][h,k] = Sp[h,k]+Sm[j,h]-b[j,k];
      Fm2[j][h,k] = Sm[h,k]*Fm1[j][h,k];
      if ( j > 1,
        Sm[h,k] = S2[h,k] + Sp[j,k] - a[j,h]; S2[1,2] = Sp[h,k] + Sm[j,k] - b[j,h];
        Fm1[j][h,k] = Sm[h,k]*S2[1,2];
      , \\ j == 1
        Sm[h,k] = S2[h,k]*Sp[h,k];          \\ Sides in -1.
        S2[h,k] += Sp[1,k] - a[1,h];          Sp[h,k] += Sm[1,k] - b[1,h];
        Fm1[1][h,k] = Sp[h,k]*S2[h,k];); Eadd += 8;
    );
    if ( h == 1, Sm[1,k] = S2[1,k]*Sp[1,k] );
    Sp[h,k] = Fp[1][n,n]*Fp[n][n,n];        \\ Sides in 1.
    if (k>t,
      Fp[1][n,n] = T1 + T3; Fp[n][n,n] = T2 + T4; Eadd += 2;,
      Fp[1][n,n] = T3 + a[k,k]<<2; Fp[n][n,n] = T4 + b[k,k]<<2; Eadd2 += 2; Eadd += 2;
    );
    S2[h,k] = Fp[1][n,n]*Fp[n][n,n];        \\ Sides in 2.
));
for( i=1, n, Fp[i][n,n] = a[i,i]*b[i,i] );  \\ Finally, the vertices.

\\\\\\\\\\\\ Interpolation \\\\\\\\\\\\\
\\ 1. Work on pyramids. (remove "12")
for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
  T[i,j][h,k] -= Fp[i][j,h]; add++; ))));

\\ 2. Decoupling on faces.
for( i=1,n-2, for( j = i+1, n-1, for( h = j+1, n,
  Fm1[i][j,h] = (Fp[i][j,h] - Fm1[i][j,h])>>1; add++; shifts++;
  Fm2[i][j,h] = (Fp[i][j,h] - Fm2[i][j,h])>>1; add++; shifts++;)));

\\ 3. Remove complete sides.
for( i=1,n-2, for( j = i+1, n-1, for( h = j+1,n, Fp[i][j,h] -= Sp[i,j]; add++;)));
```

```
\\ 4. Work on pyramids. (remove "7")
for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
  T[i,j][h,k] -= Fp[i][j,k]; add++; ))));

\\ 5. Work on sides...
for( j=2,n, if(j>t, shifts++; T0=Fp[j][n,n]<<1); for(i = 1, j-1,
  S2[i,j] = (S2[i,j] - Sm[i,j])/3;  Sm[i,j] = (Sp[i,j] - Sm[i,j])>>1;
  add++; div++; add++; shifts++;
  Sp[i,j] -= Fp[i][n,n];
  S2[i,j] = (S2[i,j] - Sp[i,j])>>1 - if(j>t,T0, add2++; Fp[j][n,n]<<1);
  add += 3; shifts++; ));

\\ 6. Inner points of faces.
for( i=1,n-2, for( j=i+1, n-1, for( h=j+1, n,
  Fm1[i][j,h] -= Sm[i,j]; Fm2[i][j,h] -= Sm[i,h];
  Fp[i][j,h] -= Sp[i,h]; add += 3;)));

\\ 7. Work on sides (obtain "3")
for( i=1,n, for(j = i+1,n, Sp[i,j] -= Fp[j][n,n]; add++; ));

\\ 8. Work on pyramids.
for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
 T[i,j][h,k] -= Fp[i][h,k]; add++; ))));

\\ 9. Work on faces (obtain "0")
for( i=2,n-2, for( j = i+1, n-1, for( h = j+1, n, Fp[i][j,h] -= Sp[j,h]; add++; )));

\\ 10. End working on pyramids.
for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
 T[i,j][h,k] -= Fp[j][h,k]; add++; ))));

\\ 11. Continue working on sides.
for( i=1,n-1, for( j = i+1, n,  Sp[i,j] -= Sm[i,j]; add++; ));

\\ 12. Continue working on faces.
for( j = 2, n-1, for( h = j+1, n, Fp[1][j,h] -= Sp[j,h]; add++; ));

\\ 13. End faces.                    \\ Faces without pi
for( i=2,n-2, for( j = i+1, n-1, for( h = j+1, n,
 Fm1[i][j,h] -= Sm[j,h]; Fm2[i][j,h] -= Sm[j,h];
 Fp [i][j,h] -= Fm1[i][j,h];
 Fm2[i][j,h] -= Fp [i][j,h];
 Fm1[i][j,h] -= Fm2[i][j,h]; add += 5; )));

for( j = 2, n-1, for( h = j+1 , n,   \\ "Spurious faces" with pi
 Fp [1][j,h] -= Fm1[1][j,h];
 Fm2[1][j,h] -= Fp [1][j,h];
 Fm1[1][j,h] -= Fm2[1][j,h];
 Fm2[1][j,h] -= Sm[j,h]; add += 4; ));

\\ 14. End sides.
for( i=1,n-1, for( j = i+1, n, Sm[i,j] -= S2[i,j]; add++;));

print("--------------- Final situation ------------------------------------");
\\ \\ Visualise sides
```

```
\\ for( i=1,n-1, for( j = i+1, n,
\\  print("Side(+)[", i, ",", j, "] = ",Sp[i,j], "\tSide(-)[", i, ",", j, "] = ",Sm[i,j],
\\  "      Side(2)[", i, ",", j, "] = ",S2[i,j])));
\\ \\ Visualise faces
\\ for( i=1,n-2, for( j = i+1, n, for( h = j+1, n,
\\     print("F+ [", i, ",", j, ",", h, "] = ",Fp[i][j,h]); )));
\\
\\ for( i=1,n-2, for( j = i+1, n, for( h = j+1, n,
\\     print("F-1 [", i, ",", j, ",", h, "] = ",Fm1[i][j,h],
\\           "\t\tF-2 [", i, ",", j, ",", h, "] = ",Fm2[i][j,h]); )));
\\ \\ Visualise tetrahedra
\\ for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
\\  print("T[",i,",",j,",",h,",",k,"] = ", T[i,j][h,k])))));

print("Interpolation:\n add    = ", binomial(N,2)*8+binomial(N,3)*11+binomial(N,4)*4
                              + binomial(N-1,3),
      ",\n shifts = ", binomial(N,2)*2+binomial(N,3)*2+N-S,
      ",\n _1_2   = ", binomial(S,2),
      ",\n div    = ", binomial(N,2),
      ";\nEvaluation:\n add    = ",2*(5*binomial(N,2) +  \\ Ev.Sides
                                    6*binomial(N,3) +  \\ Ev.Faces
                                    2*binomial(N,4)),  \\ Ev.Pyramids
       ",\n shifts = ", 2*(N-S),
       ",\n _1_2   = ",2*(binomial(N,2)+binomial(S,2))," with 1 <= S <= N.");
print("Expected: add = ", binomial(n,2)*8+binomial(n,3)*11+binomial(n,4)*4
                         + binomial(n-1,3),
      ", shifts = ", binomial(n,2)*2+binomial(n,3)*2+n-t,
      ", _1_2 = "  , binomial(t,2),
      ", div = "   , binomial(n,2),
      "; Eval: add = ",2*(5*binomial(n,2) +  \\ Ev.Sides
                          6*binomial(n,3) +  \\ Ev.Faces
                          2*binomial(n,4)),  \\ Ev.Pyramids
       ", shifts = ", 2*(n-t), ", _1_2 = ",2*(binomial(n,2)+binomial(t,2)));
print("Obtained: add = ", add, ", shifts = ",shifts,", _1_2 = ",add2,", div = ",
       div,"; Eval: add = ",Eadd,", shifts = ",Eshifts,", _1_2 = ",Eadd2);

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
print("Check ",binomial(n,2), " sides"); str1 = "";str2 = "";str3 = "";
for( i=1,n-1, for( j = i+1, n,
    str1 = Strexpand(str1,
            if(Sp[i,j]==a[i,i]*b[j,j] + a[i,j]*b[i,j] + a[j,j]*b[i,i]," 1"," 0"));
    str2 = Strexpand(str2, if(Sm[i,j]==a[i,i]*b[i,j] + a[i,j]*b[i,i]," 1"," 0"));
    str3 = Strexpand(str3, if(S2[i,j]==a[j,j]*b[i,j] + a[i,j]*b[j,j]," 1"," 0")) ));

print(str1); print(str2); print(str3);
print("Check ",binomial(n,3), " faces"); str1 = "";str2 = "";str3 = "";

for( i=1,n-2, for( j = i+1, n, for( h = j+1, n,
  str1 = Strexpand(str1, if(Fp[i][j,h] == a[j,j]*b[i,h] + a[i,j]*b[j,h]
                                       + a[j,h]*b[i,j] + a[i,h]*b[j,j]," 1"," 0"));
  str2 = Strexpand(str2, if(Fm1[i][j,h]== a[h,h]*b[i,j] + a[i,h]*b[j,h]
                                       + a[j,h]*b[i,h] + a[i,j]*b[h,h]," 1"," 0"));
  str3 = Strexpand(str3, if(Fm2[i][j,h]== a[i,i]*b[j,h] + a[i,j]*b[i,h]
                                       + a[i,h]*b[i,j] + a[j,h]*b[i,i]," 1"," 0"));)));
print(str1); print(str2); print(str3);
print("Check ",binomial(n,4), " pyramids"); str1 = "";
```

```
 for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
  str1 = Strexpand(str1, if(T[i,j][h,k] == a[i,j]*b[h,k] + a[i,h]*b[j,k] + a[i,k]*b[j,h]+
                                          a[j,h]*b[i,k] + a[j,k]*b[i,h] + a[h,k]*b[i,j],
       " 1"," 0")) ))));
 print(str1);}
```

```
\\ (C) 2011 Marco Bodrato and Alberto Zanoni
\\ This code is released under GPL 3.0 licence.

multivariateToom3_bis(n=5, t=0) =
{
 local(a,b, Sp,Sm,S2, Fp,Fm1,Fm2, T, i,j,h,k, T1=0, T2=0, T3=0, T4=0, T5=0, T6=0,
       Eadd=0, Eshifts=0, Eadd2=0, Eadd3=0, add = 0, shifts = 0, div = 0, add2 = 0);

 if(t<1|t>n,t=n);
 \\\\\\\\\\\\ Evaluation \\\\\\\\\\\\\
 a  = matrix(n, n, i,j, if ( i < j, eval(Str("a", i,j)),eval(Str("a", i))));
 b  = matrix(n, n, i,j, if ( i < j, eval(Str("b", i,j)),eval(Str("b", i))));

 Sp = matrix(n,n);   Fp  = vector(n, i, matrix(n,n));
 Sm = matrix(n,n);   Fm1 = vector(n, i, matrix(n,n));
 S2 = matrix(n,n);   Fm2 = vector(n, i, matrix(n,n));   T = matrix(n,n,i,j,matrix(n,n));

 \\ First evaluate partial sides...
 for( i=1, n-1, for( j = i+1, n, Eadd += 2;
  Sp[i,j] = a[i,j] + a[j,j];     Sm[i,j] = b[i,j] + b[j,j]; ));

 \\ ...then some useful partial results: a[h,h] + a[i,h] + a[j,h], and similarly for b.
 for( i=1, n-2, for( j = i+1, n-2, for( h = j+2, n, Eadd += 2;
   T[i,j][h-1,h] = Sp[i,h] + a[j,h];   Fm1[i][j,h] = Sm[i,h] + b[j,h]; )));

 \\ Start the true evaluation.
 for( i=1, n-1,
  \\ Prepare temporaries for all sides starting from vertex i.
  for( j = i+1, n-1,    Eadd += 4; Eadd2 += 2;
   Fp[i][j,n]  = Sp[i,j] + a[i,i]; S2[i,j]    = Fp[i][j,n]  - a[i,j]<<1; \\ Sides in  1
   Fm2[i][j,n] = Sm[i,j] + b[i,i]; Fp[j][n,n] = Fm2[i][j,n] - b[i,j]<<1; \\ Sides in -1
  );
  Eadd += 4; Eadd2 += 2;\\ j == n
  T1 = Sp[i,n] + a[i,i];  S2[i,n]    = T1 - a[i,n]<<1; \\ Sides in  1
  T2 = Sm[i,n] + b[i,i];  Fp[n][n,n] = T2 - b[i,n]<<1; \\ Sides in -1
  \\ End of temporaries preparation.

  for( j = i+1, n,        Eadd += 2; Eadd3 += 2;
   T3 = S2[i,j] + 3*Sp[i,j]; T4 = Fp[j][n,n] + 3*Sm[i,j]; \\ (1 2 4) for a and b
   for(h = j+1, n, \\ Faces
    if (h == j+1, T5 = if(j<n,Fp[i][j,n],T1)  + Sp[i,h] + a[j,h];
                  T6 = if(j<n,Fm2[i][j,n],T2) + Sm[i,h] + b[j,h]; Eadd += 4;
                ,
                  Fp[1][n,n] = T5 + T[i,j][h-1,h] + a[h-1,h];
                  S2[n-1,n]  = T6 + Fm1[i][j,h] + b[h-1,h];
                  T5 = if(j<n,Fp[i][j,n],T1)  + T[i,j][h-1,h];
                  T[i,j][h-1,h] = Fp[1][n,n]*S2[n-1,n];
```

```
                T6 = if(j<n,Fm2[i][j,n],T2) + Fm1[i][j,h];      Eadd += 6;
   );
   for( k = h+2, n,     \\ Pyramids.
     Fm2[i][j,h] = T5 + T[i,j][k-1,k] + a[h,k]; Fp[i][j,h] = T6 + Fm1[i][j,k] + b[h,k];
     T[i,j][h,k] = Fm2[i][j,h]*Fp[i][j,h]; Eadd += 4; );

   \\ Evaluate faces.
   Fp[1][n,n] = Sp[i,j] + S2[i,h] - a[j,h];
   if ( h == n, Sp[i,j] = if(j<n,Fp[i][j,n]*Fm2[i][j,n],T1*T2)); \\ Sides in 1.

   Fp[i][j,h]  = T5*T6;                              \\ Faces with all 1
   Fm1[i][j,h] = Sm[i,j] + Fp[h][n,n] - b[j,h]; Eadd += 4;
   Fm2[i][j,h] = Fp[1][n,n]*Fm1[i][j,h];             \\ Faces with 1 on (i,j)
   Fp[1][n,n]  = S2[i,j]    + Sp[i,h] - a[j,h];
   S2[n-1,n]   = Fp[j][n,n] + Sm[i,h] - b[j,h]; Eadd += 4;
   Fm1[i][j,h] = Fp[1][n,n]*S2[n-1,n];               \\ Faces with 1 on (i,h)
  );
 if (j == n, Sp[i,n] = T1*T2;);            \\ Some sides in 1.
 Sm[i,j] = S2[i,j]*Fp[j][n,n];             \\ Sides in -1.
 S2[i,j] = T3*T4;                          \\ Sides in  2.
));
for( i=1, n, Fp[i][n,n] = a[i,i]*b[i,i] );  \\ Finally, the vertices.

\\\\\\\\\\\\ Interpolation \\\\\\\\\\\\\

\\ ***** As in the multivariateToom3 function *****

print("--------------- Final situation ------------------------------------");
print("Interpolation:\n add    = ", binomial(N,2)*8+binomial(N,3)*11
                                  + binomial(N,4)*4+binomial(N-1,3),
     ",\n shifts = ", binomial(N,2)*2+binomial(N,3)*2+N-S,
     ",\n _1_2   = ", binomial(S,2),
     ",\n div    = ", binomial(N,2),
     ";\nEvaluation:\n add    = ",2*(4*binomial(N,2) + \\ Ev. sides
                                    6*binomial(N,3) + \\ Ev. faces
                                    2*binomial(N,4)), \\ Ev. pyramids
      ",\n shifts = ", 0, ",\n _1_2   = ",2*(binomial(N,2)),
      ",\n _1_3   = ",2*(binomial(N,2)));
print("Expected: add = ", binomial(n,2)*8+binomial(n,3)*11+binomial(n,4)*4
                        + binomial(n-1,3),
     ", shifts = ", binomial(n,2)*2+binomial(n,3)*2+n-t,
     ", _1_2 = "  , binomial(t,2),
     ", div = "   , binomial(n,2),
     "; Eval: add = ",2*(4*binomial(n,2) +  \\ Ev. sides
                         6*binomial(n,3) +  \\ Ev. faces
                         2*binomial(n,4)),  \\ Ev. pyramids
      ", shifts = ", 0, ", _1_2 = ",2*binomial(n,2),", _1_3 = ",2*binomial(n,2));
print("Obtained: add = ", add, ", shifts = ",shifts,", _1_2 = ",add2,", div = ",div,
      "; Eval: add = ",Eadd,", shifts = ",Eshifts,", _1_2 = ",Eadd2, ", _1_3 = ",Eadd3);
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

print("Check ",binomial(n,2), " sides"); str1 = "";str2 = "";str3 = "";
for( i=1,n-1, for( j = i+1, n,
  str1 = Strexpand(str1,
           if(Sp[i,j] == a[i,i]*b[j,j] + a[i,j]*b[i,j] + a[j,j]*b[i,i]," 1"," 0"));
  str2 = Strexpand(str2, if(Sm[i,j] == a[i,i]*b[i,j] + a[i,j]*b[i,i]," 1"," 0"));
```

```
  str3 = Strexpand(str3, if(S2[i,j] == a[j,j]*b[i,j] + a[i,j]*b[j,j]," 1"," 0"));
 ));
 print(str1); print(str2); print(str3);
 print("Check ",binomial(n,3), " faces"); str1 = "";str2 = "";str3 = "";

 for( i=1,n-2, for( j = i+1, n, for( h = j+1, n,
   str1 = Strexpand(str1, if(Fp[i][j,h] == a[j,j]*b[i,h] + a[i,j]*b[j,h]
                                        + a[j,h]*b[i,j] + a[i,h]*b[j,j]," 1"," 0"));
   str2 = Strexpand(str2, if(Fm1[i][j,h]== a[h,h]*b[i,j] + a[i,h]*b[j,h]
                                        + a[j,h]*b[i,h] + a[i,j]*b[h,h]," 1"," 0"));
   str3 = Strexpand(str3, if(Fm2[i][j,h]== a[i,i]*b[j,h] + a[i,j]*b[i,h]
                                        + a[i,h]*b[i,j] + a[j,h]*b[i,i]," 1"," 0"));
 )));
 print(str1); print(str2); print(str3);
 print("Check ",binomial(n,4), " pyramids"); str1 = "";
 for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
  str1 = Strexpand(str1, if(T[i,j][h,k] == a[i,j]*b[h,k] + a[i,h]*b[j,k] + a[i,k]*b[j,h]+
                                           a[j,h]*b[i,k] + a[j,k]*b[i,h] + a[h,k]*b[i,j],
       " 1"," 0")) ))));
 print(str1);
}
```

Appendix E. Unbalanced multivariate Toom-3 in characteristic different from 2 and 3

```
\\ (C) 2011 Marco Bodrato and Alberto Zanoni
\\ This code is released under GPL 3.0 licence.

multivariateToom3_unbalanced(n=5, t=0) =
{
 local(a,b, Sp,Sm,S2, Fp,Fm1,Fm2, T, i,j,h,k, Eadd=0, Eshifts=0,
       T1=0,T2=0,Eadd2=0, Eadd3=0, add=0, shifts=0, div=0, add2=0);

 if(t<1|t>n,t=n);
 \\\\\\\\\\\\ Evaluation \\\\\\\\\\\\\\
 a  = matrix(n,n, i,j, if( i == j,eval(Str("a", i)), eval(Str("a", i,j))));
 aa = vector(n, i, matrix(n,n,j,h,if(i<j && j<h, eval(Str("a", i,j,h) ))));
 b  = vector(n, i, eval(Str("b", i)));

 Sp = matrix(n, n);   Fp  = vector(n, i, matrix(n,n));
 Sm = matrix(n, n);   Fm1 = vector(n, i, matrix(n,n));
 S2 = matrix(n, n);   Fm2 = vector(n, i, matrix(n,n));  T = matrix(n,n,i,j,matrix(n,n));

 \\ Evaluate first partial sides.
 for( i=1, n-1, for( j = i+1, n,  Sp[i,j] = a[i,j] + a[j,i]; Sm[i,j] = Sp[i,j] + a[i,i];
                                  S2[i,j] = Sm[i,j] - a[i,j]<<1; Eadd += 3; Eadd2 += 1));

 \\ Evaluate some useful partial results for faces and pyramids
 \\ in the free slots Fm1, Fm2.
  for( i=1, n-1, for( j = i+1, n, for( h = j+1, n,
    Fm1[i][j,h] = Sp[i,j] + aa[i][j,h];
    Fm2[i][j,h] = Fm1[i][j,h] + Sm[i,h];  Eadd += 2;)));

 \\ Start evaluation.
 forstep( k=n, 2, -1,  if (k>t,T1 = a[k,k]<<3; Eshifts += 1;);
```

```
  forstep( h = k-1, 1, -1,
    Fp[1][n,n] = Sm[h,k] + a[k,k]; T2 = S2[h,k] - a[k,k]; Eadd += 2;
    S2[h,k]   += 3*Sp[h,k];                               Eadd += 1; Eadd3 += 1;
    Fp[n][n,n] = b[h] + b[k];      Sp[h,k] = b[h] - b[k]; Eadd += 2;

    forstep( j = h-1, 1, -1,    \\ Faces in (+1)
      Fm1[j][h,k] = Fm2[j][h,k] + Fp[1][n,n]; Fp[2][n,n] = Fp[n][n,n] + b[j];
      Eadd += 2;
      forstep( i = j-1, 1, -1,  \\ Pyramids.
        Sm[h,k]     = Fm1[j][h,k] + Fm1[i][j,h] + Fm2[i][h,k] + aa[i][j,k];
        Fp[j][h,k]  = Fp[2][n,n] + b[i]; Eadd += 4;
        T[i,j][h,k] = Sm[h,k]*Fp[j][h,k];
      );
      Fp[j][h,k] = Fm1[j][h,k]*Fp[2][n,n];  \\ Faces in 1.

    \\ Evaluate faces with -1 in two different positions.
      Fp[2][n,n]  = T2+S2[j,k]+Sp[j,h]-aa[j][h,k]; Sm[h,k] = b[j] + Sp[h,k];
      Fm2[j][h,k] = Fp[2][n,n]*Sm[h,k];
      Fp[2][n,n]  = Sp[j,k]-T2+S2[j,h]-aa[j][h,k]; Sm[h,k] = b[j] - Sp[h,k];
      Fm1[j][h,k] = Fp[2][n,n]*Sm[h,k];            Eadd += 8;
    );
    Sm[h,k] = T2*Sp[h,k];                   \\ Sides in -1.
    Sp[h,k] = Fp[1][n,n]*Fp[n][n,n];        \\ Sides in 1.

    Fp[n][n,n] += b[k]; Eadd += 1;
    if (k>t, Fp[1][n,n] = S2[h,k] + T1;
           , Fp[1][n,n] = S2[h,k] + a[k,k]<<3; Eadd2 += 1; ); Eadd += 1;
    S2[h,k] = Fp[1][n,n]*Fp[n][n,n];        \\ Sides in 2.
));
for( i=1, n, Fp[i][n,n] = a[i,i]*b[i] );    \\ Finally, the vertices.

\\\\\\\\\\\\ Interpolation \\\\\\\\\\\\\
\\ ***** As in the multivariateToom3 function *****

print("--------------- Final situation ------------------------------------");
print("Interpolation:\n add    = ", binomial(N,2)*8+binomial(N,3)*11+binomial(N,4)*4
                            + binomial(N-1,3),
      ",\n shifts = ", binomial(N,2)*2+binomial(N,3)*2+N-S,
      ",\n _1_2   = ", binomial(S,2),
      ",\n div    = ", binomial(N,2),
      ";\nEvaluation:\n add    = ",2*(5*binomial(N,2) +  \\ Ev.Sides
                                    6*binomial(N,3) +  \\ Ev.Faces
                                    2*binomial(N,4)),  \\ Ev.Pyramids
       ",\n shifts = ", (N-S),
       ",\n _1_2   = ",binomial(N,2)+binomial(S,2)," with 1 <= S <= N.",
       ",\n _1_3   = ",binomial(N,2));
print("Expected: add = ", binomial(n,2)*8+binomial(n,3)*11+binomial(n,4)*4
                        + binomial(n-1,3),
      ", shifts = ", binomial(n,2)*2+binomial(n,3)*2+n-t,
      ", _1_2 = "  , binomial(t,2),
      ", div = "   , binomial(n,2),
      "; Eval: add = ", 2*(5*binomial(n,2) +  \\ Ev.Sides
                           6*binomial(n,3) +  \\ Ev.Faces
                           2*binomial(n,4)),  \\ Ev.Pyramids
       ", shifts = ", (n-t), ", _1_2 = ",binomial(n,2) + binomial(t,2)
       ", _1_3 = ",binomial(n,2));
```

```
print("Obtained: add = ", add, ", shifts = ",shifts,", _1_2 = ",add2,", div = ",div,
       "; Eval: add = ",Eadd,", shifts = ",Eshifts,", _1_2 = ",Eadd2,", _1_3 = ",Eadd3);

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
print("Check ",binomial(n,2), " sides"); str1 = "";str2 = "";str3 = "";
for( i=1,n-1, for( j = i+1, n,
    str1 = Strexpand(str1,
            if(Sp[i,j]==a[i,j]*b[j] + a[j,i]*b[i]," 1",Str(" ",Sp[i,j])));
    str2 = Strexpand(str2, if(Sm[i,j]==a[i,i]*b[j] + a[i,j]*b[i]," 1"," 0"));
    str3 = Strexpand(str3, if(S2[i,j]==a[j,j]*b[i] + a[j,i]*b[j]," 1"," 0")) ));
print(str1); print(str2); print(str3);
print("Check ",binomial(n,3), " faces"); str1 = "";str2 = "";str3 = "";

for( i=1,n-2, for( j = i+1, n, for( h = j+1, n,
  str1 = Strexpand(str1, if(Fp[i][j,h] == a[j,i]*b[h] + a[j,h]*b[i]
                                       + aa[i][j,h]*b[j]," 1"," 0"));
  str2 = Strexpand(str2, if(Fm1[i][j,h]== a[h,i]*b[j] + a[h,j]*b[i]
                                       + aa[i][j,h]*b[h]," 1"," 0"));
  str3 = Strexpand(str3, if(Fm2[i][j,h]== a[i,j]*b[h] + a[i,h]*b[j]
                                       + aa[i][j,h]*b[i]," 1"," 0"));
)));
print(str1); print(str2); print(str3);
print("Check ",binomial(n,4), " pyramids"); str1 = "";
for( i=1,n-3, for( j = i+1, n-2, for( h = j+1,n-1, for( k = h+1, n,
 str1 = Strexpand(str1, if(T[i,j][h,k] == aa[i][j,h]*b[k] + aa[i][j,k]*b[h]
                                       + aa[i][h,k]*b[j] + aa[j][h,k]*b[i],
      " 1"," 0")) ))));
print(str1);
```

Acknowledgement: The second author is partially funded by research project "Robustezza e tolleranza ai guasti in reti e grafi", Sapienza University, Rome.

References

[1] Marco Bodrato, *Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0.* In Claude Carlet and Berk Sunar, editors, *WAIFI'07 proceedings*, volume 4547 of *LNCS*, pages 116–133. Springer, June 2007, URL: http://bodrato.it/papers/#WAIFI2007.

[2] Marco Bodrato and Alberto Zanoni, *Integer and polynomial multiplication: Towards optimal Toom-Cook matrices*, In Christopher W. Brown, editor, *Proceedings of the ISSAC 2007 Conference*, pages 17–24. ACM press, July 2007, URL: http://bodrato.it/papers/#ISSAC2007.

[3] John F. Canny, Erich Kaltofen, and Lakshman Yagati, *Solving systems of nonlinear polynomial equations faster*, In *Proceedings of the ACM-*

SIGSAM 1989 international symposium on Symbolic and algebraic computation, pages 121–128, New York, NY, USA, 1989. ACM Press.

[4] David G. Cantor and Erich Kaltofen, *On fast multiplication of polynomials over arbitrary algebras*, *Acta Informatica*, 28(7):693–701, 1991.

[5] Stephen A. Cook, *On the minimum computation time of functions*, PhD thesis, Harvard University, Cambridge, MA, USA 1966, URL: `http://cr.yp.to/bib/entries.html#1966/cook`.

[6] Richard Fateman, *Comparing the speed of programs for sparse polynomial multiplication*, *SIGSAM Bulletin*, 37(1):4–15, 2003.

[7] Richard Fateman, *Can you save time in multiplying polynomials by encoding them as integers?*, 2005. Draft.

[8] Anatolii Alexeevich Karatsuba and Yuri Ofman, *Multiplication of multidigit numbers on automata*, *Soviet Physics Doklady*, 7(7):595–596, 1963.

[9] Robert T. Moenck, *Practical fast polynomial multiplication*, In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 136–148, New York, NY, USA, 1976. ACM Press.

[10] Victor Y. Pan. Simple multivariate polynomial multiplication. *Journal of Symbolic Computation*, 18(3):183–186, 1994.

[11] Andrei L. Toom, *The complexity of a scheme of functional elements realizing the multiplication of integers*, Soviet Mathematics Doklady, 3:714–716, 1963. URL : `http://www.de.ufpe.br/~toom/articles/engmat/MULT-E.PDF`.

[12] André Weimerskirch and Christof Paar, *Generalizations of the karatsuba algorithm for polynomial multiplication*, Technical report, Ruhr-Universität-Bochum, 2003.

Marco Bodrato
mambaSoft, Via S. Marino 118 – 10137 Torino, Italy
email: *bodrato@mail.dm.unipi.it*
Alberto Zanoni
Dipartimento di Scienze Statistiche - Università “La Sapienza”
P.le Aldo Moro 5 – 00185 Roma, Italy
email: *zanoni@volterra.uniroma2.it*