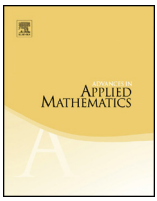
Advances in Applied Mathematics 170 (2025) 102931

Contents lists available at ScienceDirect

Advances in Applied Mathematics

journal homepage: www.elsevier.com/locate/yaama



A polynomial time algorithm for Sylvester waves when entries are bounded

Guoce Xin [a], Chen Zhang [b,*]

[a] *School of Mathematical Sciences, Capital Normal University, Beijing 100048, PR China*
[b] *Center for Combinatorics, LPMC, Nankai University, Tianjin 300071, PR China*

ARTICLE INFO

Article history:
Received 27 June 2024
Received in revised form 18 June 2025
Accepted 23 June 2025
Available online 3 July 2025

MSC:
primary 05–08
secondary 05–04, 05A17

Keywords:
Sylvester's denumerant
Sylvester wave
Cyclotomic polynomial

ABSTRACT

Sylvester's denumerant $d(t;\boldsymbol{a})$ is a quantity that counts the number of nonnegative integer solutions to the equation $\sum_{i=1}^{N} a_i x_i = t$, where $\boldsymbol{a} = (a_1, a_2, \ldots, a_N)$ is a sequence of positive integers with $\gcd(\boldsymbol{a}) = 1$. We present a polynomial time algorithm in N for computing $d(t;\boldsymbol{a})$ when $\boldsymbol{a}$ is bounded and t is a parameter. The proposed algorithm is rooted in the use of cyclotomic polynomials and builds upon recent results by Xin-Zhang-Zhang on the efficient computation of generalized Todd polynomials. The algorithm has been implemented in `Maple` under the name `Cyc-Denum` and demonstrates superior performance when $a_i \leq 500$ compared to Sills-Zeilberger's `Maple` package `PARTITIONS`.

1. Introduction

For a positive integer sequence $\boldsymbol{a} = (a_1, a_2, \ldots, a_N)$ with $\gcd(\boldsymbol{a}) = 1$ and a nonnegative integer t, Sylvester's *denumerant* [12], denoted by $d(t;\boldsymbol{a})$, is a function that enumerates the number of nonnegative integer solutions to $\sum_{i=1}^{N} a_i x_i = t$. This concept has been the

* Corresponding author.
E-mail addresses: guoce_xin@163.com (G. Xin), ch_enz@163.com (C. Zhang).

https://doi.org/10.1016/j.aam.2025.102931

subject of extensive research, and a notable result states that $d(t;\boldsymbol{a})$ is a quasi-polynomial in t of degree $N-1$.

In 2015, Baldoni et al. [4] gave a polynomial time algorithm for computing the top k coefficients of $d(t;\boldsymbol{a})$ when k is fixed. Meanwhile, they developed a `Maple` package `M-Knapsack`, as well as a `C++` package `LattE knapsack`. In [14], we introduced our own `Maple` package `CT-Knapsack`, which has a significant speed advantage over `M-Knapsack`. The experimental results can be found in [14, Section 5]. However, both Baldoni et al.'s algorithm and our own rely on Barvinok's unimodular cone decomposition. The computations of the bottom coefficients of $d(t;\boldsymbol{a})$ become challenging when N is large, as it necessitates the decomposition of high dimensional cones. This issue was addressed in our previous work [14]. Then we take note of Sills and Zeilberger's work in 2012 [11]. They provided a `Maple` package `PARTITIONS`, which can compute $p_k(t)$ for k up to 70, where $p_k(t) = d(t;1,2,\ldots,k)$. But it already takes 83.4 seconds for `CT-Knapsack` to compute $p_{12}(t)$. The computation of $p_{70}(t)$ is out of reach even for `LattE Knapsack`.

The main idea of Sills and Zeilberger's `PARTITIONS` is deriving the formula for $d(t;\boldsymbol{a})$ with the aid of the partial fraction decomposition of the generating function. This idea dates back at least to Cayley [5]. They ask `Maple` to convert the generating function

$$\sum_{t\geq 0} d(t;\boldsymbol{a})q^t = \frac{1}{\prod_{i=1}^{N}(1-q^{a_i})} \tag{1.1}$$

into partial fractions. Then for each piece, `Maple` finds the first few terms of the Maclaurin expansion and then fits the data with an appropriate quasi-polynomial using *undetermined coefficients*. The output is the list of these quasi-polynomials whose sum is the desired expression for $d(t;\boldsymbol{a})$. See [11, Section 3] for details.

Our main contribution in this paper is the development of a polynomial time algorithm, `Cyc-Denum`, which efficiently computes Sylvester's denumerant $d(t;\boldsymbol{a})$ when the entries of $\boldsymbol{a}$ are bounded. Unlike previous methods that rely on unimodular cone decomposition or partial fraction decomposition, our approach utilizes cyclotomic polynomials and the log-exponential trick [17] for computing generalized Todd polynomials. This leads to significantly improved performance, allowing us to compute $p_k(t)$ for larger k in a shorter time compared to `PARTITIONS`. Additionally, we introduce a floating-point computation algorithm, `Float-Denum`, which provides approximate results with low memory requirements. The complexity analysis of these two algorithms are presented in Theorems 2.11 and 3.1, respectively. Both algorithms, `Cyc-Denum` and `Float-Denum`, are implemented and released as part of the `Maple` software package `Cyc-Denum` [15].

Algorithm `Cyc-Denum` (see Algorithm 1), which is capable of computing $d(t;1,2,\ldots,k)$ for any $1\leq k\leq 126$ in 1 minute. This algorithm is so named since we heavily use the *f-th cyclotomic polynomial*, which is denoted by

$$\Phi_f(x) := \prod_{\zeta\in\Theta_f}(x-\zeta),$$

where Θ_f denotes the set of all f-th primitive roots of unity, i.e.,

$$\Theta_f = \{\zeta : \zeta = \mathrm{e}^{\frac{2j\pi\sqrt{-1}}{f}} \text{ with } 1 \le j \le f \text{ and } \gcd(j,f) = 1\}.$$

Algorithm `Float-Denum` (see Algorithm 2) is based on floating-point computations, which can compute $d(t;1,2,\ldots,k)$ for any $1 \le k \le 151$ in 1 minute. In contrast, `PARTITIONS` is limited to computations for k up to 58 in 1 minute. This comparison underscores the enhanced efficiency and broader scope of our two algorithms in the context of the denumerant. The detailed experiments are reported in Subsection 2.4.

It is well known [12,13] that

$$d(t;\boldsymbol{a}) = \sum_f W_f(t;\boldsymbol{a}), \tag{1.2}$$

where f ranges over all divisors of $a_1, a_2, \ldots, a_N$, and $W_f(t;\boldsymbol{a})$ is called a *Sylvester wave* and described as a *quasi-polynomial* in t of period f for each f. A *quasi-polynomial of period* f is a function that can be written as $F(t) = [P_1(t), P_2(t), \ldots, P_f(t)]$, where $P_i(t)$'s are polynomials in t, $F(t) = P_f(t)$ if $t \equiv 0 \pmod f$ and $F(t) = P_i(t)$ if $t \equiv i \pmod f$ for $1 \le i \le f-1$. For more research on the denumerant and Sylvester wave, see e.g., [1–3,6,8–10].

Example 1.1. Let $\boldsymbol{a} = (1,3,6)$, then

$$d(t;\boldsymbol{a}) = W_1(t;\boldsymbol{a}) + W_2(t;\boldsymbol{a}) + W_3(t;\boldsymbol{a}) + W_6(t;\boldsymbol{a}),$$

where

$$W_1(t;\boldsymbol{a}) = \left[\frac{t^2}{36} + \frac{5\,t}{18} + \frac{127}{216}\right], \qquad W_2(t;\boldsymbol{a}) = \left[-\frac{1}{24}, \frac{1}{24}\right],$$
$$W_3(t;\boldsymbol{a}) = \left[-\frac{1}{54}, -\frac{t}{18} - \frac{29}{108}, \frac{t}{18} + \frac{31}{108}\right], \quad W_6(t;\boldsymbol{a}) = \left[\frac{1}{6}, \frac{1}{12}, -\frac{1}{12}, -\frac{1}{6}, -\frac{1}{12}, \frac{1}{12}\right].$$

Taking $t = 12$ gives

$$W_1(12;\boldsymbol{a}) = \frac{1711}{216}, \quad W_2(12;\boldsymbol{a}) = \frac{1}{24}, \quad W_3(12;\boldsymbol{a}) = \frac{103}{108}, \quad \text{and} \quad W_6(12;\boldsymbol{a}) = \frac{1}{12}.$$

Hence $d(12;\boldsymbol{a}) = 9$. It is easy to verify that there are exactly 9 nonnegative integer solutions to $x_1 + 3x_2 + 6x_3 = 12$.

Generally, we have [12]

$$W_f(t;\boldsymbol{a}) = -\operatorname*{res}_{s=0} \sum_{\zeta\in\Theta_f} \frac{\zeta^{-t}\mathrm{e}^{-ts}}{\prod_{i=1}^{N}(1-\zeta^{a_i}\mathrm{e}^{a_i s})}, \tag{1.3}$$

where $\operatorname*{res}_{s=s_0} F(s)$ denotes the residue of $F(s)$ when expanded as a Laurent series at $s = s_0$. More precisely, $\operatorname*{res}_{s=s_0} \sum_{i\geq i_0} c_i(s-s_0)^i = c_{-1}$.

The structure of this paper is as follows. In Section 2, we commence by introducing the log-exponential trick, a key technique for the computation of generalized Todd polynomials [17]. This trick serves as a fundamental tool in the computation of $W_f(t;\boldsymbol{a})$. Subsequently, we delineate the computation scheme for $W_f(t;\boldsymbol{a})$ when f is given, and we present Algorithm `Cyc-Denum` in detail. The complexity analysis of this scheme is provided in Theorem 2.11. In Section 3, we give another scheme with better performance for computing $W_f(t;\boldsymbol{a})$ by converting the f-th roots of unity ζ's into floating-point numbers. We also provide the corresponding algorithm for this scheme, along with its complexity analysis. Section 4 contains concluding remarks.

2. Computation of $W_f(t;\boldsymbol{a})$

In this section, we provide a succinct overview of the log-exponential trick, as introduced by Xin et al. [17], for the efficient computation of generalized Todd polynomials. We utilize this trick to compute $W_f(t;\boldsymbol{a})$ for a fixed f. Additionally, we present Algorithm `Cyc-Denum`, which has been implemented in `Maple`.

2.1. Two tools

The first tool we use is a straightforward lemma that serves as a fundamental result for our calculations.

Lemma 2.1. *Let f be a positive integer, and let $F(x)$ be a rational function in x. Suppose $F(\zeta)$ exists for all ζ satisfying $\zeta^f = 1$. Then*

$$\sum_{\zeta\in\Theta_f} F(\zeta) = \frac{1}{f}\sum_{\zeta:\zeta^f=1} \zeta F(\zeta)\widehat{\Phi}_f(\zeta)\Phi_f'(\zeta), \qquad \text{where } \widehat{\Phi}_f(x) = \frac{x^f-1}{\Phi_f(x)}.$$

Proof. Note that both $\widehat{\Phi}_f(x)$ and $\Phi_f'(x)$ are polynomials in x, and $\widehat{\Phi}_f(\zeta) = 0$ for any $\zeta \in \{\zeta : \zeta^f = 1\} \setminus \Theta_f$. A direct computation using L'Hôpital's rule yields

$$\begin{aligned}
\frac{1}{f}\sum_{\zeta:\zeta^f=1} \zeta F(\zeta)\widehat{\Phi}_f(\zeta)\Phi_f'(\zeta) &= \sum_{\zeta\in\Theta_f} \zeta F(\zeta)\widehat{\Phi}_f(\zeta)\Phi_f'(\zeta)\zeta^{f-1}\lim_{x=\zeta}\frac{x-\zeta}{x^f-1}\\
&= \sum_{\zeta\in\Theta_f} F(\zeta)\Phi_f'(\zeta)\lim_{x=\zeta}\widehat{\Phi}_f(x)\frac{x-\zeta}{x^f-1}\\
&= \sum_{\zeta\in\Theta_f} F(\zeta)\Phi_f'(\zeta)\lim_{x=\zeta}\frac{x-\zeta}{\Phi_f(x)}
\end{aligned}$$

$$= \sum_{\zeta \in \Theta_f} F(\zeta)\Phi_f'(\zeta)\frac{1}{\Phi_f'(\zeta)}$$
$$= \sum_{\zeta \in \Theta_f} F(\zeta). \quad \square$$

The second tool involves the computation of generalized Todd polynomials using the log-exponential trick, a concept introduced by Xin et al. in [17]. Subsequently, we will consistently use the notation $R = \mathbb{Q}(\zeta)$, where ζ is a primitive f-th root of unity, with R collapsing to $\mathbb{Q}$ when $\zeta = 1$. This is adequate for our objectives.

Consider $a \in \mathbb{Q}$ and let $B_0, \bar{B}_0, B_1, \bar{B}_1, \ldots, B_r, \bar{B}_r$ be finite multi-sets of nonzero integers. The generalized Todd polynomials, denoted as gtd_d, are defined by their generating function

$$F(s) = \sum_{d \geq 0} gtd_d s^d = \mathrm{e}^{as} \frac{\prod_{b \in B_0} g(bs)}{\prod_{b \in \bar{B}_0} g(bs)} \prod_{i=1}^{r} \frac{\prod_{b \in B_i} g(bs, y_i)}{\prod_{b \in \bar{B}_i} g(bs, y_i)},$$

where

$$g(s) = \frac{s}{\mathrm{e}^s - 1} = 1 + o(1), \qquad g(s, y) = \frac{1}{1 - y(\mathrm{e}^s - 1)} = 1 + o(1).$$

For our purposes, we focus on the scenario where $\bar{B}_i$ is empty and $y_i = \alpha_i \in R$ for each i. Thus, the generating function simplifies to

$$F(s) = \sum_{d \geq 0} gtd_d s^d = \mathrm{e}^{as} \prod_{b \in B_0} g(bs) \prod_{i=1}^{r} \prod_{b \in B_i} g(bs, \alpha_i). \tag{2.1}$$

The log-exponential trick involves first calculating $H(s) = \ln(F(s))$ using expansion formulas:

$$h(s) = \ln g(s) = -\sum_{k \geq 1} \frac{\mathcal{B}_k}{k \cdot k!} s^k, \qquad h(s, y) = \ln g(s, y) = \sum_{k \geq 1} C_k(y) s^k, \tag{2.2}$$

and subsequently computing $\mathrm{e}^{H(s)}$. Here, the $\mathcal{B}_k$'s are the Bernoulli numbers, and $C_k(y)$'s are polynomials in y. Xin et al. [17] have provided a thorough examination of the computations involving logarithms and exponentials, along with complexity analysis. Their analysis was conducted under the assumption that the y_i's are variables. We revisit this complexity analysis, focusing on the scenario where $y_i = \alpha_i \in R$ for all i.

The complexities are measured by the number of (arithmetic) operations in the ring R. Several established results from [17] are required in our work. These were originally obtained through computational algebra approaches, particularly Newton's iteration.

A commutative ring R is said to support fast Fourier transform (FFT) if it contains a primitive 2^k-th root of unity for any nonnegative integer k. We will consistently assume that FFT is feasible.

Lemma 2.2 *([17, Lemma 8]). Let $d = 2^\ell$ for a positive integer ℓ. Given $G(s) \pmod{\langle s^d \rangle} \in R[s]$ with $G(0) = 1$, then $\ln G(s) \pmod{\langle s^d \rangle}$ can be computed using $O(d \log d)$ operations in R.*

Lemma 2.3 *([17, Theorem 9]). Let $d = 2^\ell$ for a positive integer ℓ. Given $H(s) \pmod{\langle s^d \rangle} \in R[s]$ with $H(0) = 0$, $\mathrm{e}^{H(s)} \pmod{\langle s^d \rangle}$ can be computed in $O(d \log(d))$ operations in R.*

Lemma 2.4 *([17, Corollary 17]). Consider a multi-set B of k elements within the ring R as previously defined. If $H(s) \pmod{\langle s^d \rangle}$ is provided in $R[s]$ with $H(0) = 0$, then the computation of $\sum_{b \in B} H(bs)$ can be achieved using $O(k \log^2(d) + d \log(d))$ operations in R.*

Lemma 2.5. *Let $h(s)$ and $h(s, y)$ be defined as in (2.2). Then for an $\alpha \in R$, $h(s) \pmod{\langle s^d \rangle}$ and $h(s, \alpha) \pmod{\langle s^d \rangle}$ can be computed using $O(d \log(d))$ operations in R.*

Proof. The complexity of evaluating both $h(s) \pmod{\langle s^d \rangle}$ and $h(s, y) \pmod{\langle s^d \rangle}$, where y is a variable, is established in [17, Lemma 21]. Here we take $y = \alpha \in R$. The computation of $h(s, \alpha) \pmod{\langle s^d \rangle}$ can be derived from Lemma 2.2 and

$$h(s, \alpha) = \ln g(s, \alpha) = -\ln(1 - \sum_{i \geq 1} \frac{\alpha}{i!} s^i). \quad \square$$

Theorem 2.6. *Let d be a positive integer. For given $a, B_0, B_1, \ldots, B_r$, let $F(s)$ be as defined in (2.1), i.e.,*

$$F(s) = \sum_{d \geq 0} gtd_d s^d = \mathrm{e}^{as} \prod_{b \in B_0} g(bs) \prod_{i=1}^{r} \prod_{b \in B_i} g(bs, \alpha_i), \quad \text{where } \alpha_i \in R.$$

Then we can compute $F(s) \pmod{\langle s^d \rangle}$, or the sequence $(gtd_0, gtd_1, \ldots, gtd_{d-1})$ of generalized Todd polynomials equivalently, using $O\Big((r+1)d \log(d) + \log^2(d) \sum_{i=0}^{r} |B_i|\Big)$ operations in R.

Proof. Let

$$H(s) = \ln(F(s)) = as + \sum_{b \in B_0} h(bs) + \sum_{i=1}^{r} \sum_{b \in B_i} h(bs, \alpha_i).$$

We proceed by constructing $H(s) \pmod{\langle s^d \rangle}$ and subsequently computing $F(s) \equiv e^{H(s)} \pmod{\langle s^d \rangle}$, which results in the sequence $(gtd_0, gtd_1, \ldots, gtd_{d-1})$.

In Step 0, we utilize Lemma 2.5 to compute $h(s)$ using $O(d \log(d))$ operations in R, and compute $h(s, \alpha_i)$ for $1 \leq i \leq r$ using $O(rd \log(d))$ operations in R.

In Step 1, we apply Lemma 2.4 to compute $\sum_{b\in B_0} h(bs) \pmod{\langle s^d\rangle}$ using $O\Big(d\log(d)+|B_0|\log^2(d)\Big)$ operations in R and $\sum_{b\in B_0} h(bs,\alpha_i) \pmod{\langle s^d\rangle}$ for $1\le i\le r$ using $O\Big(rd\log(d)+\log^2(d)\sum_{i=1}^r |B_i|\Big)$ operations in R.

In Step 2, we invoke Lemma 2.3 to compute $\mathrm{e}^{H(s)} \pmod{\langle s^d\rangle}$. The operational cost for this step amounts to $O(d\log(d))$ operations in R.

The total complexity is the sum of the complexities of the individual steps, leading to the claimed bound. □

2.2. Computation of $W_f(t;\boldsymbol{a})$

Throughout this and the next subsections, we often operate within the field $R = \mathbb{Q}[x]/\langle\Phi_f(x)\rangle$. An element $P(x)+\langle\Phi_f(x)\rangle$ in R is represented by its *standard form* $\langle P(x)\rangle_R$, which is the unique polynomial representative of degree at most $\varphi(f)-1$, where $\varphi(f)=\#\{j : 1\le j\le f \text{ and } \gcd(j,f)=1\}$ is Euler's totient function. Specifically, $\langle P(x)\rangle_R$ is determined through two steps: i) Calculate the remainder Q of P when divided by x^f-1; ii) Calculate the remainder of Q when divided by $\Phi_f(x)$.

Multiplications in R are fast, with a complexity of $O(\beta\log\beta)$ operations in $\mathbb{Q}$, where $\beta=\varphi(f)$. This is supported by a well known result in computer algebra:

Proposition 2.7 *([7, Corollary 9.7]). Let D be a ring (commutative, with 1) and $m(x)$ a monic polynomial of degree n in $D[x]$. Then one multiplication in the residue class ring $D[x]/\langle m(x)\rangle$ can be done using $O(n\log n)$ operations in D.*

The following lemma asserts that $\frac{1}{1-\zeta}$ can be expressed as a polynomial in ζ for any f-th root of unity ζ $(\neq 1)$ (hence a polynomial of standard form over R if $\zeta\in\Theta_f$).

Lemma 2.8. *Suppose $\zeta\neq 1$ is a f-th root of unity, and $\theta_f(x)=\sum_{i=0}^{f-1} x^i$. Then we have*

$$\frac{1}{1-\zeta} = -\frac{1}{f}\zeta\theta_f'(\zeta).$$

Proof. Observe that $x^f-1=(x-1)\theta_f(x)$. Taking the derivative of both sides with respect to x yields $fx^{f-1}=\theta_f(x)+(x-1)\theta_f'(x)$. The lemma follows from $\theta_f(\zeta)=0$ for any $\zeta\neq 1$. □

For a fixed f and each $\zeta\in\Theta_f$, $\zeta^{a_i}=1$ if and only if $f\mid a_i$. Denote by $n(f\mid\boldsymbol{a}) := \#\{i : f\mid a_i\}$. Then (1.3) can be written as

$$\begin{aligned} W_f(t;\boldsymbol{a}) &= -\operatorname*{res}_{s=0}\sum_{\zeta\in\Theta_f}\frac{\zeta^{-t}\mathrm{e}^{-ts}}{\prod_{i=1}^N(1-\zeta^{a_i}\mathrm{e}^{a_i s})} \\ &= -\operatorname*{res}_{s=0}\sum_{\zeta\in\Theta_f}\frac{\zeta^{-t}}{\prod_{i=1}^N(1-\zeta^{a_i}\mathrm{e}^{a_i s})}\sum_{m\ge 0}\frac{(-ts)^m}{m!} \end{aligned}$$

$$= \sum_{m\geq 0} \frac{(-1)^{m+1}}{m!} t^m \operatorname*{res}_{s=0} s^m \sum_{\zeta\in\Theta_f} \frac{\zeta^{-t}}{\prod_{i=1}^{N}(1-\zeta^{a_i}\mathrm{e}^{a_i s})} \tag{2.3}$$

$$= \sum_{m=0}^{n(f|\boldsymbol{a})-1} \frac{(-1)^{n(f|\boldsymbol{a})+m+1}}{m!\prod_{i:f|a_i} a_i} t^m [s^{n(f|\boldsymbol{a})-1-m}] \sum_{\zeta\in\Theta_f} \mathcal{W}_f(t;\boldsymbol{a},\zeta),$$

where

$$\mathcal{W}_f(t;\boldsymbol{a},\zeta) = \zeta^{-t} \prod_{i:f\nmid a_i} v_i(\zeta) \prod_{i:f|a_i} g(a_i s) \prod_{i:f\nmid a_i} g(a_i s, v_i(\zeta)-1),$$

$v_i(x) = \langle -\frac{1}{f} x^{a_i}\theta_f'(x^{a_i})\rangle_R$ is obtained by Lemma 2.8 and the fact that $\Phi_f(\zeta) = 0$ for any $\zeta \in \Theta_f$, and $[s^{n(f|\boldsymbol{a})-1-m}]\mathcal{W}_f(t;\boldsymbol{a},\zeta)$ must be equal to 0 when $m \geq n(f \mid \boldsymbol{a})$ since $\mathcal{W}_f(t;\boldsymbol{a},\zeta)$ is a power series in s.

Note that we need to compute such $v_i(x)$ at most $f-1$ times. For any positive integer a satisfying $f \nmid a$, we have $\zeta^a = \zeta^{a \pmod f}$. Therefore, we only need to compute $\langle -\frac{1}{f} x^a \theta_f'(x^a)\rangle_R$ for some $1 \leq a \leq f-1$.

The problem simplifies to the computation of $[s^{n(f|\boldsymbol{a})-1-m}] \sum_{\zeta\in\Theta_f} \mathcal{W}_f(t;\boldsymbol{a},\zeta)$ for all $0 \leq m \leq n(f \mid \boldsymbol{a}) - 1$. The most straightforward scenario occurs when $f = 1$. In this case, $n(1 \mid \boldsymbol{a}) = N$, $\Theta_1 = \{1\}$, and

$$\mathcal{W}_1(t;\boldsymbol{a},1) = \prod_{i=1}^{N} g(a_i s).$$

Applying Theorem 2.6 to compute

$$\mathcal{W}_1(t;\boldsymbol{a},1) \pmod{\langle s^N\rangle} = \sum_{k=0}^{N-1} A_k s^k, \quad \text{where} \quad A_k \in \mathbb{Q}. \tag{2.4}$$

Substituting (2.4) into (2.3) yields

$$W_1(t;\boldsymbol{a}) = \Big[\sum_{m=0}^{N-1} \frac{(-1)^{N+m+1}}{m!\prod_{i=1}^{N} a_i} A_{N-1-m} t^m \Big]. \tag{2.5}$$

In the case when $f > 1$, we apply Theorem 2.6 to determine

$$\prod_{i:f|a_i} g(a_i s) \prod_{i:f\nmid a_i} g(a_i s, v_i(\zeta)-1) \pmod{\langle s^{n(f|\boldsymbol{a})}\rangle} = \sum_{k=0}^{n(f|\boldsymbol{a})-1} \widehat{M}_k(\zeta) s^k, \tag{2.6}$$

where $\widehat{M}_k(x)$ is a polynomial in x for each k. Consequently, we obtain

$$[s^{n(f|\boldsymbol{a})-1-m}]\mathcal{W}_f(t;\boldsymbol{a},\zeta) = \zeta^{-t} M_m(\zeta), \tag{2.7}$$

where

$$M_m(x) = \Big\langle \widehat{M}_{n(f|\boldsymbol{a})-1-m}(x) \prod_{i:f\nmid a_i} v_i(x) \Big\rangle_R. \tag{2.8}$$

Theorem 2.9. *Under the aforementioned notations, for each* $0 \le m \le n(f \mid \boldsymbol{a}) - 1$, *assume*

$$M_m(x)\widehat{\Phi}_f(x)\Phi_f'(x) \equiv \sum_{i=0}^{f-1} c_{m,i}x^i \pmod{\langle x^f - 1\rangle}. \tag{2.9}$$

Then

$$[s^{n(f|\boldsymbol{a})-1-m}] \sum_{\zeta\in\Theta_f} \mathcal{W}_f(t;\boldsymbol{a},\zeta) = c_{m,\hat{t}-1}, \quad \text{where} \quad \hat{t} \equiv t \pmod{f}. \tag{2.10}$$

Furthermore, by substituting (2.10) *into* (2.3), *we find that* $W_f(t;\boldsymbol{a}) = [P_1(t), \ldots, P_f(t)]$, *where*

$$P_i(t) = \sum_{m=0}^{n(f|\boldsymbol{a})-1} \frac{(-1)^{n(f|\boldsymbol{a})+m+1}}{m!\prod_{i:f|a_i} a_i} c_{m,i-1}t^m. \tag{2.11}$$

Proof. A straightforward computation yields

$$\begin{aligned}
[s^{n(f|\boldsymbol{a})-1-m}] \sum_{\zeta\in\Theta_f} \mathcal{W}_f(t;\boldsymbol{a},\zeta) &= \sum_{\zeta\in\Theta_f} \zeta^{-t} M_m(\zeta) && \text{(by (2.7))} \\
&= \frac{1}{f} \sum_{\zeta:\zeta^f=1} \zeta\zeta^{-t} \sum_{i=0}^{f-1} c_{m,i}\zeta^i && \text{(by Lemma 2.1)} \\
&= \frac{1}{f} \sum_{i=0}^{f-1} c_{m,i} \sum_{\zeta:\zeta^f=1} \zeta^{i+1-\hat{t}} = \frac{1}{f} c_{m,\hat{t}-1} \sum_{\zeta:\zeta^f=1} \zeta^0 = c_{m,\hat{t}-1},
\end{aligned}$$

where the penultimate equation holds because $\sum_{\zeta:\zeta^f=1} \zeta^j = 0$ for any $j \pmod f \neq 0$. □

Example 2.10. Returning to Example 1.1, recall $\boldsymbol{a} = (1, 3, 6)$. We provide the details of the computation of $W_3(t;\boldsymbol{a})$ to explain our computation scheme. It is obvious that $n(3 \mid \boldsymbol{a}) = 2$. By Lemma 2.8,

$$\frac{1}{1-\zeta} = \Big\langle -\frac{1}{3}x(2x+1) \Big\rangle_R \Big|_{x=\zeta} = \frac{\zeta+2}{3}$$

holds for any $\zeta \in \Theta_3$. According to (2.3),

$$W_3(t;\boldsymbol{a}) = \sum_{m=0}^{1} \frac{(-1)^{m+1}}{18\ m!} t^m [s^{1-m}] \sum_{\zeta\in\Theta_3} \mathcal{W}_3(t;\boldsymbol{a},\zeta),$$

where

$$\mathcal{W}_3(t;\boldsymbol{a},\zeta) = \zeta^{-t}\ \frac{\zeta+2}{3}\ g(3s)\ g(6s)\ g\Big(s, \frac{\zeta-1}{3}\Big).$$

We apply Theorem 2.6 to determine

$$g(3s)\ g(6s)\ g\Big(s, \frac{\zeta-1}{3}\Big) \pmod{\langle s^2\rangle} = 1 + \frac{2\zeta-29}{6} s.$$

By (2.8),

$$\begin{aligned} M_0(x) &= \frac{x+2}{3}\ \frac{2\ x-29}{6} \pmod{\langle \Phi_3(x)\rangle} = -\frac{3\ x}{2} - \frac{10}{3}, \\ M_1(x) &= \frac{x+2}{3}\cdot 1 \pmod{\langle \Phi_3(x)\rangle} = \frac{x+2}{3}. \end{aligned}$$

Furthermore, we have $\widehat{\Phi}_3(x) = \frac{x^3-1}{\Phi_3(x)} = x-1$ and $\Phi_3'(x) = 2x+1$. Then we can compute

$$\begin{aligned} M_0(x)\widehat{\Phi}_3(x)\Phi_3'(x) &\equiv -\frac{31\ x^2}{6} + \frac{29\ x}{6} + \frac{1}{3} \pmod{\langle x^3-1\rangle}, \\ M_1(x)\widehat{\Phi}_3(x)\Phi_3'(x) &\equiv x^2 - x \pmod{\langle x^3-1\rangle}. \end{aligned}$$

Finally, by Theorem 2.9, $W_3(t;\boldsymbol{a}) = [P_1(t), P_2(t), P_3(t)]$, where

$$\begin{aligned} P_1(t) &= \frac{-1}{0!\times 18}\cdot\frac{1}{3} + \frac{1}{1!\times 18} t\cdot 0 = -\frac{1}{54}, \\ P_2(t) &= \frac{-1}{0!\times 18}\cdot\frac{29}{6} + \frac{1}{1!\times 18} t\cdot(-1) = -\frac{t}{18} - \frac{29}{108}, \\ P_3(t) &= \frac{-1}{0!\times 18}\cdot\Big(-\frac{31}{6}\Big) + \frac{1}{1!\times 18} t\cdot 1 = \frac{t}{18} + \frac{31}{108}. \end{aligned}$$

The result is in perfect agreement with that presented in Example 1.1.

2.3. Algorithm and complexity analysis

We present Algorithm `Cyc-Denum` to encapsulate the computation of $W_f(t;\boldsymbol{a})$. This algorithm has been implemented in the namesake `Maple` package, and its complexity is asserted in Theorem 2.11.

Algorithm 1: `Cyc-Denum`.

Input: A positive integer sequence $\boldsymbol{a}$ with $\gcd(\boldsymbol{a}) = 1$.
A (symbolic) nonnegative integer t.
Output: The list of $W_f(t;\boldsymbol{a})$ as described in (1.2).
1 Let S be the set of all divisors of the entries of $\boldsymbol{a}$.
2 **for** $f \in S$ **do**
3 **if** $f = 1$ **then**
4 Compute (2.4) and obtain $W_1(t;\boldsymbol{a})$ by (2.5).
5 **else**
6 Compute $v_i(x) = \langle -\frac{1}{f} x^{a_i} \theta_f'(x^{a_i}) \rangle_R$ for all a_i satisfying $f \nmid a_i$, and derive (2.6) using Theorem 2.6.
7 For each $0 \le m \le n(f \mid \boldsymbol{a}) - 1$, compute $M_m(x)$ as in (2.8) and $\sum_{i=0}^{f-1} c_{m,i} x^i$ as in (2.9).
8 Obtain $W_f(t;\boldsymbol{a}) = [P_1(t), P_2(t), \dots, P_f(t)]$, where $P_i(t)$'s are as in (2.11).
9 **return** The list of $W_f(t;\boldsymbol{a})$ for all $f \in S$.

Theorem 2.11. *Let $\boldsymbol{a} = (a_1, a_2, \dots, a_N)$ be a sequence of positive integers with $\gcd(\boldsymbol{a}) = 1$ and f be a factor of certain a_i.*

(1) *Algorithm `Cyc-Denum` correctly computes $W_1(t;\boldsymbol{a})$ using $O(N \log^2(N))$ operations in $\mathbb{Q}$.*
(2) *For $f > 1$, Algorithm `Cyc-Denum` correctly computes $W_f(t;\boldsymbol{a})$ using $O(d)$ operations in $\mathbb{Q}[x]/\langle x^f - 1\rangle$ and $O\big(fd\log(d) + N\log^2(d)\big)$ operations in R, where $d = n(f \mid \boldsymbol{a})$.*

Proof. In Step 0, we compute $\prod_{i:f\mid a_i} a_i$ and $m!$ for $0 \le m \le n(f \mid \boldsymbol{a}) - 1$ using $O(d)$ operations in $\mathbb{Q}$ (or $O(N)$ operations in $\mathbb{Q}$ if $f = 1$).

For part (1), we proceed to compute (2.4) by Theorem 2.6 using $O(N\log^2(N))$ operations in $\mathbb{Q}$ and obtain $W_1(t;\boldsymbol{a})$ by (2.5) using $O(N)$ operations in $\mathbb{Q}$. Then the total complexity is clearly $O(N\log^2(N))$ operations in $\mathbb{Q}$.

For part (2), we conclude the computation with the following steps.

In Step 1, we compute $v_i(x) = \langle -\frac{1}{f} x^{a_i} \theta_f'(x^{a_i}) \rangle_R$ for all $i \in \{i : f \nmid a_i\}$ using $O(f)$ operations in R.

In Step 2, we compute $\langle \prod_{i:f\nmid a_i} v_i(x) \rangle_R$ using $O(N - d)$ operations in R.

In Step 3, we utilize Theorem 2.6 to compute (2.6). This step uses at most $O\big(fd\log(d) + N\log^2(d)\big)$ operations in R.

In Step 4, we compute $M_m(x)$ as in (2.8) for all $0 \le m \le d-1$ using $O(d)$ operations in R.

In Step 5, we compute $\widehat{\Phi}_f(x)\Phi_f'(x)$ and (2.9) for all $0 \le m \le d-1$ using $O(d)$ operations in $\mathbb{Q}[x]/\langle x^f - 1\rangle$.

In Step 6, we derive $P_i(t)$ as in (2.11) for all $1 \le i \le f$ using $O(f)$ operations in $\mathbb{Q}$.

Then the total complexity is the sum of the complexities of all the above steps, yielding the asserted bound. □

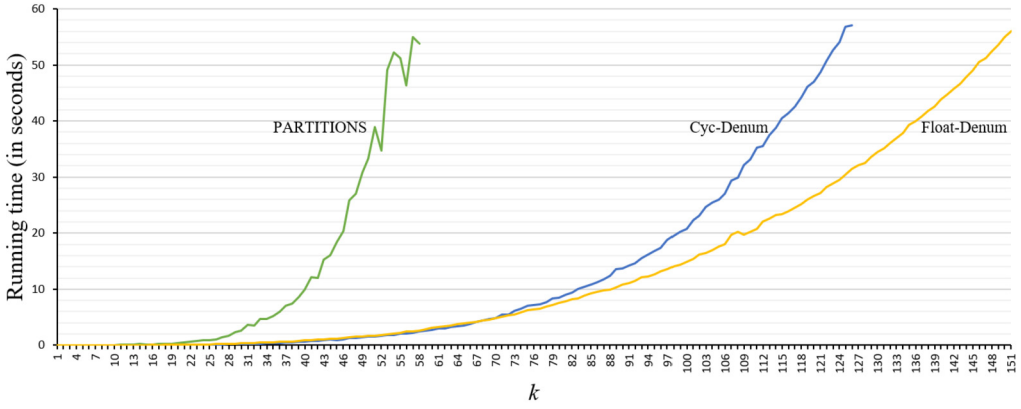
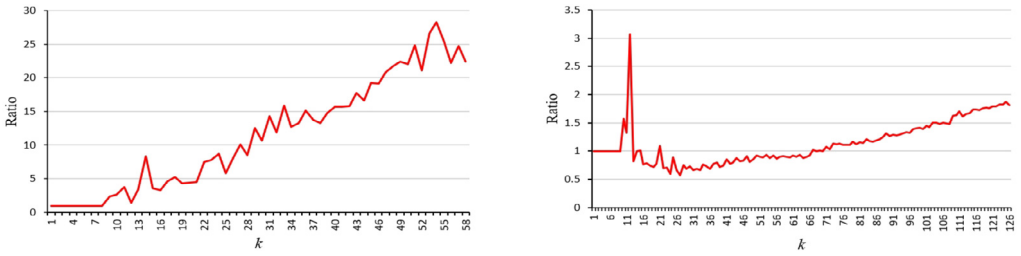


Fig. 1. Running time (in seconds) for computing $d(t; 1, 2, \ldots, k)$. (For interpretation of the colors in the figure, the reader is referred to the web version of this article.)



(a) `PARTITIONS/Cyc-Denum` (b) `Cyc-Denum/Float-Denum`

Fig. 2. The ratio of running time for computing $d(t; 1, 2, \ldots, k)$ between different algorithms.

2.4. Experiments

In this subsection, we report the results of our computational experiments, which are all achieved on the same personal laptop.

One experiment involves computing $d(t; 1, 2, \ldots, k)$ using the algorithms `PARTITIONS`, `Cyc-Denum`, and `Float-Denum` (the latter described in Section 3). To ensure the reliability of the experimental data, we repeated each computation three times and recorded the average running time. We retained only data with running times under 1 minute and plotted the results (see Fig. 1). `PARTITIONS` (green line) computes $d(t; 1, 2, \ldots, k)$ within 1 minute for all $k \leq 58$. `Cyc-Denum` (blue line) achieves this for $k \leq 126$, and `Float-Denum` (yellow line) for $k \leq 151$. The running time ratios of these algorithms are shown in Fig. 2. Note that the ratios for small k should be considered irrelevant because the computation time of all the above three algorithms is close to 0 seconds.

Another experiment is computing $d(t; \boldsymbol{a})$ for random sequences $\boldsymbol{a}$ with $1 \leq a_i \leq 500$ $(1 \leq i \leq N)$. We increase N sequentially from $N = 2$ and compute 100 random sequences for each N. Because of the high memory of partial fraction decomposition, `PARTITIONS` does not perform well in this experiment. For instance, the computation

Table 1
The number of successful computations in 1 and 1.5 minutes for $56 \le N \le 63$.

N	56	57	58	59	60	61	62	63
The number of successful computations in 1 minute	97	97	85	77	58	45	26	19
The number of successful computations in 1.5 minutes	100	100	100	100	100	97	95	89

of $\boldsymbol{a} = (428, 393)$ is out of reach for `PARTITIONS`, while `Cyc-Denum` completed the computation in 0.156 seconds. Therefore, we focus on the experimental data of `Cyc-Denum`. It successfully computes the denumerant of each $\boldsymbol{a}$ in 1 minute for $N \le 55$ and in 2 minutes for $N \le 63$. The number of successful computations in 1 and 1.5 minutes for $56 \le N \le 63$ are shown in Table 1.

3. Floating-point computation

Another idea for computing $d(t; \boldsymbol{a})$ is converting each root of unity ζ to a floating-point number. Then all computations are over complex field $\mathbb{C}$. Although this method does not yield precise results, the low memory requirements enable the prediction of $d(t; \boldsymbol{a})$ when N is significantly large.

For a fixed f, let $\eta_f \in \mathbb{C}$ be the floating-point value of $\mathrm{e}^{\frac{2\pi\sqrt{-1}}{f}}$. Rewrite

$$W_f(t; \boldsymbol{a}) = \sum_{m=0}^{n(f|\boldsymbol{a})-1} \frac{(-1)^{n(f|\boldsymbol{a})+m+1}}{m! \prod_{i:f|a_i} a_i} t^m [s^{n(f|\boldsymbol{a})-1-m}] \sum_{j \in J_f} \mathcal{V}_f(t; \boldsymbol{a}, j), \tag{3.1}$$

where $J_f := \{j : 1 \le j \le f \text{ and } \gcd(j, f) = 1\}$, and

$$\mathcal{V}_f(t; \boldsymbol{a}, j) = \frac{\eta_f^{-jt}}{\prod_{i:f\nmid a_i}(1 - \eta_f^{ja_i})} \prod_{i:f|a_i} g(a_i s) \prod_{i:f\nmid a_i} g\Big(a_i s, \frac{\eta_f^{ja_i}}{1 - \eta_f^{ja_i}}\Big).$$

In fact, Theorem 2.6 is applicable to the case when $\mathbb{Q} \subseteq R$ (hence to $R = \mathbb{C}$) (See [17]). Then for each $j \in J_f$, we can use Theorem 2.6 to figure out

$$\prod_{i:f|a_i} g(a_i s) \prod_{i:f\nmid a_i} g\Big(a_i s, \frac{\eta_f^{ja_i}}{1 - \eta_f^{ja_i}}\Big) \pmod{\langle s^{n(f|\boldsymbol{a})} \rangle} = \sum_{k=0}^{n(f|\boldsymbol{a})-1} A_k(f, j) s^k, \tag{3.2}$$

where $A_k(f, j) \in \mathbb{C}$ is a constant complex number with respect to f and j for each k. And

$$[s^{n(f|\boldsymbol{a})-1-m}] \mathcal{V}_f(t; \boldsymbol{a}, j) = \frac{A_{n(f|\boldsymbol{a})-1-m}(f, j)}{\prod_{i:f\nmid a_i}(1 - \eta^{ja_i})} \Big[\eta^{-j}, \eta^{-2j}, \ldots, \eta^{-jf}\Big] \tag{3.3}$$

is a quasi-polynomial in t. Substituting (3.3) into (3.1) gives $W_f(t;\boldsymbol{a}) = [P_1(t),\dots,P_f(t)]$, where

$$P_i(t) = \frac{1}{\prod_{i:f|a_i} a_i} \sum_{m=0}^{n(f|\boldsymbol{a})-1} \frac{(-1)^{n(f|\boldsymbol{a})+m+1}}{m!} t^m \sum_{j\in J_f} \frac{\eta^{-ij} A_{n(f|\boldsymbol{a})-1-m}(f,j)(\zeta)}{\prod_{i:f\nmid a_i}(1-\eta^{ja_i})}. \tag{3.4}$$

The following algorithm clarifies the procedure for computing $d(t;\boldsymbol{a})$ by floating-point computation scheme. Additionally, Theorem 3.1 provides the complexity analysis of the computation of $W_f(t;\boldsymbol{a})$ under this scheme.

Algorithm 2: `Float-Denum`.

Input: A positive integer sequence $\boldsymbol{a}$ with $\gcd(\boldsymbol{a}) = 1$.
A (symbolic) nonnegative integer t.
Output: The list of $W_f(t;\boldsymbol{a})$ as described in (1.2).

Let S be the set of all divisors of the entries of $\boldsymbol{a}$.
for $f \in S$ **do**
  for $j \in J_f$ **do**
    Compute $\frac{1}{1-\eta^{ja_i}}$ (hence $\frac{\eta^{ja_i}}{1-\eta^{ja_i}}$) for all a_i satisfying $f \nmid a_i$, and obtain (3.2) by Theorem 2.6.
    Compute η^{ji} for $1 \le i \le f$ and obtain $[s^{n(f|\boldsymbol{a})-1-m}]\mathcal{V}_f(t;\boldsymbol{a},j)$ by (3.3) for each $0 \le m \le n(f \mid \boldsymbol{a}) - 1$.
  Obtain $W_f(t;\boldsymbol{a}) = [P_1(t), P_2(t),\dots,P_f(t)]$, where $P_i(t)$'s are as in (3.4).
return The list of $W_f(t;\boldsymbol{a})$ for all $f \in S$.

Theorem 3.1. *Let $\boldsymbol{a} = (a_1, a_2, \dots, a_N)$ be a positive integer sequence satisfying $\gcd(\boldsymbol{a}) = 1$ and f be a factor of certain a_i. Algorithm `Float-Denum` correctly computes $W_f(t;\boldsymbol{a})$ using $O\big((fd\log(d) + N\log^2(d))\varphi(f)\big)$ operations in $\mathbb{C}$, where $d = n(f \mid \boldsymbol{a})$.*

Proof. We complete the computation by the following steps.

In Step 0, we compute $\prod_{i:f|a_i} a_i$ and $m!$ for all $0 \le m \le n(f \mid \boldsymbol{a}) - 1$ using $O(d)$ operations in $\mathbb{C}$.

In Step 1, we compute η_f^i and $\frac{1}{1-\eta_f^i}$ (hence $\frac{\eta_f^i}{1-\eta_f^i}$) for $1 \le i \le f-1$ using $O(f)$ operations in $\mathbb{C}$. Then we can read the values of either η_f^a or $\frac{1}{1-\eta_f^a}$ for any positive integer a satisfying $f \nmid a$ since $\eta_f^a = \eta_f^{a \pmod f}$.

In Step 2, we compute $\frac{1}{\prod_{i:f\nmid a_i}(1-\eta^{ja_i})}$ for all $j \in J_f$ using $O(\varphi(f)(N-d))$ operations in $\mathbb{C}$.

In Step 3, we use Theorem 2.6 to obtain (3.2) using $O\big((fd\log(d) + N\log^2(d))\varphi(f)\big)$ operations in $\mathbb{C}$.

In Step 4, for all $j \in J_f$, we compute $[s^{n(f|\boldsymbol{a})-1-m}]\mathcal{V}_f(t;\boldsymbol{a},j)$ for $1 \le t \le f$ using $O(f\varphi(f))$ operations in $\mathbb{C}$.

In Step 5, we substitute all $[s^{n(f|\boldsymbol{a})-1-m}]\mathcal{V}_f(t;\boldsymbol{a},\zeta)$ into (3.1) to obtain $W_f(t;\boldsymbol{a})$ using $O(d)$ operations in $\mathbb{C}$.

The total complexity is the sum of all the above steps. □

Example 3.2. The results of Example 1.1 by Algorithm `Float-Denum` (by setting digits $= 10$) are as follows:

$$\begin{aligned}
W_1(t;\boldsymbol{a}) &\approx [0.02777777778\ t^2 + 0.2777777778\ t + 0.5879629633];\\
W_2(t;\boldsymbol{a}) &\approx [-0.04166666667, 0.04166666667]\\
W_3(t;\boldsymbol{a}) &\approx [7.928932732 \times 10^{-12}\ t - 0.01851851851, -0.05555555557\ t - 0.2685185186,\\
&\qquad 0.05555555554\ t + 0.2870370370]\\
W_6(t;\boldsymbol{a}) &\approx [0.1666666667, 0.08333333344, -0.08333333332, -0.1666666667,\\
&\qquad -0.08333333339, 0.08333333323].
\end{aligned}$$

Now taking $t = 12$, we will obtain $d(12;\boldsymbol{a}) \approx 9.000000001$. It differs from the exact value 9 by 10^{-9}. Readers can verify other values.

Remark 3.3. The errors of Algorithm `Float-Denum` are related to the digits we set. For instance, the exact value of $d(1789682;1,3,6)$ is equal to 88971554961. If we set digits $= 10$, then we obtain $d(1789682;1,3,6) \approx 8.897155496 \times 10^{10}$. The error reaches 1. If we set digits $= 20$, then we will obtain $d(1789682;1,3,6) \approx 8.8971554961000000001 \times 10^{10}$ with error 10^{-9}, but the running time increases.

4. Concluding remarks

We have developed a fast algorithm for the Sylvester wave when $a_i \le C$ for a constant C (with default value 500 in our experiments). From Theorem 3.1 one sees that in such a case, the Sylvester wave can be computed in polynomial time in N. This extends Baldoni et al.'s result (only) on top coefficients.

Based on the equivalence between the wave function representation and partial fraction decomposition (see [11]), we observe that Algorithm `Cyc-Denum` can be adapted to efficiently compute partial fraction decompositions for proper rational functions sharing the denominator form specified in (1.1).

The idea is also adapted to give a fast algorithm for the Ehrhart series in an upcoming paper [16].

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No. 12071311).

References

[1] G. Agnarsson, On the Sylvester denumerants for general restricted partitions, in: Proceedings of the Thirtythird Southeastern International Conference on Combinatorics, Graph Theory and Computing (Boca Raton, FL, 2002), vol. 154, 2002, pp. 49–60.

[2] F. Aguiló–Gost, D. Llena, Computing denumerants in numerical 3-semigroups, Quaest. Math. 41 (2018) 1083–1116.
[3] G.E. Andrews, Partitions: at the interface of q-series and modular forms, Ramanujan J. 7 (2003) 385–400.
[4] V. Baldoni, N. Berline, J.A. De Loera, B.E. Dutra, M. Köppe, M. Vergne, Coefficients of Sylvester's denumerant, Integers 15 (2015) A11.
[5] A. Cayley, Researches on the partition of numbers, Philos. Trans. R. Soc. Lond. 146 (1856) 127–140.
[6] L.G. Fel, B.Y. Rubinstein, Sylvester waves in the Coxeter groups, Ramanujan J. 6 (2002) 307–329.
[7] J. von zur Gathen, J. Gerhard, Modern Computer Algebra, Cambridge Univ. Press, Cambridge, 2013.
[8] P. Lisoněk, Denumerants and their approximations, J. Comb. Math. Comb. Comput. 18 (1995) 225–232.
[9] C. O'Sullivan, Partitions and Sylvester waves, Ramanujan J. 47 (2018) 339–381.
[10] C. O'Sullivan, Revisiting the saddle-point method of Perron, Pac. J. Math. 298 (2019) 157–199.
[11] A.V. Sills, D. Zeilberger, Formulæ for the number of partitions of n into at most m parts (using the quasi-polynomial ansatz), Adv. Appl. Math. 48 (2012) 640–645.
[12] J.J. Sylvester, On the partition of numbers, Q. J. Math. 1 (1857) 141–152.
[13] N. Uday Kiran, An algebraic approach to q-partial fractions and Sylvester denumerants, Ramanujan J. 59 (2022) 671–712.
[14] G. Xin, C. Zhang, An algebraic combinatorial approach to Sylvester's denumerant, Ramanujan J. 66 (2025) 64.
[15] G. Xin, C. Zhang, Package `Cyc-Denum`, https://pan.baidu.com/s/16HlL3HNQpqLyhFBJHI2y6A?pwd=CycD, with passcode `CycD`.
[16] G. Xin, C. Zhang, Fast computation of Ehrhart series for magic labelling polytope, in preparation.
[17] G. Xin, Y. Zhang, Z. Zhang, Fast evaluation of generalized Todd polynomials: applications to MacMahon's partition analysis and integer programming, J. Symb. Comput. 133 (2025) 102420.