A Human-Guided AI Approach for Symbolic Cumulant Calculation

Bingye Yang [*1] **Jiashun Jin** [*2 3] **Yumou Qiu** [*1]

Abstract

Cumulants play an important role in probability and statistics, but explicit formula for high-order cumulants are hard to derive, especially in multivariate case. To tackle the problem, we develop a human-guided AI (hugAI) approach. First, we outline a clear strategy by developing a graph-based theoretical framework. Second, we divide the task into many small steps, and for each step, we communicate with AI (sometimes for several rounds) by carefully written prompts. Finally, the AI execute the formula with a python code. This way, our hugAI approach is able to execute formulas for cumulants of multivariate random variables at any given order. We have carefully validated the formulas with several approaches to make sure they are correct. As applications, we apply our results to Edgeworth expansion for multivariate random variables. Our study primarily uses GPT but we also investigate a handful other LLMs.

1. Introduction

Cumulant is a fundamental concept in probability and statistics, with a wide range applications. Suppose we have a p-variable random variable X and let $M_X(t)$ be the corresponding moment generating function (MGF). For possibly repetitive integers $i'_1, \ldots, i'_d \in \{1, \ldots, p\}$, the corrsponding cumulant is

$$\kappa_{i'_1,\ldots,i'_d} = \frac{\partial^d}{\partial t_{i'_1} \cdots \partial t_{i'_d}} \log M_{\mathbf{X}}(\mathbf{t})\Big|_{\mathbf{t}=\mathbf{0}}. \tag{1.1}$$

In many theoretical and practical situations, we wish to *derive a symbolic and explicit formula (SEF) for* $\kappa_{i'_1,\ldots,i'_d}$ *that is as simple as possible*. For lower-order cumulants, such an SEF is not hard to derive. For example, in the univariate case, the SEF for the first 4 cumulants are $\kappa_1 = \mu = \mathbb{E}[X]$, $\kappa_{1,1} = \mathrm{Var}(X)$, $\kappa_{1,1,1} = \mathbb{E}[(X-\mu)^3]$, and $\kappa_{1,1,1,1} = \mathbb{E}[(X-\mu)^4] - 3(\mathrm{Var}(X))^2$, respectively.

Our goal is to derive the SEF for $\kappa_{i'_1,\ldots,i'_d}$ for general $p, d, i'_1, i_2, \ldots, i'_d$. The problem is of interest in many applications, including but not limited to Edgeworth expansion and deriving moderate deviation bounds (Saulis & Statulevicius, 1991). See Section 2.2 for more discussion.

How to derive the SEF for cumulants in multivariate case is a difficult problem. A classical approach is to use the Speed formula proposed by Terry Speed (Speed, 1983):

$$\kappa_{i'_1,\ldots,i'_d} = \sum_{\pi} (-1)^{|\pi|-1} \{(|\pi|-1)!\} \,\mathrm{prod}(\pi), \tag{1.2}$$

where π is any partition of $\{1, 2, \ldots, d\}$ (we omit the definition of $\mathrm{prod}(\pi)$ for space reasons; see our full paper for details). However, directly using the Speed formula in (1.2) is inefficient and computationally challenging: we may have $\mathrm{prod}(\pi_1) = \mathrm{prod}(\pi_2)$ for many pairs of π_1 and π_2 and the number of possible π is the Bell number $B(d)$ (which grows rapidly as d increases; e.g., $B(10) = 115975$). The advancement of AI technology provides new opportunity. However, A BruteForth AI approach where we directly feed (1.1) or (1.2) is inefficient and computationally slow.

Our proposal is human-guided AI (hugAI) approach. Using graph theory and combinatorics, we discover a new Speed-type formula. To apply the formula, we divide the task into many small steps, and for each steps, we communicate with AI (sometimes with several rounds) with carefully written prompts, making sure that it fully understands the problem and provides the correct solution. After that, we ask the AI to write a python code and execute a SEF for $\kappa_{i'_1,i'_2,\ldots,i'_d}$, for any given input $(p, d, i'_1, i'_2, \ldots, i'_d)$. See Figure 1.

Compared with the approaches above (i.e., that of directly applying the Speed formula, or a brute-force use of an LLM), our approaches have two main advantages: it substantially improves the computation speed and directly produces the most simplified SEF.

[*]Equal contribution [1] School of Mathematical Sciences, Peking University, Beijing, China [2] Department of Statistics & Data Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA [3] School of Statistics and Data Science, Southeast University, Nanjing, Jiangsu, China . Correspondence to: Bingye Yang <byyang25@stu.pku.edu.cn>.

Accepted at the ICML 2026 Workshop on AI as a Tool for Mathematics, Computer Science, and Machine Learning, Seoul, South Korea, 2026.

2. Theoretical results with applications

2.1. A new Speed-type formula

Fix d possibly repetitive indices $i'_1, \ldots, i'_d$. Suppose only q of them are distinct, denoted by $i_1, \ldots, i_q$, with multiplicities $w_1, \ldots, w_q$ (note that $w_1 + \cdots + w_q = d$). For each $1 \le \ell \le q$, split (which is not necessarily unique; see below)

$$w_{1,\ell} + \cdots + w_{m_\ell,\ell} = w_\ell, \qquad w_{u,\ell} \ge 1. \tag{2.3}$$

For each split, construct a graph where (a), (b), (c), and (d).

(a) For color c_ℓ, create nodes $(1,\ell), \ldots, (m_\ell, \ell)$ and label node (u,ℓ) by $X_{i_\ell}^{w_{u,\ell}}$.

(b) Divide all nodes into k non-empty disjoint groups, requiring that each group contains at most one node from each color group.

(c) Put an edge between two nodes if and only if they are in the same group.

(d) Then each group is a complete component, there is no edge between two nodes with the same color, and an isolated node is also treated as a complete component.

We call the graph constructed in this way a fully labeled admissible graph (FLAG). Now suppose $\mathcal{G}$ has components $\mathcal{C}_1, \ldots, \mathcal{C}_k$. So the number of component of $\mathcal{G}$ is k. If Y_i is the label on node i, define

$$\mathrm{prod}(\mathcal{G}) = \prod_{j=1}^{k} \mathbb{E}\Big(\prod_{i \in \mathcal{C}_j} Y_i \Big).$$

For the fixed colored and labeled node set from one splitting solution, call a node permutation eligible if it only permutes nodes within the same color group and preserves node labels. Two FLAGs $\mathcal{G}$ and $\mathcal{H}$ are isomorphic, denoted by $\mathcal{G} \sim \mathcal{H}$, if $\mathcal{H}$ can be obtained from $\mathcal{G}$ by an eligible node permutation. Such a permutation only renames indistinguishable nodes; it preserves the component-wise moments and hence gives $\mathrm{prod}(\mathcal{H}) = \mathrm{prod}(\mathcal{G})$. Thus one isomorphic class corresponds to one symbolic moment product. We divide all constructed FLAGs first by number of components and then by isomorphism; for branch size k, let $b_{q,k}$ be the number of isomorphic classes and pick one representative $\mathcal{G}_{q,k,t}$ from each class. This gives

$$\mathcal{F} = \{\mathcal{G}_{q,k,t} : 1 \le t \le b_{q,k},\ 1 \le k \le d\}.$$

Finally, suppose that out of the k component in the set $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2 \ldots, \mathcal{C}_k\}$, only $\tilde{k}$ of them are distinct, which appear in $\mathcal{C}$ for $r_1, r_2, \ldots, r_{\tilde{k}}$ times, respectively. Then the redundancy for $\mathcal{G}$ is $r(\mathcal{G}) = r_1! r_2! \ldots r_{\tilde{k}}!$

Theorem 2.1 (A new Speed-type formula)**.** *For $i'_1, \ldots, i'_d \in \{1, \ldots, p\}$ with q unique indices, the cumulant $\kappa_{i'_1,\ldots,i'_d}$ can be expressed as*

$$\kappa_{i'_1,\ldots,i'_d} = \sum_{k=1}^{d} \sum_{t=1}^{b_{q,k}} \frac{a_{q,k,t}}{r(\mathcal{G}_{q,k,t})} \,\mathrm{prod}(\mathcal{G}_{q,k,t}), \tag{2.4}$$

where $a_{q,k,t} = (-1)^{k-1}(k-1)! \times \prod_{\ell=1}^{q} \frac{w_\ell!}{\prod_{u=1}^{m_\ell} w_{u,\ell}!}$.

Because each $\mathrm{prod}(\mathcal{G}_{q,k,t})$ is a distinct symbolic moment product, (2.4) provides the most simplified formula for symbolic calculation of cumulants.

2.2. Application to Edgeworth expansion

Let $X_1, X_2, \ldots, X_n$ be iid samples from a p-dimensional distribution with mean $\boldsymbol{\mu}$ and identity covariance matrix $\mathbf{I}_p$. Let

$$\mathbf{Z}_n = \sqrt{n}(\bar{\mathbf{X}}_n - \boldsymbol{\mu}).$$

Let $\phi_p(\mathbf{z})$ be the density function of $N(\mathbf{0}, \mathbf{I}_p)$. The multivariate Edgeworth expansion for $\mathbf{Z}_n$ has density function

$$f_n(\mathbf{z}) = \phi_p(\mathbf{z}) \Big\{ 1 + \sum_{j^*=1}^{s^*} n^{-\frac{j^*}{2}} Q_{j^*}(\mathbf{z}) \Big\} + o(n^{-\frac{s^*}{2}}), \tag{2.5}$$

where each $Q_{j^*}(\mathbf{z})$ is a polynomial. In general, computing $Q_{j^*}(\mathbf{z})$ is already a complicated symbolic task: it is a polynomial in $\mathbf{z}$ whose coefficients are complicated functions of cumulants. Our Edgeworth result gives an explicit representation of $Q_{j^*}(\mathbf{z})$ in terms of cumulants and Hermite polynomials. Substituting the SEF of cumulants from (2.4) then gives an Edgeworth expansion whose coefficients are written only by expectations products of $\mathbf{X}_1$. We skip the detailed combinatorial form of $Q_{j^*}(\mathbf{z})$ here and focus on this substitution-based symbolic computation. We provide several examples in section 3.

3. Algorithms

The graph-based formula reduces the SEF problem to constructing $\mathcal{F}$, computing $a_{q,k,t}$ and $r(\mathcal{G}_{q,k,t})$, and applying (2.4). We use AI through three prompts. The first prompt generates all pairwise non-isomorphic FLAGs for a fixed set of nodes. The second prompt repeats this step over all solutions of (2.3). The third prompt computes $a_{q,k,t}$, $r(\mathcal{G}_{q,k,t})$, and $\mathrm{prod}(\mathcal{G}_{q,k,t})$, and then aggregates identical total products.

Algorithm 1. HugAI approach for computing SEF of κ **Input:** $i'_1, \ldots, i'_d$

1: Obtain the distinct indices $i_1, \ldots, i_q$ from $i'_1, \ldots, i'_d$, and record their multiplicities $w_1, \ldots, w_q$.

2: **Prompt 1: FLAGs for a fixed node set.** For one fixed solution of (2.3), construct nodes (u, ℓ) with color c_ℓ and label $X_{i_\ell}^{w_{u,\ell}}$. Ask AI to generate all FLAGs on this fixed colored and labeled node set, identify isomorphic FLAGs, and return one representative from each isomorphic class for every branch size k.

3: **Prompt 2: enumeration over all splitting solutions.** Ask AI to enumerate all solutions of 2.3 for every $1 \le \ell \le q$. For each splitting solution, apply Prompt 1 and collect all representatives $\mathcal{G}_{q,k,t}$.

4: **Prompt 3: SEF aggregation.** For each representative $\mathcal{G}_{q,k,t}$, compute the number $a_{q,k,t}$, the redundancy $r(\mathcal{G}_{q,k,t})$, and the total product $\mathrm{prod}(\mathcal{G}_{q,k,t})$. Then aggregate identical total products.

5: Substitute these quantities into the graph-based formula (2.4).

Output: The simplified expression formula for

$$\kappa_{i'_1,\ldots,i'_d} = \sum_{k=1}^{d} \sum_{t=1}^{b_{q,k}} \frac{a_{q,k,t}}{r(\mathcal{G}_{q,k,t})} \ \mathrm{prod}(\mathcal{G}_{q,k,t}),$$

with identical total products merged.

The generated cumulant output is a symbolic expression in raw moments. For a multivariate random vector $(X_1, X_2, X_3)^\top$, here is an example.

$$\begin{aligned}
\kappa_{1,1,1,2,3} &= \mathbb{E}\left(X_1^3 X_2 X_3\right) - 3\mathbb{E}(X_1)\,\mathbb{E}\left(X_1^2 X_2 X_3\right) - \mathbb{E}\left(X_1^3 X_3\right)\mathbb{E}(X_2) \\
&- \mathbb{E}\left(X_1^3 X_2\right)\mathbb{E}(X_3) - 3\mathbb{E}(X_1 X_3)\,\mathbb{E}\left(X_1^2 X_2\right) - 3\mathbb{E}(X_1 X_2 X_3)\,\mathbb{E}\left(X_1^2\right) \\
&- 3\mathbb{E}(X_1 X_2)\,\mathbb{E}\left(X_1^2 X_3\right) - \mathbb{E}\left(X_1^3\right)\mathbb{E}(X_2 X_3) + 6\mathbb{E}(X_1)\,\mathbb{E}\left(X_1^2 X_3\right)\mathbb{E}(X_2) \\
&+ 6\mathbb{E}(X_1)\,\mathbb{E}\left(X_1^2 X_2\right)\mathbb{E}(X_3) + 2\mathbb{E}\left(X_1^3\right)\mathbb{E}(X_3)\,\mathbb{E}(X_2) \\
&+ 12\mathbb{E}(X_1 X_3)\,\mathbb{E}(X_1 X_2)\,\mathbb{E}(X_1) + 6\mathbb{E}(X_1 X_3)\,\mathbb{E}\left(X_1^2\right)\mathbb{E}(X_2) \\
&+ 6\mathbb{E}(X_1 X_2 X_3)\,\mathbb{E}^2(X_1) + 6\mathbb{E}(X_1 X_2)\,\mathbb{E}\left(X_1^2\right)\mathbb{E}(X_3) \\
&+ 6\mathbb{E}(X_1)\,\mathbb{E}\left(X_1^2\right)\mathbb{E}(X_2 X_3) - 18\mathbb{E}(X_1)\,\mathbb{E}\left(X_1^2\right)\mathbb{E}(X_3)\,\mathbb{E}(X_2) \\
&- 18\mathbb{E}(X_1 X_3)\,\mathbb{E}^2(X_1)\,\mathbb{E}(X_2) - 18\mathbb{E}(X_1 X_2)\,\mathbb{E}^2(X_1)\,\mathbb{E}(X_3) \\
&- 6\mathbb{E}^3(X_1)\,\mathbb{E}(X_2 X_3) + 24\mathbb{E}^3(X_1)\,\mathbb{E}(X_3)\,\mathbb{E}(X_2)\,.
\end{aligned}$$

Figure 1. Symbolic expression for $\kappa_{1,1,1,2,3}$

Algorithm 2. Edgeworth expansion by cumulant SEFs **Input:** p, s^*

1: **for** $j^* = 1, \ldots, s^*$ **do**

2: Write the Edgeworth polynomial $Q_{j^*}(\mathbf{z})$ in its standard form, whose coefficients are cumulants of $\mathbf{X}_1$ and whose polynomial part is given by Hermite polynomials $H_{\boldsymbol{\alpha}}(\mathbf{z})$.

3: Identify every cumulant appearing in the coefficients of $Q_{j^*}(\mathbf{z})$.

4: Replace each cumulant by its SEF in moments obtained from (2.4).

5: Expand the expression, merge identical moment products, and simplify the coefficients.

6: **end for**

7: Substitute $Q_1(\mathbf{z}), \ldots, Q_{s^*}(\mathbf{z})$ into the density expansion (2.5).

Output: $Q_1(\mathbf{z}), Q_2(\mathbf{z}), \ldots, Q_{s^*}(\mathbf{z})$

For a concrete two-dimensional Edgeworth example, take $p = 2$ and $s^* = 2$. Then

$$\begin{aligned}
f_n(z_1, z_2) = \phi_2(z_1, z_2)\{1 + n^{-1/2} Q_1 + n^{-1} Q_2\} \\
+ O(n^{-3/2}).
\end{aligned}$$

The first and second correction is

$$Q_1(z_1, z_2) = a_{10}^{(1)} z_1 + a_{01}^{(1)} z_2 + a_{30}^{(1)} z_1^3 + a_{21}^{(1)} z_1^2 z_2 + a_{12}^{(1)} z_1 z_2^2 + a_{03}^{(1)} z_2^3,$$

$$\begin{aligned}
Q_2(z_1, z_2) = {} & a_{00}^{(2)} + a_{20}^{(2)} z_1^2 + a_{02}^{(2)} z_2^2 + a_{11}^{(2)} z_1 z_2 + a_{40}^{(2)} z_1^4 + a_{04}^{(2)} z_2^4 \\
& + a_{31}^{(2)} z_1^3 z_2 + a_{13}^{(2)} z_1 z_2^3 + a_{22}^{(2)} z_1^2 z_2^2 + a_{60}^{(2)} z_1^6 + a_{06}^{(2)} z_2^6 \\
& + a_{51}^{(2)} z_1^5 z_2 + a_{15}^{(2)} z_1 z_2^5 + a_{42}^{(2)} z_1^4 z_2^2 + a_{24}^{(2)} z_1^2 z_2^4 + a_{33}^{(2)} z_1^3 z_2^3.
\end{aligned}$$

And some of the coefficients are shown as follows,

$$\begin{aligned}
a_{10}^{(1)} &= -\frac{1}{2}\mathbb{E}[X_1^3] - \frac{1}{2}\mathbb{E}[X_1 X_2^2],\ a_{01}^{(1)} = -\frac{1}{2}\mathbb{E}[X_1^2 X_2] - \frac{1}{2}\mathbb{E}[X_2^3], \\
a_{30}^{(1)} &= \frac{1}{6}\mathbb{E}[X_1^3],\ a_{21}^{(1)} = \frac{1}{2}\mathbb{E}[X_1^2 X_2],\ a_{12}^{(1)} = \frac{1}{2}\mathbb{E}[X_1 X_2^2],\ a_{03}^{(1)} = \frac{1}{6}\mathbb{E}[X_2^3].
\end{aligned}$$

$$\begin{aligned}
a_{00}^{(2)} = {} & -\frac{1}{24}\Big(9\mathbb{E}[X_1^2]^2 + 6\mathbb{E}[X_1^2]\mathbb{E}[X_2^2] + 5\mathbb{E}[X_1^3]^2 + 6\mathbb{E}[X_1^3]\mathbb{E}[X_1 X_2^2] \\
& - 3\mathbb{E}[X_1^4] + 9\mathbb{E}[X_1^2 X_2]^2 + 6\mathbb{E}[X_1^2 X_2]\mathbb{E}[X_2^3] - 6\mathbb{E}[X_1^2 X_2^2] \\
& + 12\mathbb{E}[X_1 X_2]^2 + 9\mathbb{E}[X_1 X_2^2]^2 + 9\mathbb{E}[X_2^2]^2 + 5\mathbb{E}[X_2^3]^2 - 3\mathbb{E}[X_2^4]\Big).
\end{aligned}$$

Here the cumulants are obtained from the hugAI SEF and can then be simplified under special conditions such as zero mean and identity covariance matrix.

3.1. Computation cost

BruteForth enumerates all partitions in $\mathcal{P}(d)$ and then merges identical terms, while hugAI works directly on FLAG representatives in $\mathcal{F}$. Table 1 reports the running times of BruteForth and hugAI; Figure 2 compares the running times of code generated by ChatGPT, Gemini, and DeepSeek using the same prompts and the same canonical symbolic format.

Table 1. Running times of BruteForth and hugAI for trivariate cumulants.

$[w_1, w_2, w_3]$	terms	Brute	hugAI
$[3, 3, 3]$	686	$3.27{\cdot}10^{-1}$	$5.74{\cdot}10^{-2}$
$[3, 3, 4]$	1429	1.83	$1.76{\cdot}10^{-1}$
$[3, 4, 4]$	3058	11.9	$6.57{\cdot}10^{-1}$
$[4, 4, 4]$	6721	89.2	2.87

3.2. Comparison of different approaches

We compare three ways to obtain symbolic cumulant expressions.

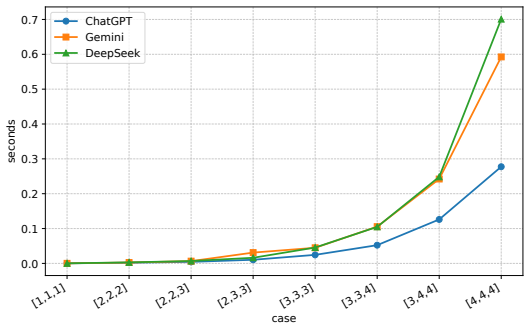

Figure 2. Running times of ChatGPT, Gemini, and DeepSeek implementations on the same trivariate cumulant cases.

Human coder. The most direct approach is to implement the partition formula by hand, enumerate all set partitions $\pi \in \mathcal{P}(d)$, compute $\mu(\pi)\mathrm{prod}(\pi)$, and then merge identical total products. This approach is correct in principle, but the implementation cost is high, the expressions are difficult to simplify, and the computation becomes slow as d increases.

BruteForth. A lighter approach is to prompt AI directly from the partition formula. Then AI performs the enumeration and aggregation automatically. This reduces human coding effort and can produce correct symbolic results, but it still works on $\mathcal{P}(d)$, so the computation remains slow for high-order cumulants.

hugAI. Our approach first uses the FLAG framework to work on representatives in $\mathcal{F}$ rather than all partitions in $\mathcal{P}(d)$. It then computes $a_{q,k,t}$, $r(\mathcal{G}_{q,k,t})$, and $\mathrm{prod}(\mathcal{G}_{q,k,t})$ through the structured prompts. Thus hugAI has the best computational performance and gives a compact SEF with merged coefficients, although it relies on the graph-based theoretical framework.

3.3. Comparison of different LLMs

We compare code generated by different LLMs under the same three prompts. For each LLM, we check whether the code from each step runs successfully and whether the final output gives the complete SEF. Symbol $\surd$ denotes success.

LLM	Step 1	Step 2	Step 3	Result
ChatGPT	$\surd$	$\surd$	$\surd$	$\surd$
DeepSeek	$\surd$	$\surd$	$\surd$	$\surd$
Gemini	$\surd$	$\surd$	$\surd$	$\surd$

Applying the same prompts to other LLMs may not solve all steps correctly, since different LLMs have different abilities in symbolic reasoning and coding. This is an additional problem to test LLM intelligence, and is separate work. A more fine-grained evaluation can further split each prompt into smaller steps and record how far each LLM can proceed.

LLM	1(a)	1(b)	2(a)	2(b)	3(a)	3(b)	Run
LLM A	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
LLM B	$\surd$	$\surd$	$\surd$	$\times$	–	–	$\times$
LLM C	$\surd$	$\times$	–	–	–	–	$\times$

4. Evaluation

We validate correctness and reproducibility from several directions. First, we evaluate code generated by different LLMs using the same three prompts, and check whether the generated programs produce a complete SEF, run successfully. Second, we compare our results with BruteForth AI (i.e., a bruteforce application of LLM to the Speed formula (1.2)). Finally, in special cases, cumulants are not hard to compute manually, so we can use these cases to verify our results. In the table below, $\surd$ means that the symbolic expression from the hugAI SEF is identical to the hand-simplified benchmark expression up to the ordering of terms. We consider three cases.

Case 1. $\mathbf{X} \sim N(\mathbf{0}, \mathbf{I}_p)$. In this case, all cumulants of order higher than two vanish. Thus the hand-simplified benchmark for every listed multiplicity vector is zero.

Case 2. The entries of $\mathbf{X} = (X_1, \ldots, X_p)^\top$ are iid from $\mathrm{exponential}(\lambda)$, where $\lambda = \mathbb{E}(X_j)$. Since the entries are independent, every mixed cumulant involving at least two distinct indices vanishes. Thus the hand-simplified benchmark for every listed multiplicity vector is zero.

Case 3. Let N_0, N_1, N_2, N_3 be independent, $N_j \sim \mathrm{Poisson}(\lambda_j)$, and

$$X_1 = N_0 + N_1, \qquad X_2 = N_0 + N_2, \qquad X_3 = N_0 + N_3.$$

In this case, the common component N_0 gives a simple benchmark for all mixed cumulants involving X_1, X_2, and X_3: for any $w_1, w_2, w_3 \geq 1$,

$$\kappa_{\underbrace{1,\ldots,1}_{w_1},\underbrace{2,\ldots,2}_{w_2},\underbrace{3,\ldots,3}_{w_3}} = \lambda_0.$$

$[w_1, w_2, w_3]$	d	Case 1	Case 2	Case 3
$[1,1,1]$	3	$\surd$	$\surd$	$\surd$
$[1,1,2]$	4	$\surd$	$\surd$	$\surd$
$[1,2,2]$	5	$\surd$	$\surd$	$\surd$
$[2,2,2]$	6	$\surd$	$\surd$	$\surd$
$[2,2,3]$	7	$\surd$	$\surd$	$\surd$
$[2,3,3]$	8	$\surd$	$\surd$	$\surd$
$[3,3,3]$	9	$\surd$	$\surd$	$\surd$

References

Saulis, L. and Statulevicius, V. A. *Limit Theorems for Large Deviations*. Kluwer Academic Publishers, 1991.

Speed, T. P. Cumulants and partition lattices. *Australian Journal of Statistics*, 25(2):378–388, 1983.