LinApart: optimizing the univariate partial fraction decomposition

Levente Fekésházy, Bakar Chargeishvili, Sam Van Thurenhout, Gábor Somogyi

The structure of the presentation

- Motivation
- Mathematical background
- Implementations
  - Mathematica
  - C
- Conclusions and outlook
  - Possible generalizations
  - Summary

Motivation

- Univariate partial fraction decomposition is a standard tool in any calculation, especially during integration.
- During our research, in order to compute phase space integrals relevant to a NNLO subtraction scheme we had to sequentially integrate over different variables. Before every integration we had to partial fraction our expressions to bring the terms to a form, where we can express the integrals as a multiple polylogarithm.
- Unfortunately, the complexity of the terms and of the whole expression reached a point, where the standard publicly available tools could not provide results or at least not in the desired time frame.

Motivation

$$
\begin{aligned}
f(x_a, x_b; y) = \Big[& (-4+y)(1-y+x_b y)(2-y+x_b y)(4-y+x_b y)(1-x_a-y+x_b y)^3 \\
& \times (-1+x_a-y+x_b y)(-4-4x_b-y+x_b y)(-4x_b-y+x_b y) \\
& \times (-4x_a-4x_b-y+x_b y)(4x_a-4x_b-y+x_b y)(2+2x_b-y+x_b y)^3 \\
& \times (6+2x_b-y+x_b y)(2-4x_a+2x_b-y+x_b y)(2+4x_a+2x_b-y+x_b y) \\
& \times (-1+x_a-x_a y+x_a x_b y)(1+x_a-x_a y+x_a x_b y)(-2+2x_a-x_a y+x_a x_b y) \\
& \times (2+2x_a-x_a y+x_a x_b y)(-x_b+x_a x_b-x_a y+x_a x_b y)^3 \\
& \times (-4+2x_a+2x_a x_b-x_a y+x_a x_b y)(4+2x_a+2x_a x_b-x_a y+x_a x_b y) \\
& \times (1-2x_a+x_a^2-y-x_a y+x_b y+x_a x_b y) \\
& \times \left(2x_b-2x_a x_b+x_a y-x_b y-x_a x_b y+x_b^2 y\right)^3 \Big]^{-1}
\end{aligned}
$$

Mathematical background

- Let us examine a rational function $f(x) = \frac{x^l}{Q(x)}$, where $Q = \prod_{i=1}^{n}(x - a_i)^{m_i}$ and $l < \deg Q$.
- The partial fraction decomposition means we have to separate the poles of said function.

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{c_{ij}}{(x - a_i)^j}$$

- The task is the calculation of the c_{ij} coefficients.

Mathematical background

- A high-schooler would do it by solving a system of equation, in some computer algebra book the Chinese reminder theorem or a fast reconstructive algorithms is advised.
- But this series is nothing else but a Laurent-expansion! Thus we can use the residuum theorem!

$$c_{ij} = \mathrm{Res}(g_{ij}, a_i) = \frac{1}{(m_i - j)!} \lim_{x \to a_i} \frac{d^{m_i - j}}{dx^{m_i - j}} \Big((x - a_i)^{m_i} f(x) \Big)$$

Mathematical background

- Since we remove the pole at a_i we can easily take the limit and getting:

$$c_{ij} = \frac{1}{(m_i - j)!} \frac{d^{m_i - j}}{da_i^{m_i - j}} a_i^l \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{1}{(a_i - a_k)^{m_k}}.$$

- Where the derivates can be easily computed using the generalized chain rule and derivative rule of polynomials.

$$f(x) = \sum_{i=1}^{n} \sum_{j_{-1}+j_0+j_1+\cdots+\hat{j_i}+\cdots+j_n = m_i - 1} \binom{l}{j_{-1}} \frac{a_i^{l-j_{-1}}}{(x - a_i)^{j_0+1}} \prod_{\substack{k=1 \\ k \neq i}}^{n} \binom{m_k + j_k - 1}{j_k} \frac{(-1)^{j_k}}{(a_i - a_k)^{m_k + j_k}}$$

Mathematical background

- What is we have an improper fraction, meaning $l \geq \deg Q$? In this case we can factor the fraction into a proper fraction and a monomial.

$$f(x) = x^{l-(\deg Q-1)} \frac{x^{\deg Q-1}}{Q(x)}$$

- Do the partial fractoin decomposition on the proper fraction term, then drag the monomial inside the sums. Where we are left with terms of the form $g(x) = \frac{x^p}{(x-a)^q}$, which can be conveniently written as a sum.

$$g(x) = \sum_{i=0}^{p-q} \binom{p-1-i}{q-1} a^{p-q-i} x^i + \sum_{i=p-q+1}^{p} \binom{p}{i} a^i (x-a)^{p-q-i}$$

Implementations

-Mathematica

- Extensive testing has revealed that leveraging the efficiency of the built-in differentiation routine, outperforms alternative methods. Thus in the Mathematica implementation we compute the residuum with

$$c_{ij} = \frac{1}{(m_i - j)!} \lim_{x \to a_i} \frac{d^{m_i - j}}{dx^{m_i - j}} \Big((x - a_i)^{m_i} f(x) \Big).$$

- Our implementation works put of the box and can take every expression that Apart can. Furthermore, we also included pre-processing options, which can make the decomposition of large expression faster.

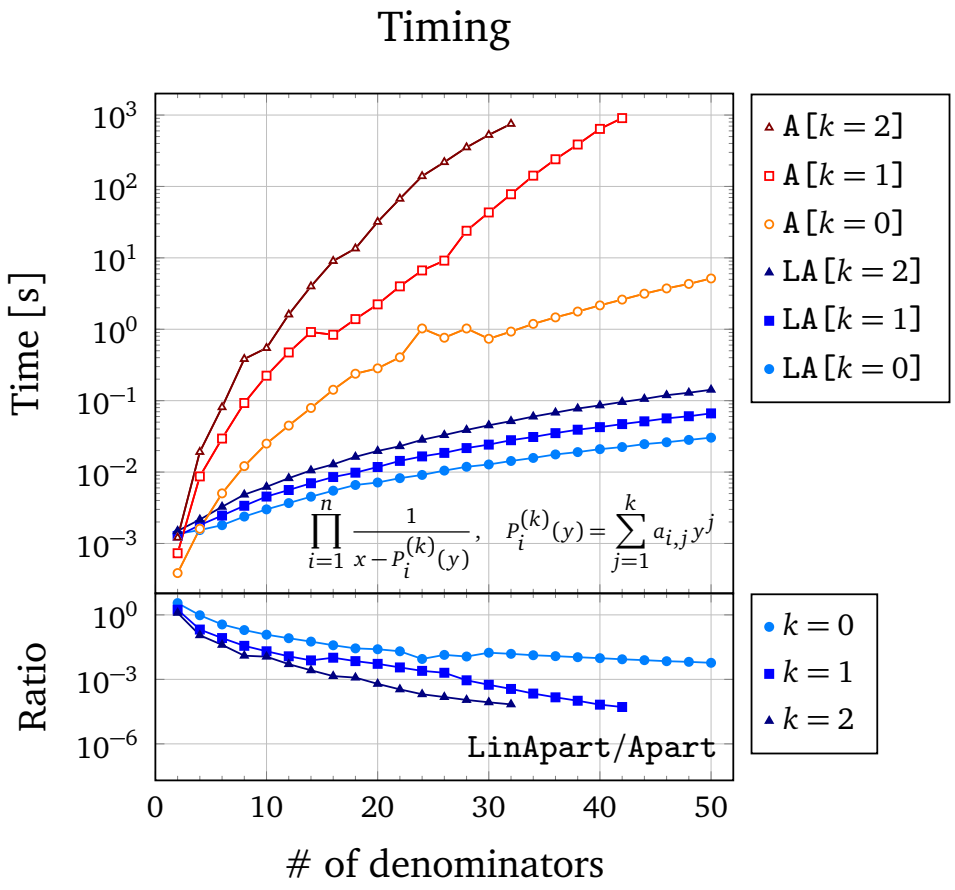
Implementations

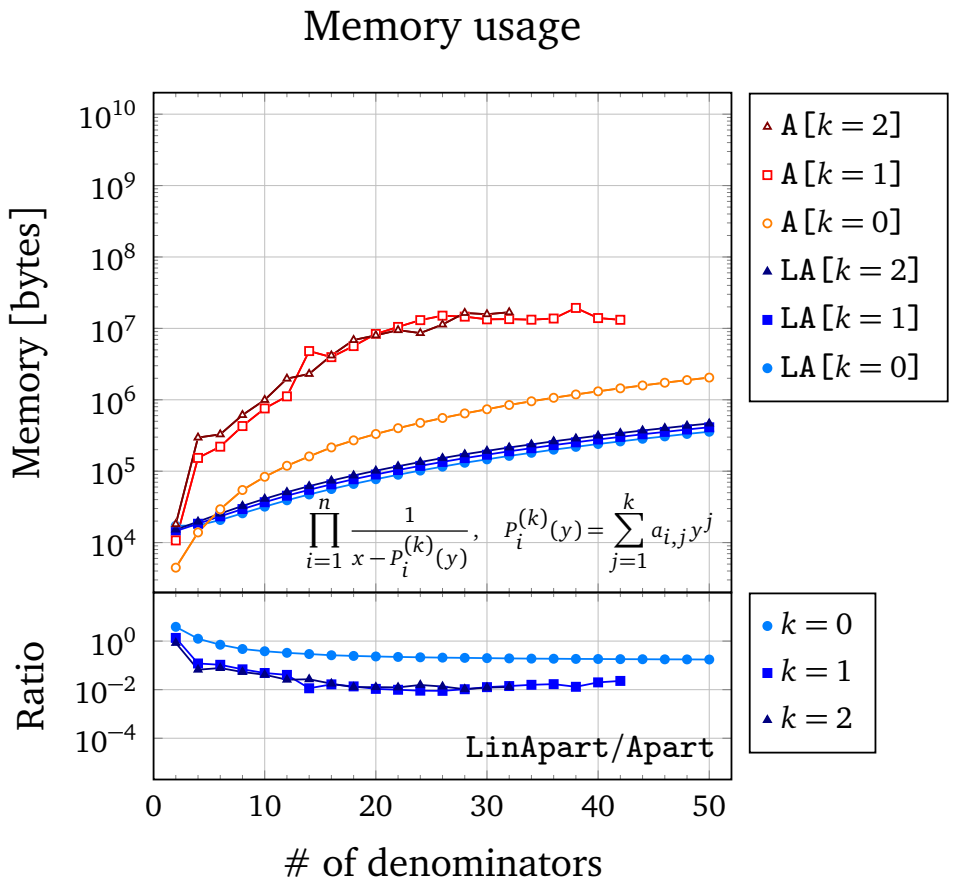
-Mathematica

- The question is how much do we gain compared to other algorithms, like Apart or Maple's parfrac.
- To tackle this question, first we must define the complexity of a fraction. After careful consideration we choose the following attributes:
  1. the number of distinct denominators
  2. The complexity of each individual denominator. In fact, even considering only linear denominators of the form $x - a_i$, the roots a_i may be functions of further variables and symbolic constants.
  3. The multiplicity of the denominator factors.
  4. The polynomial order of the numerator.

Implementations

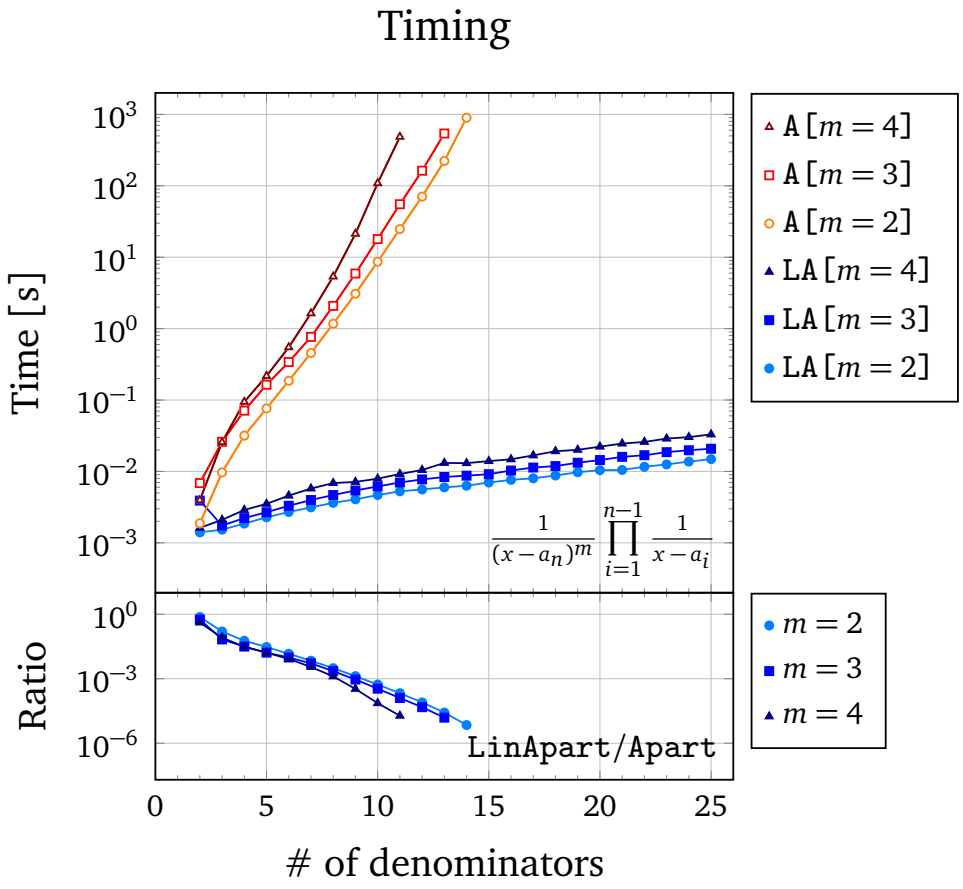
-Mathematica

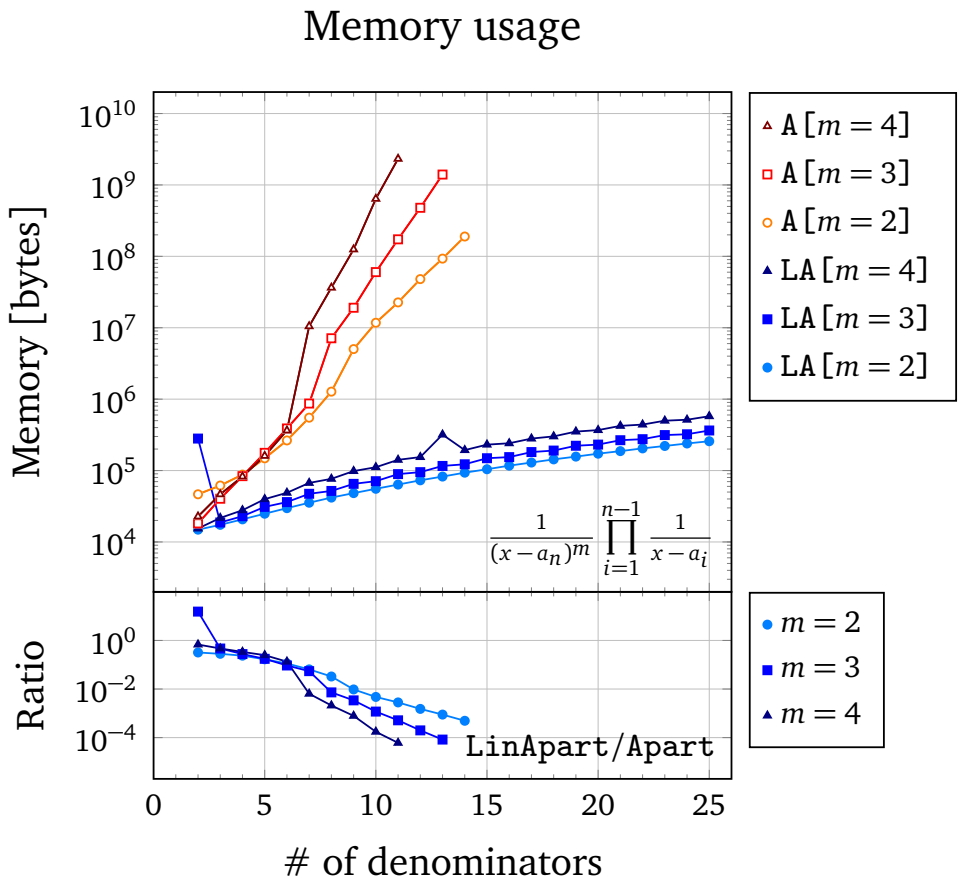




Implementations

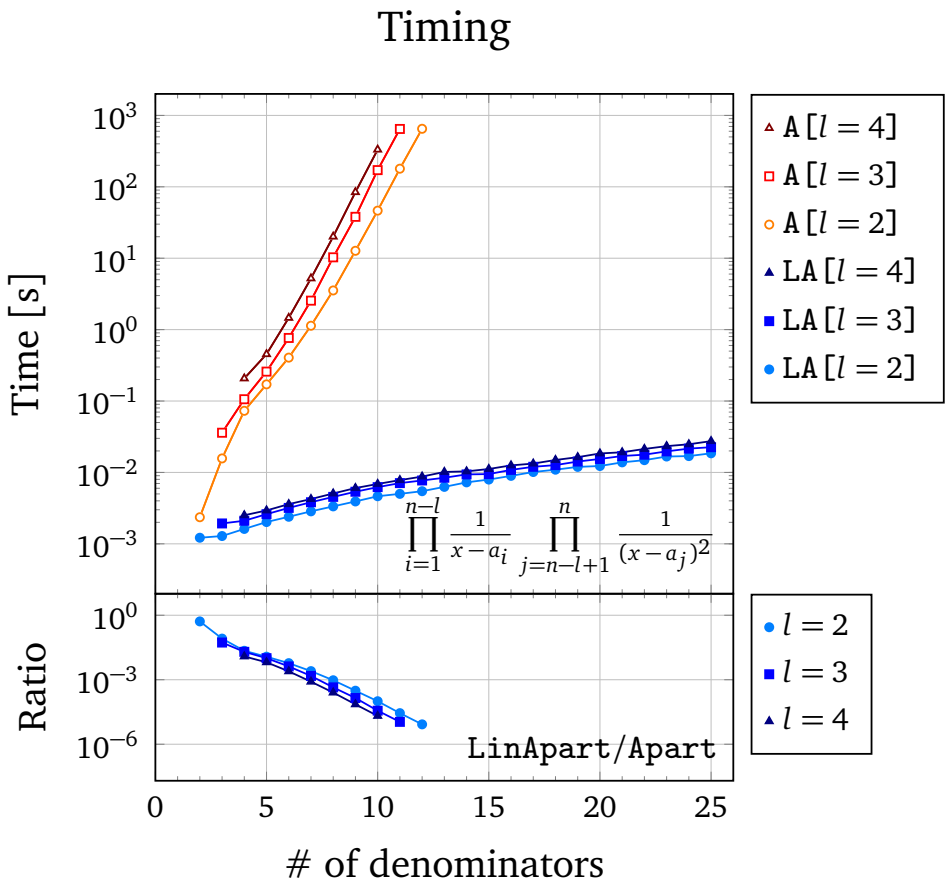
-Mathematica

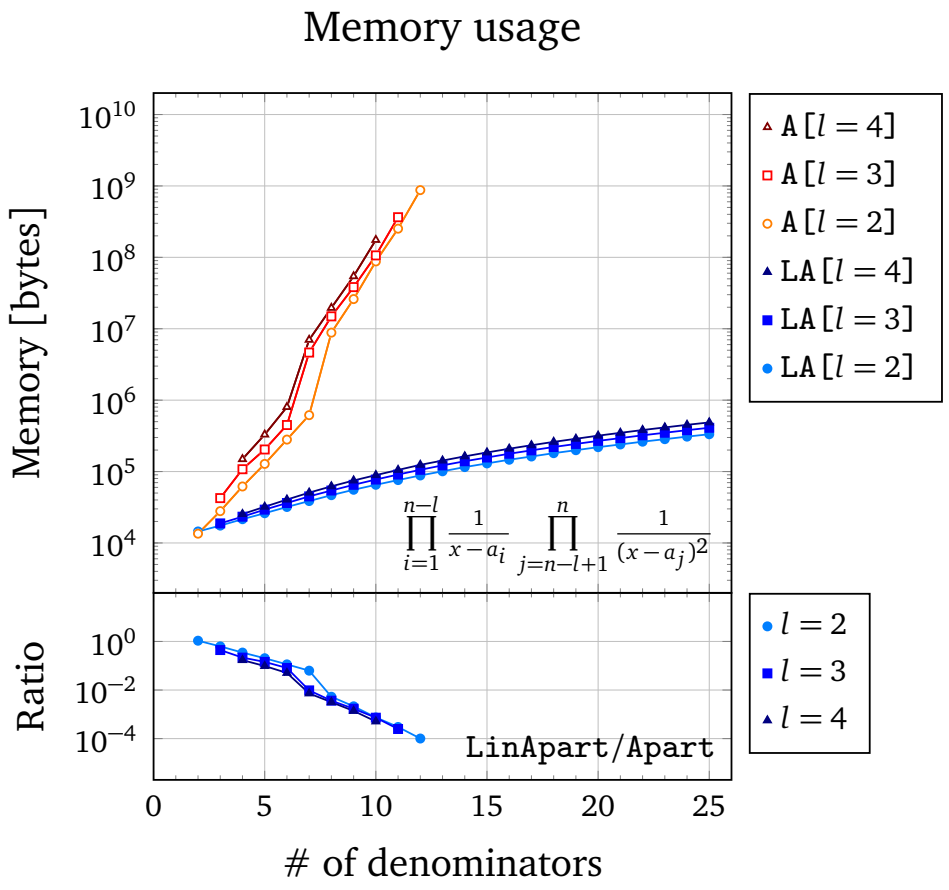




Implementations

-Mathematica





Implementations

-Mathematica

S

$$
\begin{aligned}
f(x_a, x_b; y) = \Big[& (-4+y)(1-y+x_b y)(2-y+x_b y)(4-y+x_b y)(1-x_a-y+x_b y)^3 \\
& \times (-1+x_a-y+x_b y)(-4-4x_b-y+x_b y)(-4x_b-y+x_b y) \\
& \times (-4x_a-4x_b-y+x_b y)(4x_a-4x_b-y+x_b y)(2+2x_b-y+x_b y)^3 \\
& \times (6+2x_b-y+x_b y)(2-4x_a+2x_b-y+x_b y)(2+4x_a+2x_b-y+x_b y) \\
& \times (-1+x_a-x_a y+x_a x_b y)(1+x_a-x_a y+x_a x_b y)(-2+2x_a-x_a y+x_a x_b y) \\
& \times (2+2x_a-x_a y+x_a x_b y)(-x_b+x_a x_b-x_a y+x_a x_b y)^3 \\
& \times (-4+2x_a+2x_a x_b-x_a y+x_a x_b y)(4+2x_a+2x_a x_b-x_a y+x_a x_b y) \\
& \times (1-2x_a+x_a^2-y-x_a y+x_b y+x_a x_b y) \\
& \times \left(2x_b-2x_a x_b+x_a y-x_b y-x_a x_b y+x_b^2 y\right)^3 \Big]^{-1}
\end{aligned}
$$

Implementations
-C

- Not every language has a built-in fast symbolic derivative function. In these cases we must use

$$\frac{x^l}{Q(x)} = \sum_{i=1}^{n} \sum_{j_{-1}+j_0+j_1+\cdots+\hat{j_i}+\cdots+j_n=m_i-1} \binom{l}{j_{-1}} \frac{a_i^{l-j_{-1}}}{(x-a_i)^{j_0+1}} \prod_{\substack{k=1 \\ k\neq i}}^{n} \binom{m_k+j_k-1}{j_k} \frac{(-1)^{j_k}}{(a_i-a_k)^{m_k+j_k}}$$

  paired with

$$\frac{x^p}{(x-a)^q} = \sum_{i=0}^{p-q} \binom{p-1-i}{q-1} a^{p-q-i} x^i + \sum_{i=p-q+1}^{p} \binom{p}{i} a^i (x-a)^{p-q-i}$$

Implementations
-C

- This implementation is ought be used as the last step of a partial fraction decomposition routine. It need the multiplicities, the power of the numerator and the roots as input.
- It can be connected to any symbolic computer algebra program (Wolfram Mathematica, FORM, SymPy etc.).
- We connected it to Mathematica with LibraryLink, but the native Mathematica implementation outperformed it.

Conclusion and outlook
-Possible generalizations

- This univariate partial fraction decomposition with linear denominators is a very specific problem. One can immediately think of two possible generalization:
  1. higher order polynomials in the denominator,
  2. application to the multivariable case.
- The former can be tackled with some help of higher functions like Factor in Mathematica and the Vieta's formulas.
- For the latter there are already existing algorithms, which do exactly the same (Leinartas's algorithm, Gröbner basis method).
- Of course one can use univariate partial fractioning iteratively, but then the well-known spurious singularities are going to arise.

Conclusion and outlook
-Summary

- The univariate partial fraction decomposition is vital to QFT calculations. Unfortunately, increasing the order of approximation the complexity of the expression increase rapidly. In some cases the publicly available native function of popular symbolic algebra programs are insufficient.
- We presented new implementations of the Laurent-expansion method, with which we were able to decrease the time and memory usage of this operation to a level, where even complicated functions' decomposition take mere second and consume just Mbs.
- We also presented a closed formula, which only involves sums and product, thus is programmable in any language of choice.

Conclusion and outlook
-Summary

- The source code and example can be obtained at `https://github.com/fekeshazy/LinApart`, while the documentation is published on arXiv (`https://arxiv.org/abs/2405.20130`).

Thank you for your attention!