Breaking the 2-loop barrier for generalized IBP reduction algorithms

A.A.Radionov and F.V.Tkachov*

INR RAS, Moscow
*speaker

Talk at the Bogolyubov-2019 Conf.
JINR, Dubna
2019-09-12

The ideology of the project is exactly opposite to that of the "analytical" bubble represented by the preceding talk.

The variety of analytical tricks defies automation -- too many *specialists* are needed.

For efficient automation something different is needed.

---

classical IBP -- Tkachov, 1981 ("p-way")

generalized IBP -- Tkachov, 1996
based on existence theorem proved by Bernshtein, 1972

very tempting -- but extremely hard

Message:
we are currently playing with true 2-loop BT operators
(examples at end)

since it is so hard:

Rule of vertical transcendence of interdisciplinary boundaries

For best results, the said boundaries should be transparent.

application level ("physics")
  understanding the problem
  formulation
math
  theor math
  implementtion math
programming
  architecture
  coding

BT method is hard -- efficiency is key,
no BS is tolerable at any level.

BS at various levels

application level

"only analytical answers survive in eternity" (E.Remiddi)
BUT significant digits survive even better

"analytical answer"
BUT actually needed is a piece of code/parceable data

formulation

point of reference:
for specific m's, k's, compute the amplitude
anything on top of that is a bonus

math

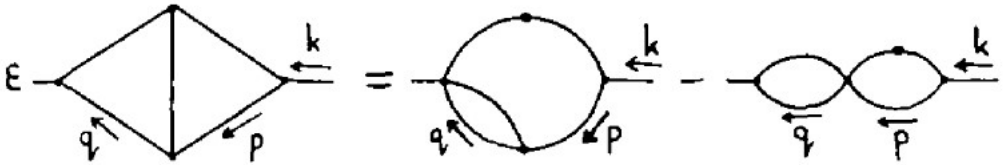
wide-spread belief in magic of fancy stuff
numerology (cf. the talk by Chetyrkin)

programming

the BS of "industrial strength" tools (cf. the BS of today's plenary talks)
a belief that a CAS would do things for you

What is IBP

classical IBP -- Tkachov, 1981
a standard tool for large-scale loop calculations



NB in terms of Feynman parameters,
the left integral has 5 of them restricted by sum x_i = 1
-- a standard d=4 symplex.

the integrals on the right have 4 of them, and correspond to setting one of the x's to zero -- boundaries of the d=4 symplex.

This pattern will emerge in generalized IBP.

What is generalized IBP (BT; Tkachov, 1996)

1940-s: L.Schwarts : a convenient formalism for generalized functions

early 1950-s: I.M.Gelfand & co. played with g.f.s; analytically continued x^mu appeared in Gelfand-Shilov, v.1

1954: Gelfand formulates hypothesis:
V(x,y,...)^mu exists for any polynomial P.

1972: Bernshtein proves a remarkable theorem: for any polynomial P(x,y,...) there exists a differential operator P(x,y,...), whose coefficients are polynomails of x,y,... such that

$$b^{-1}\overline{\mathcal{P}}(x,\partial)V^{\mu+1}(x) = V^{\mu}(x)$$

b(mu) is the Bernshtein-Sato function, much studied by mathematicians

Idea: use this equation and integration by parts to increase the power.

BASIC PROBLEM

Construct BT-operators **for a pair** of polynomials (pQFT, D#4).

BASIC METHOD

brute force --> a huge system of linear equations

Special case

Explicit solution for one loop --Tkachov, 1996.

$$\frac{1}{\Delta}\left(1-\frac{(x+A)\partial_x}{2(\mu+1)}\right)V^{\mu+1}(x)=V^{\mu}(x)\,,$$

where $\Delta=\left(Z-R^{\mathrm{T}}\tilde{V}^{-1}R\right)$ and $A=R^{\mathrm{T}}\tilde{V}^{-1}$.

Much used by Passarino, Uccirati et al. 2000s.

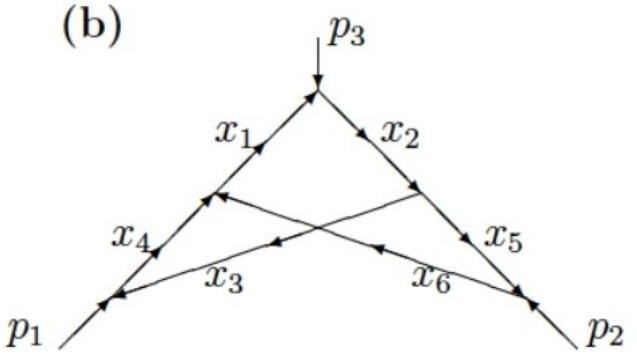
A two-loop example by GRACE, 2012: a single polynomial. An algorithm due to Oaku, uses Groebner basis, runs on a Japanese CAS.

$$J = \int_0^1 dx_1 \cdots dx_6 \delta(1 - \sum x) \frac{1}{(\mathcal{D} + i\epsilon)^2}$$

$$\mathcal{D} = \sum_S W_S p_S^2 - U \sum_j x_j m_j^2$$

$$\sum W_S p_S^2 = f_1 s_1 + f_2 s_2 + f_3 s_3 \ .$$

$$U = x_{12}x_{3456} + x_{34}x_{56}$$



Expand [(−8 457 445 293 636 043 504 800 000 * s * Dx5 + 8 457 445 293 636 043 504 800 000 * x1 * Dx25 +
раскрыть скобки

8 457 445 293 636 043 504 800 000 * x2 * Dx25 + 8 457 445 293 636 043 504 800 000 * x3 * Dx35 +
8 457 445 293 636 043 504 800 000 * x4 * Dx35 − 160 691 460 579 084 826 591 200 000 * x1 * Dx55 −
67 659 562 349 088 348 038 400 000 * x2 * Dx55 − 50 744 671 761 816 261 028 800 000 * x3 * Dx55 −
126 861 679 404 540 652 572 000 000 * x4 * Dx55 −
8 457 445 293 636 043 504 800 000 * x5 * Dx55 + 140 957 421 560 600 725 080 000 * Dx11 −
281 914 843 121 201 450 160 000 * Dx12 + 140 957 421 560 600 725 080 000 * Dx22 +
140 957 421 560 600 725 080 000 * Dx13 − 140 957 421 560 600 725 080 000 * Dx23 +
140 957 421 560 600 725 080 000 * Dx33 − 140 957 421 560 600 725 080 000 * Dx14 +
140 957 421 560 600 725 080 000 * Dx24 − 281 914 843 121 201 450 160 000 * Dx34 +
140 957 421 560 600 725 080 000 * Dx44 − 1 832 446 480 287 809 426 040 000 * Dx15 −
986 701 950 924 205 075 560 000 * Dx25 − 1 832 446 480 287 809 426 040 000 * Dx35 −
986 701 950 924 205 075 560 000 * Dx45 + 84 997 325 201 042 237 223 240 000 * Dx55 −
16 914 890 587 272 087 009 600 000 * Dx5) / (−2 819 148 431 212 014 501 600 000)]

4]= $$-\frac{Dx11}{20}+\frac{Dx12}{10}-\frac{Dx13}{20}+\frac{Dx14}{20}+\frac{13\,Dx15}{20}-\frac{Dx22}{20}+\frac{Dx23}{20}-\frac{Dx24}{20}+\frac{7\,Dx25}{20}-\frac{Dx33}{20}+$$
$$\frac{Dx34}{10}+\frac{13\,Dx35}{20}-\frac{Dx44}{20}-\frac{7\,Dx45}{20}+6\,Dx5-\frac{603\,Dx55}{20}+3\,Dx5\,s-3\,Dx25\,x1+57\,Dx55\,x1-$$
$$3\,Dx25\,x2+24\,Dx55\,x2-3\,Dx35\,x3+18\,Dx55\,x3-3\,Dx35\,x4+45\,Dx55\,x4+3\,Dx55\,x5$$

The general 2-loop case has remained a challenge.

Coding level: Oberon

Non-mainstream. **********
For details, see Fyodor Tkachov at AIHENP, Beijing, 2013

Oberon = Pascal-88 or Ultra Pascal
dialect: **Component Pascal**
Niklaus Wirth (Turing Prize, 1984) and his team at ETH Zurich

small, simple language. no fancy features,
but **full protection for programmer**

Oberon IS NOT just another language
-- as decimal system IS NOT just another notation for numbers among the zillion of non-positional systems

For system-level programming, **better than C**.
For large-scale frameworks, **better than anything**.
One never has to worry about segviols

FT: Oberon, a "silver bullet" https://youtu.be/HvAipsXmJpk

GopherCon 2015: Robert Griesemer - The Evolution of Go
https://youtu.be/0ReKdcpNyQg

How he could not forget Oberon for 15 years in the industry, and initiated Golang at Google.

Java, C#, Go -- all under strong influence of Oberon

Project Informatika-21 www.inr.ac.ru/~info21/
POCATOM --cf. talk by Ilkaev
drones
algotrading
.....

Computer algebra level: Gulo (BEAR 3.0)

principles: Tkachov 1990 (at a JINR CAS workshop)

-- user must have a full control over data representation, over arithmetic, sorts, etc.

-- emphasis on homogenized data and the corresponding algorithms, priority (but not limited) to sequential processing

-- separation of "algebra" and data representation

A small engine BEAR built with Oberon, used in 2001-2005 in a number of projects (Czarnecki et al.: "daunting challenge": 4-loops, B-decays)

comparisons with a C++ system:
runs 8 times faster, written in a few months instead of some years.

Current version: Gulo (BEAR 3.0)
both interfaces and low-level alrotihms much impoved.
Handles arbitrarily large data.
Some tests: as fast on SSD as in RAM (cooperates with OS file cache).

Zipper2

v.1 was used in 2001-2005 by Czarnecki et al.
A solver for huge systems of homogeneous linear equations.

Starting design point: Gauss elimination.
+ optimizations for finding BT operators:
based on statistics
based on study of Groebner basis algorithms etc.

Two phases, Forward and Backward, equally cumbersome (a real bad intermediate blow-up)

#1 **R-trick** = a fast algorithm to determine existence of a non-trivial solution and to quickly generate just one solution.
2nd phase (backward pass) practically eliminated -- **a radical speedup**.

#2 A fast arithmetic is important, we are currently experimenting with various options. (Such experimentation is impossible with C++ etc.)

#3 We have so far just scratched the surface of **storage optimizations** -- the design of Gulo + Zipper2 provides many options.

Partial BT-operators

The two polynomials enter the integral with different powers.
Partial BT operators: only one power is lowered.

$$D\left(x,\mu,\partial\right)P_0^{\mu_0}P_1^{\mu_1}=b\left(\mu\right)P_0^{\mu_0-1}P_1^{\mu_1-1}$$

$$D_1\left(x,\mu,\partial\right)P_0^{\mu_0}P_1^{\mu_1}=b_1\left(\mu\right)P_0^{\mu_0}P_1^{\mu_1-1}$$

$$D_0\left(x,\mu,\partial\right)P_0^{\mu_0}P_1^{\mu_1}=b_0\left(\mu\right)P_0^{\mu_0-1}P_1^{\mu_1}$$

The general operator is composed from two partial ones

$$D\left(x,\mu,\partial\right)P_0^{\mu_0}P_1^{\mu_1}=D_1\left(x,\mu-e_0,\partial\right)D_0\left(x,\mu,\partial\right)P_0^{\mu_0}P_1^{\mu_1}$$

Surprise: partial operators are much simpler -- and **much easier to find** -- than the general one.
This is not obvious from the corresponding systems of equations.

Summary of optimizations

1) perfect software (Oberon, Gulo, Zipper2), designed with utmost care in regard of (1) algorithmic and (2) storage optimizations

2) significant optimizations based on the observed statistics

3) R-trick eliminates the second phase

4) computation is restricted to the much simpler partial BT-operators

Net effect

Simplest 2-loop sunset diagram:

2005: with the old BEAR and old Zipper, could not reach the end of calculation.

2019: ~1 sec on a low-end notebook

It all works fast enough that one can indulge in complex games, e.g. reconstruct an exact dependence on k^2

Unforeseen **new option**: a reuse of information about the polynomials for different values of kinematic parameters
-> a further significant speedup.

Likely not the last one.
The situation is far from being as bleak as some believed it to be.
>> See examples in a separate document.