Analytic multi-loop results using finite fields and dataflow graphs with FiniteFlow

Tiziano Peraro (University of Zürich)
RADCOR 2019
9 September 2019

Based on:
Experiments at LHC

- high-accuracy (% level)
- large SM background
- high c.o.m. energy $\Rightarrow$ multi-particle states

We need scattering amplitudes

- describe hard partonic interaction
- high accuracy $\Rightarrow$ loops (% level $\sim$ 2 loops)
- multi-particle $\Rightarrow$ high multiplicity

Theoretical studies of amplitudes

- structures of QFT/gauge theories
Two and higher loops

- **Algebraic** calculations for multi-loop amplitudes
  - preferred strategy @ $\ell \geq 2$ loops
    - faster/more stable evaluation
    - better suited for many multi-loop techniques
    - allows more tests, studies, etc... and better control
  - often characterized by **high complexity**

- **Complexity** can be a combination of
  - number of loops for high accuracy
  - number of legs for high multiplicity
  - numbers of scales (invariants, external/internal masses)
Loop amplitudes

- An integrand contribution to $\ell$-loop amplitude

$$A = \int_{-\infty}^{\infty} \left( \prod_{i=1}^{\ell} d^{d}k_{i} \right) \frac{\mathcal{N}}{D_{1}D_{2}D_{3}\cdots}$$

- rational function in the components of loop momenta $k_{j}$
- polynomial numerator $\mathcal{N}$
- quadratic denominators corresp. to loop propagators

$$D_{j} = l_{j}^{2} - m_{j}^{2}$$
Computing amplitudes

1. Write amplitudes as l.c. of Feynman integrals

\[ A = \sum_j a_j I_j \]

- \( a_j \) rational functions of invariants
- \( I_j \) integrals with a “nice” / “standard” form

2. reduce \( I_j \) to linearly independent Master Integrals (MIs)

\[ \{ G_1, G_2, \ldots \} \subset \{ I_j \} \]

- generate and solve Integration-By-Parts (IBP) identities


\[ I_j = \sum_k c_{jk} G_k \]

3. Compute the MIs
Finite fields and functional reconstruction

A major bottleneck

- Large intermediate expressions
- Intermediate stages much more complicated than final result

Functional reconstruction

- reconstruct analytic results from numerical evaluations
  - evaluation over finite fields $\mathbb{Z}_p$ (i.e. modulo prime integers $p$)
  - use machine-size integers, $p < 2^{64} \Rightarrow$ fast and exact
  - collect numerical evaluations and infer analytic result
- sidesteps large intermediate expressions & highly parallelizable
- first applications
  - IBPs and univ. reconstruction \textit{von Manteuffel, Schabinger (2014)}
  - helicity amplitudes and multivariate reconstruction \textit{T.P. (2016)}
Some notable examples

- **FINRed (private)** [von Manteuffel]
  - several results for 4-loop form factors [von Manteuffel, Schabinger]

- **FINITEFLOW [T.P.]**
  - Several two-loop five-point amplitudes
    [Badger, Brønnum-Hansen, Hartanto, T.P.;
    Badger, Chicherin, Gehrmann, Heinrich, Henn, T.P., Wasser, Zhang, Zoia]
  - Matter dependence of the four-loop cusp anomalous dimension
    [Henn, T.P., Stahlhofen, Wasser]

- Private code
  [Abreu, Dormans, Febres Cordero, Ita, Page, Sotnikov, Zeng]
  - analytic five-parton amplitudes

- **FIRE 6 [A.V. Smirnov, F.S. Chuharev]**
  - Four-loop quark form factor with quartic fundamental colour factor [Lee, Smirnov, Smirnov, Steinhauser]
The black-box interpolation problem

Given a rational function $f$ in the variables $z = (z_1, \ldots, z_n)$ over $\mathbb{Q}$

- Reconstruct analytic form of $f$, given a numerical procedure

\[(z, p) \rightarrow f \rightarrow f(z) \mod p.\]

- Evaluate $f$ numerically for several $z$ and $p$

- Efficient multivariate reconstruction algorithms exist
e.g. T.P. (2016,2019), Klappert, Lange (2019)

- Upgrade analytic $f$ over $\mathbb{Q}$ using rational reconstruction algorithm
[Wang (1981)] and Chinese remainder theorem
The black-box interpolation problem

Given a rational function $f$ in the variables $z = (z_1, \ldots, z_n)$ over $\mathbb{Q}$

- Reconstruct analytic form of $f$, given a numerical procedure

\[
(z, p) \rightarrow f \rightarrow f(z) \mod p.
\]

- evaluate $f$ numerically for several $z$ and $p$

- efficient multivariate reconstruction algorithms exist
e.g. T.P. (2016,2019), Klappert, Lange (2019)

- upgrade analytic $f$ over $\mathbb{Q}$ using rational reconstruction algorithm
  [Wang (1981)] and Chinese remainder theorem

Question in this talk
How to build the black box?
Example: Scattering amplitudes over finite fields

- External states (momenta and polarizations)
  - rational parametrization with momentum twistors variables
    Hodges (2009), Badger, Frellesvig, Zhang (2013), Badger (2016)
- Tree-level
  - diagrams or recursion relations (e.g. Berends-Giele)
- Loop integrands
  - Feynman diagrams and t'Hooft algebra
  - Unitarity cuts sewing tree-level currents
    - higher finite-dim. representation of internal states in dim. reg.
- Integrand reduction
  - linear fit to a “nice” integrand basis
How to build a code for fast numerical evaluations of finite fields? We can consider a few options:

1. Low-level coding (e.g. in C/C++/Fortran)?
   - ✔ very good runtime efficiency
   - ✗ harder to program
   - ✗ limits usability

2. Low-level coding + high-level interfaces?
   - • common algorithms in C++ (e.g. linear solvers, fits, etc.)
   - • high-level wrapper (e.g. for Mathematica/Python)
   - ✔ good efficiency and usability
   - ✗ not flexible
   - ✗ these algorithms are often intermediate steps
How to build the black box?

Observations:

- A typical multi-loop algorithm involves several steps
  - solving linear systems
  - substitutions / changes of variables
  - etc.

- Large simplifications often occur at the very last stages
  - it’s best to do everything numerically
  - only the final expression reconstructed analytically

- Many algorithms share common “building blocks”
FiniteFlow [T.P. (2019)] has three main components

1. “basic” algorithms in C++ over finite fields
   - dense/sparse linear solvers, linear fits, evaluating rat. functions, list manipulations, etc.

2. higher-level framework to combine them into complex ones
   - output of a basic algorithm is input of others
   - graphical representation of your calculation (dataflow graphs)

3. multivariate reconstruction algorithms

FiniteFlow

- build complex algorithms without any low-level programming (e.g. from Mathematica interface)
- many methods for amplitudes can be cast in this framework
FiniteFlow: using data flow graphs

- **FiniteFlow** uses (simplified) data flow graphs
  - **Nodes** represent numerical algorithms
  - **Arrows** represent lists of numerical values
- In my implementation, a node has
  - 0 or more lists (arrows) of input values
  - 1 list (arrow) of output values
Example of a graph
Example: Evaluation of rational functions

- input: a list of values $z = (z_1, \ldots, z_n)$
- output: a list of rational functions $\{f_1, f_2, \ldots\}$ at $z$

$$f_i(z) = \frac{p_i(z)}{q_i(z)} = \frac{\sum_{\alpha} n_{i,\alpha} z^{\alpha}}{\sum_{\beta} d_{i,\beta} z^{\beta}},$$

\[ z \xrightarrow{\text{rat. fun. eval.}} \{ f_1(z), f_2(z), \ldots \} \]
Example: Matrix multiplication

- Two lists as input
  1. entries of a matrix $A$
  2. entries of a matrix $B$
- use row-major order to store them as a list
- output: entries of matrix $C$ such that

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

A diagram illustrating matrix multiplication with inputs $A_{ij}$ and $B_{ij}$ and output $C_{ij}$. 
Example: Linear solver

- A $n \times m$ linear system with parametric rational entries

$$
\sum_{j=1}^{m} A_{ij} x_j = b_i, \quad (i = 1, \ldots, n), \quad A_{ij} = A_{ij}(z), \quad b_i = b_i(z)
$$

- input: list of values for parameters $z = (z_1, \ldots, z_n)$
- output: solution $c_{ij} = c_{ij}(z)$ such that

$$
x_i = \sum_{j \in \text{indep}} c_{ij} x_j + c_{i0} \quad (i \notin \text{indep})
$$

\[z \xrightarrow{\text{linear solver}} \{c_{ij}(z)\}\]
Learning algorithms

• Some algorithms have a **learning phase**
  • used to learn information for defining its output
  • must be completed before using them

• Example: **linear solver**
  • learn: its rank, dep. and indep. unknowns, indep. eq.s
  • learning phase: solve the system numerically a few times
  • optional: mark & sweep equations (sparse solver)
• Any graph $G_1$ can be used as a subgraph by an algorithm (a node) $A$ belonging to another graph $G_2$
  • $A$ will evaluate $G_1$ several times to compute its output
  • input of $G_1 = $ auxiliary variables chained with inputs of $A$

• Examples
  • Laurent expansion w.r.t. a variable
  • maps: evaluate $G_1$ for several inputs
  • partial reconstructions w.r.t. a subset of variables
  • (total or partial) fits w.r.t. and ansatz
  • etc...
• IBPs are \textbf{large} and \textbf{sparse} linear systems

• they reduce Feynman integrals $I_j$ to a lin. indep. set of MIs $G_j$

\[
I_i = \sum_j c_{i j} G_j
\]

• amplitudes and other multi-loop objects can be reduced mod IBPs

\[
A = \sum_j a_j I_j = \sum_{j k} a_j c_{j k} G_k = \sum_j A_j G_j, \quad \text{with} \quad A_j = \sum_k a_k c_{k j}
\]

• final results for $A_k$ often much simpler than $c_{i j}$

$\Rightarrow$ solve IBPs numerically and compute $A_j$ via a matrix multiplication

• Similar approach for \textbf{differential equations} for MIs
IBP reduction

- Input node: \( \{\epsilon, x\} \)
- Evaluate: \( \alpha_j \)
- Mat. mul: \( C_{jk} \)
- Output

Diagram:

- Input node: \( \{\epsilon, x\} \) to evaluate \( \alpha_j \)
- Evaluate \( \alpha_j \) to mat. mul \( C_{jk} \)
- Mat. mul \( C_{jk} \) to output
Coefficients of the $\epsilon$-expansion

If MIs are known analytically in terms of special functions $f_k$

$$G_j = \sum_k g_{jk}(\epsilon, x) f_k + \mathcal{O}(\epsilon),$$

$G_1 = \{\epsilon, x\} \xrightarrow{\text{MIs coeff.s}} A_j \xrightarrow{\text{mat. mul}} \xrightarrow{\text{evaluate}} g_{jk} \xrightarrow{\text{output}}$

$G_2 = \{x\} \xrightarrow{\text{Laurent}} G_1 \xrightarrow{\text{output}}$
Cutting-edge applications of FiniteFlow

- Five-point two-loop amplitudes
  - Several planar results for five partons and $W + 4$ partons
    [Badger, Brønnum-Hansen, Hartanto, T.P. (2017-2019)]
  - all-plus five gluon non-planar [Badger, Chicherin, Gehrmann, Heinrich, Henn, T.P., Wasser, Zhang, Zoia (2019)]
Cutting-edge applications of FiniteFlow

- Matter dependence of the 4-loop cusp anomalous dimension

[Henn, T.P., Stahlhofen, Wasser (2019)] ⇒ see M. Stahlhofen's talk
Public codes

- **FiniteFlow**

  https://github.com/peraro/finiteflow

  - C++ code
  - Mathematica interface (strongly recommended)

- **FiniteFlow MathTools**

  https://github.com/peraro/finiteflow-mathtools

  - packages FFUtils, LiteMomentum, LiteIBP, Symbols
  - examples (amplitudes, IBPs, diff. equations and many more)
Example of graphs in FiniteFlow

Piecing together the all-plus five gluon amplitude (only planar contributions are shown)
Summary

• Finite fields and functional reconstruction
  • enhance the possibilities of our theoretical predictions
  • new results unattainable with traditional computer algebra
  • public code \texttt{FINITEFLOW}

• Progress on 2-loop 5-point and other complex processes

Outlook

• More applications
  • massive processes, phase-space integrals, \ldots

• High level of automation for higher-loop predictions