Loop amplitude calculations on a computer

Do-It-Yourself guide

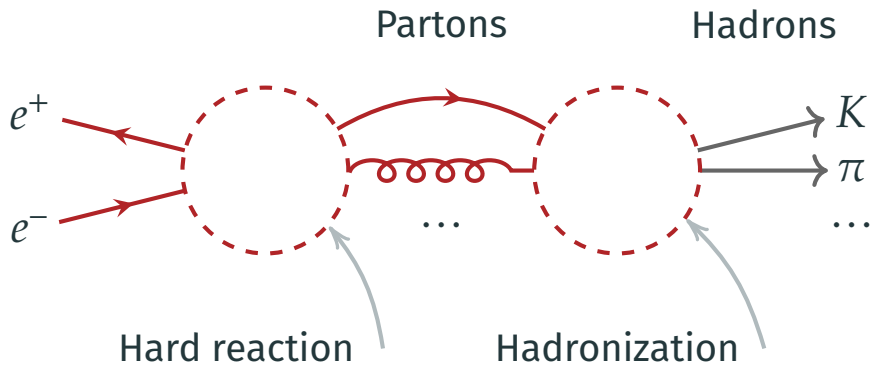
Vitaly Magerya

Institut für Theoretische Physik,
Karlsruher Institut für Technologie
(ITP KIT)

Hamburg, CAPP 2021

Goals

Goal of the lecture: take a *loop amplitude*, calculate it *on a computer*: analytically (and then numerically), from start to finish, with our own code.



Target quantity: the *total cross-section of e^+e^- annihilation to hadrons*,

$$\sigma(e^+e^- \to \text{hadrons}) = \sigma(e^+e^- \to \text{partons});$$

we shall calculate $\mathscr{O}(\alpha_s^2)$ corrections to it (and then $\mathscr{O}(\alpha_s^3)$ too).
Model: QCD with N_f *massless quarks*, and N_t *massive quarks* of mass m.

Motivation: the R ratio

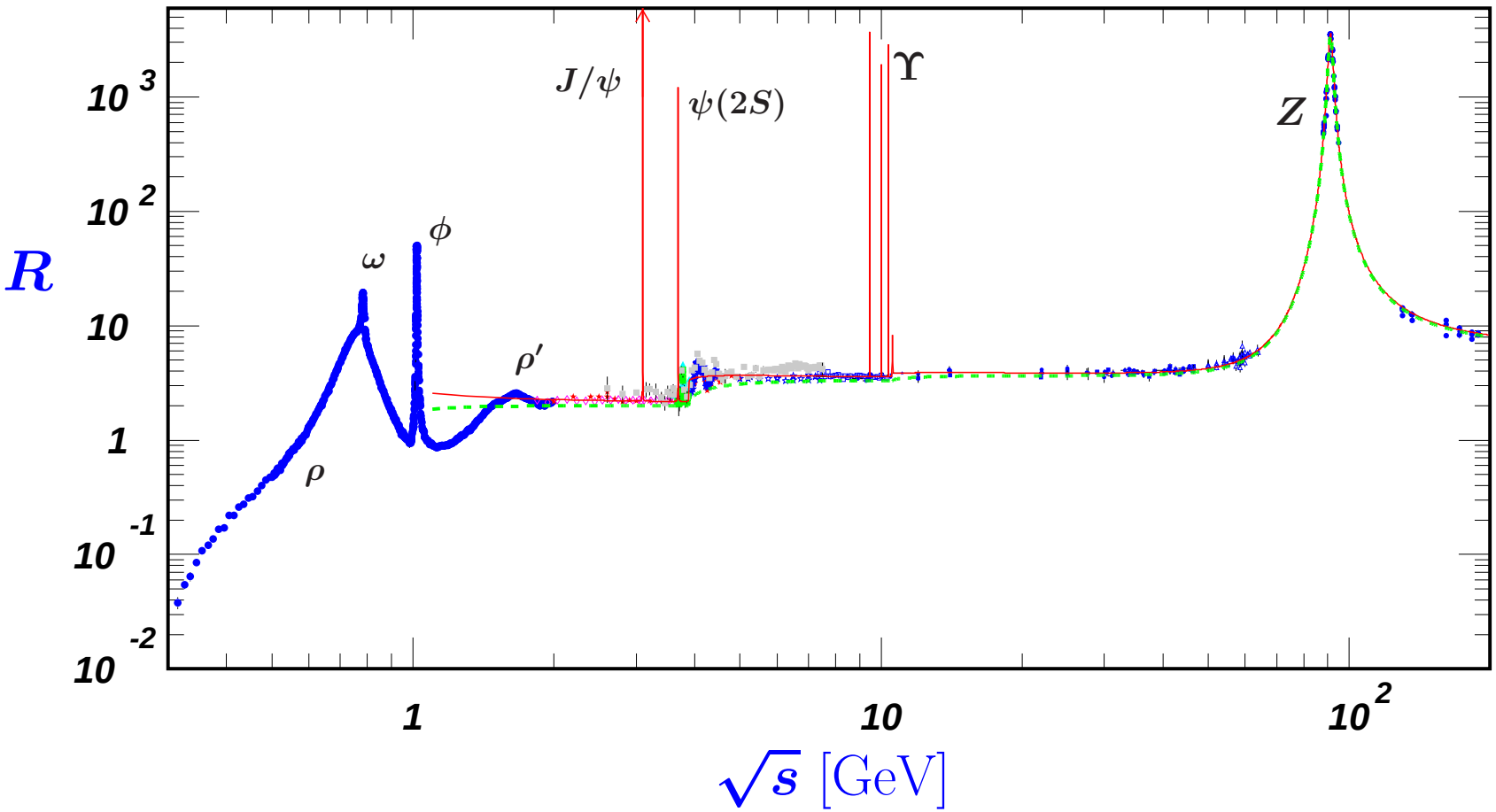
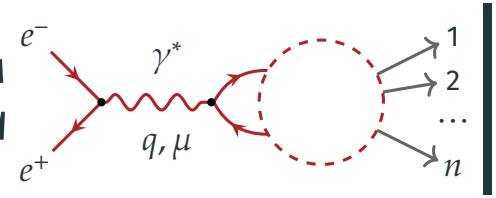


Figure: $R \equiv \sigma_{\text{tot}}(e^+e^- \to \text{hadrons})/\sigma_{\text{tot,leading}}(e^+e^- \to \gamma^* \to \text{muons})$, from Ezhela, Lugovsky, Zenin '03, available at pdg.lbl.gov/2020/hadronic-xsections.

Calculating e^+e^- annihilation to hadrons

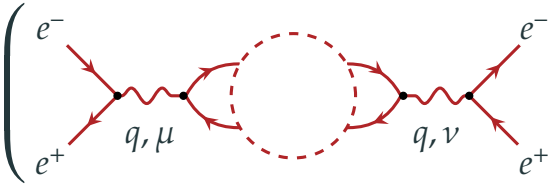
The square matrix element of the total cross-section is

$$|M(e^+e^- \to \text{partons})|^2 = \sum_n \int \mathrm{dPS}_n \left| \sum \right|^2 .$$



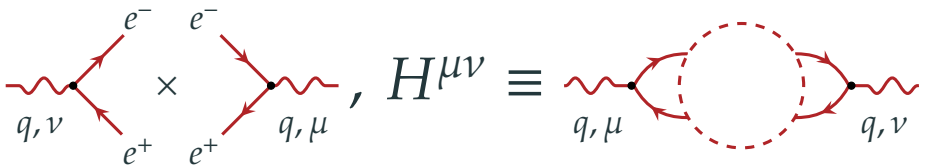
Often calculating it is easier via the optical theorem:

$$|M|^2 = -2\,\mathrm{Re}\left(\right).$$



Further, we can split this into the leptonic (L) and the hadronic (H) tensors:

$$|M|^2 = \frac{2}{q^4}\,\mathrm{Re}\left(L_{\mu\nu}H^{\mu\nu}\right),\; L_{\mu\nu} \equiv \quad \times \quad ,\; H^{\mu\nu} \equiv \quad .$$



All the loop integration and α_s corrections are in $H^{\mu\nu}$, and $L_{\mu\nu}$ is simple.

Tensor structures and projectors

Because $H^{\mu\nu}(q)$ is a Lorentz-covariant tensor, it can only have this general structure:

$$H^{\mu\nu}(q) = g^{\mu\nu}F_1(q) + q^\mu q^\nu F_2(q).$$

Because of Ward identities, $q_\mu H^{\mu\nu} = H^{\mu\nu}q_\nu = 0$, the structure is further restricted to

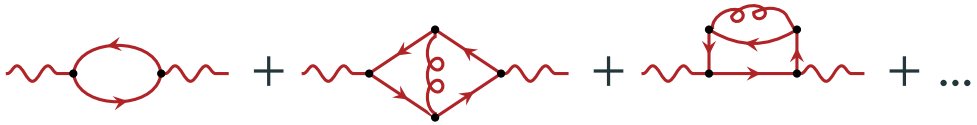
$$H^{\mu\nu}(q) = F_1(q)\left(g^{\mu\nu} - \frac{q^\mu q^\nu}{q^2}\right).$$

To invert this and get F_1 from $H^{\mu\nu}$ we must construct a projector:

$$F_1 = H^{\mu\nu}\frac{g_{\mu\nu}}{d-1}.$$

Calculting scalar values like F_1 is simpler than tensor values like $H^{\mu\nu}$, and a tensor decomposition like this along with projector constuction is almost always needed.

Calculation plan

1. Generate Feynman diagrams for the process.
   * $F_1 =$  $+ \ldots$
   * QGRAF with MATHEMATICA output.
2. Apply the Feynman rules.
   * $F_1 = \int \mathrm{d}^d l_1 \, \mathrm{Tr}\left(\gamma^\mu \not{k}_1 \gamma^\nu \not{k}_2\right) \cdots + \ldots$
   * Custom MATHEMATICA code.
3. Resolve Dirac and color tensor summation, convert to the scalar integral families.
   * $F_1 = N_f C_a \cdots I_{123} + \ldots$
   * MATHEMATICA $\longrightarrow$ FORM $\longrightarrow$ MATHEMATICA.
4. Use IBP relations to reduce to smaller set of "master integrals".
   * $F_1 = \left(N_f C_a \cdots + \ldots\right) I_{111} + \ldots$
   * MATHEMATICA $\longrightarrow$ KIRA $\longrightarrow$ MATHEMATICA.
5. Evaluate the master integrals.
   * Numerically: sector decomposition with PYSECDEC.
   * Analytically: differential equations with FUCHSIA.

General idea: use MATHEMATICA to glue everything together.

Diagram generation

Feynman diagrams with QGRAF

QGRAF is a widely used program for Feynman diagram generation available at `http://cfif.ist.utl.pt/~paulo/qgraf.html`.
To generate diagrams with QGRAF:

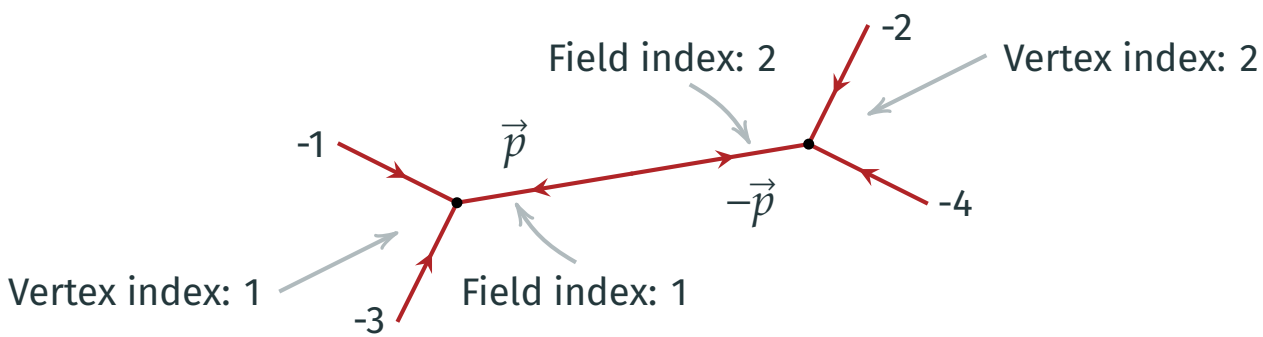
1. Create `qgraf.dat`, defining the process (incoming particles, outgoing particles, loop count, names of the momenta, vetoed diagrams), and listing the *model* and *style files*.
   See `qgraf.dat.example`.
2. Create a *model file* with a list of fields and vertices.
   See `qgraf-modfile` for the QCD model we'll use.
3. Create a *style file* (output template) defining the output format.
   See `qgraf-stylefile` for the Mathematica output style definition we'll use.

Also introducing: `qgraf.sh` and `mkdia.py` programs that will make our QGRAF usage simpler.
Use the MATHEMATICA notebook `show-diagrams.nb` to view the diagrams.

QGRAF diagram structure

Short summary:

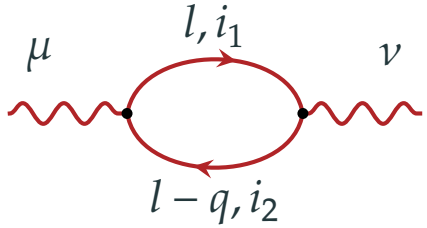
- In QGRAF diagrams consist of vertices and propagators (edges).
- Each vertex has a list of “fields” (legs).
- Vertex fields (legs) and their momenta are presented as incoming into the vertex.
- Both vertices and their legs are identified by their indices.
- Vertices and internal legs have positive indices.
- External legs have negative indices: -1, -3, -5, … for incoming particles, -2, -4, -6, … for outgoing.



Feynman rules

Notation on a computer

Feynman rules *in a book*:



$$= \int \frac{\mathrm{d}^d l}{(2\pi)^d} i g_e Q_{f_1} \delta_{i_1 i_2} \delta_{f_1 f_2} \operatorname{Tr}\left(\gamma^\mu \frac{\not{l} - \not{q}}{(q-l)^2 + i0} \gamma^\nu \frac{\not{l}}{l^2 + i0} \right) i g_e Q_{f_2} \delta_{i_2 i_1} \delta_{f_2 f_1}.$$

Feynman rules *on a computer*:

```
In[1] := One Feynman diagram, please???

Syntax::sntxf: "One Feynman diagram"
cannot be followed by ", please???".
```

MATHEMATICA notation, I

To operate on Feynman rules in MATHEMATICA, we need to choose a notation. Any will work; we'll use the following.

Index names:

* μ_i, Lorentz indices, $1 \ldots d$: `lor[i]`;
* i_i, fundamental color indices, $1 \ldots N_c$: `fun[i]`;
* a_i, adjoint color indices, $1 \ldots N_a = \frac{1}{2}\left(N_c^2 - 1\right)$: `adj[i]`;
* f_i, light quark flavors (up, down, etc), $1 \ldots N_f$: `flv[i]`;
* t_i, heavy quark flavors (top, maybe bottom), $1 \ldots N_t$: `flvt[i]`;
* s_i, Dirac (spinor) indices, $1 \ldots 4$: `spn[i]`.

Tensors:

* $f^{a_1 a_2 a_3}$, $SU(N_c)$ structure constants: `colorf[a1,a2,a3]`;
* $T^a_{i_1 i_2}$, $SU(N_c)$ generators, `colorT[a,i1,i2]`;
* $\left(\gamma^\mu\right)_{s_1 s_2}$, Dirac matrices: `gammachain[gamma[μ],s1,s2]`;
* $\left(\not{p}\right)_{s_1 s_2}$, Dirac slash notation: `gammachain[slash[p],s1,s2]`;

MATHEMATICA notation, II

- Propagator denominators:
  - Massless: $\frac{1}{p^2+i0} \to$ `den[p]`.
  - Massive: $\frac{1}{p^2-m^2+i0} \to$ `den[p,m2]`.
- Momenta components: $p^\mu \to$ `momentum[p,`μ`]`.
- Scalar products: $p \cdot q \to$ `sp[p,q]`.
- Delta functions:
  - Quark flavor: $\delta_{f_1 f_2} \to$ `deltaf[`f_1`,`f_2`]`.
  - Heavy quark flavor: $\delta_{t_1 t_2} \to$ `deltaft[`t_1`,`t_2`]`.
  - Generic: $\delta_{xy} \to$ `delta[`x`,`y`]`.
- Quark electric charges:
  - Light: $Q_{f_i} \to$ `chargeQ[`f_i`]`.
  - Heavy: $Q_{t_i} \to$ `chargeQt[`t_i`]`.

Everything else (products, sums, powers, etc): the normal MATHEMATICA expressions.

Feynman rules: propagators

* Quark propagator: $i\delta_{f_1 f_2}\delta_{i_1 i_2}\dfrac{\left(\not{p}\right)_{s_2 s_1}}{p^2}$.
* Heavy quark propagator: $i\delta_{t_1 t_2}\delta_{i_1 i_2}\dfrac{\left(\not{p}+\mathbb{1}m_t\right)_{s_2 s_1}}{p^2-m^2}$.
* Gluon propagator: $-i\delta_{a_1 a_2}\left(\dfrac{g_{\mu_1\mu_2}}{p^2}-(\xi-1)\dfrac{p^{\mu_1}p^{\mu_2}}{\left(p^2\right)^2}\right)$.
* Ghost propagator: $i\delta_{a_1 a_2}\dfrac{1}{p^2}$.
* Photon propagator: not needed, photons are external for $H^{\mu\nu}$.

See `Amplitude[]` in `allthecode.m` for the implementation.

Feynman rules: vertices

- Anti-quark/quark/gluon vertex: $ig_s\delta_{f_1f_2}T^{a_3}_{i_1i_2}\left(\gamma^{\mu_3}\right)_{s_1s_2}$.
- Anti-quark/quark/photon vertex: $ig_eQ_{f_1}\delta_{f_1f_2}\delta_{i_1i_2}\left(\gamma^{\mu_1}\right)_{s_1s_2}$.
- Anti-ghost/ghost/gluon vertex: $g_sf^{a_3a_2a_1}p_1^{\mu_3}$.
- Three-gluon vertex:

$$g_sf^{a_1a_2a_3}\left(g^{\mu_1\mu_2}\left(p_1-p_2\right)^{\mu_3}+(123\to231)+(123\to312)\right).$$

- Four-gluon vertex:

$$-ig_s\left(f^{\nu\mu_1\mu_2}f^{\nu\mu_3\mu_4}\left(g^{\mu_1\mu_3}g^{\mu_2\mu_4}-g^{\mu_1\mu_4}g^{\mu_2\mu_3}\right)+(1342)+(1423)\right).$$

  Note the fresh summation index ν.

Tensor summation

Dirac tensor summation with FORM

FORM can expand traces of gamma matrices:

$$\mathrm{Tr}\left(\gamma^\mu \not{p}\right) = \left(\gamma^\mu\right)_{s_1 s_2} \left(\not{p}\right)_{s_2 s_1}$$

```
= gammachain[gamma[lor[mu]],spn[1],spn[2]]*
  *gammachain[slash[p],spn[2],spn[1]]
→ gammachain[gamma[mu],slash[p],spn[1],spn[1]]
→ g_(i,mu,p)
  traceN i;
→ p(mu)
→ momentum[p,lor[mu]]
```

On the FORM side:

- i can be any arbitrary (but unique) integer;
- `mu` must be declared as an index: `index mu = d`;
- `p` must be declared as a vector: `vector p`;
- `p` must be a single variable, not an expression.

See: `example.dirac-trace.frm`.

Color tensor summation with COLOR.H

COLOR.H is a FORM package for $SU(N)$ tensor summation, available at
https://www.nikhef.nl/~form/maindir/packages/color/
Usage in summary:

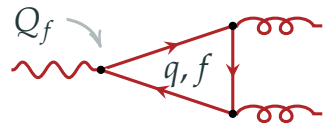
$$\begin{aligned}
\mathrm{Tr}\left(T^{a_1}T^{a_1}\right) &= T^{a_1}_{i_1 i_2} T^{a_1}_{i_2 i_1} \\
&= \texttt{colorT[ajd[1],fun[1],fun[1]]*} \\
&\quad \texttt{*colorT[ajd[2],fun[2],fun[1]]} \\
&\to \texttt{T(fun1,fun2,adj1)*T(fun2,fun1,adj1)} \\
&\quad \texttt{\#call docolor} \\
&\to \texttt{NA*I2R} \\
&\to N_a\, T_f
\end{aligned}$$

See: `example.color-trace.frm`.

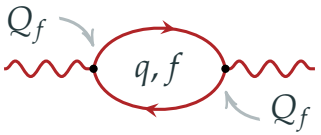
Quark flavor summation

Flavor-related factors are complicated by the dependence of charge on flavor (Q_f). Three cases are relevant:

$\sim N_f,$



$\sim \sum Q_f,$



$\sim \sum Q_f^2,$

and the same three cases for the heavy quarks (N_t, $\sum Q_t$, $\sum Q_t^2$).
No library for this; just some FORM code that:

1. Apply each $\delta_{f_1 f_2}$ factor by renaming f_1 into f_2 (if $f_1 \neq f_2$).
2. Recognize the possible remaining cases ($Q_f^2 \delta_{ff}$, $Q_f \delta_{ff}$, δ_{ff}).

See: `example.flavor-trace.frm`.

IBP relations

Integral family construction (for IBP)

An IBP integral family with L loop momenta l_i, and E external momenta p_i, is the set of integrals

$$I_{\nu_1,\nu_2,\ldots,\nu_N} \equiv \int \frac{\mathrm{d}^d l_1}{(2\pi)^d} \cdots \frac{\mathrm{d}^d l_L}{(2\pi)^d} \frac{1}{D_1^{\nu_1} \ldots D_N^{\nu_N}},$$

where $N = L\,(L+1)\,/2 + LE$, and the denominators,

$$D_i = \left(c_{ij}\, l_j + c'_{ik}\, p_k\right)^2 - m_i^2 + i0,$$

are linearly independent when viewed as polynomials in the scalar products

$$s \equiv \left\{l_i \cdot l_j,\ l_i \cdot p_j\right\}.$$

This ensures that each s_i can be uniquely expressed in terms of D_j:

$$D_i = M_{ij}\, s_j + C_i \quad \Rightarrow \quad s_i = \left(M^{-1}\right)_{ij} \left(D_j - C_j\right).$$

To express an integral in $I_{\nu\ldots}$ notation: replace all s by D, count D_i powers.

Integral symmetries

Compare these two integrals:

$$I_1 = \int \frac{\mathrm{d}^d l}{\left(p_1 - l\right)^2 \left(p_1 + p_2 - l\right)^2 l^2} \text{ and } I_2 = \int \frac{\mathrm{d}^d l'}{\left(p_1 + l'\right)^2 \left(p_2 - l'\right)^2 l'^2}.$$

To see that they are equal, use Feynman parameters:

$$I_i = \Gamma\left(3 - \frac{d}{2}\right) \int \mathrm{d}x_1 \mathrm{d}x_1 \mathrm{d}x_1 \, x_1 x_2 x_3 \, \delta(1 - x_1 - x_2 - x_3) \, \mathscr{U}_i^{3-d} \mathscr{F}_i^{d/2-3},$$

$$\mathscr{U}_1 = x_1 + x_2 + x_3, \; \mathscr{F}_1 = (x_1 + x_2) \, x_3 p_1^2 + (x_1 + x_3) \, x_2 p_2^2 + 2 x_2 x_3 p_1 \cdot p_2,$$

$$\mathscr{U}_2 = x_1 + x_2 + x_3, \; \mathscr{F}_2 = (x_2 + x_3) \, x_1 p_1^2 + (x_1 + x_3) \, x_2 p_2^2 + 2 x_1 x_2 p_1 \cdot p_2.$$

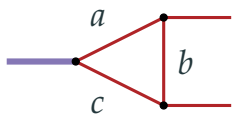
Both expressions become identical under $x_{1,2,3} \leftrightarrow x_{3,2,1}$. In momenta space this corresponds to $l' = l - p_1$. [Pak '11; FEYNSON]

Automated implementation: FEYNSON (github.com/magv/feynson).

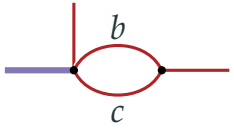
Example: `example.feynson.symmetrize.in`.

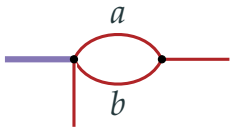
Scaleless (zero) integral detection

For efficiency, scaleless integrals should be put to zero early.
Consider the triangle family with one off-shell leg:

$$I_{a,b,c} \equiv$$ 

Two subsectors of this family are zero:

$$I_{0,b,c} \equiv$$ 

$$= 0, \qquad \text{and} \qquad I_{a,b,0} \equiv$$ 

$$= 0.$$

Sufficient criteria (Lee '13): a family (or a subsector) is zero if there are such x-independent k_i, that

$$\sum_i k_i \, x_i \frac{\partial}{\partial x_i} \left(\mathscr{F}(x) + \mathscr{U}(x) \right) = \mathscr{F}(x) + \mathscr{U}(x),$$

where $\mathscr{F}$, $\mathscr{U}$, and x give the corresponding Feynman parameterization.
Implementation: any IBP solver, also FEYNSON.
Example: `example.feynson.zero-sectors.in`.

IBP software

FIRE6 (bitbucket.org/feynmanIntegrals/fire) [Smirnov, Chukharev '19]

* *Fast and parallel* Laporta-style IBP reduction implementation.
* Has a (terrible) MATHEMATICA interface, but C++ core.
* Can use modular arithmetic to control intermediate expression swell, and for much greater parallelizibility (thousands of cores).
* Requires LITERED to produce symmetry tables.

KIRA (kira.hepforge.org): [Maierhöfer, Usovitsch, Uwer '18]

* *Fast and parallel* Laporta-style IBP reduction implementation.
* Optionally uses modular arithmetic methods. [Klappert, Lange, et al '20]
* Main drawback: excessive memory use (e.g. 200GB for a 4-loop problem).

IBP software, contd.

LiteRed (inp.nsk.su/~lee/programs/LiteRed) [Lee '13]

- Heuristic-driven IBP relation solution for general indices.
- Written in Mathematica, *easy to use*.
- Not parallelizable, and *too slow* for larger problems.
- Guarantees the minimal number of masters.
- Contains auxiliary functions for integral differentiation, Feynman parameterization, and dimentional recurrence construction.

Forcer (github.com/benruijl/forcer) [Ruijl, Ueda, Vermaseren '17]

- Hand-crafted reduction for massless 2-point functions up to 4 loops.
- Written in Form, parallelizable.
- The *fastest thing for 2-point functions*.

Many others: Reduze, Air, Fmft/Matad, private implementations, etc.

IBP reduction with Kira

Usage in short: [kira.hepforge.org]

- Define kinematics (`config/kinematics.yaml`).
- List integral families (`config/integralfamilies.yaml`).
- Create a jobs file (e.g. `jobs.yaml`), defining
  - for which integrals to *write down* IBP relations (r and s bounds);
  - for which integrals to *solve* IBP relations (r, s, and d bounds, or a list).
- Get the results as Mathematica (or Form) substitution tables.

Guide to the notation:

- r is the sum of denominator powers (positive indices);
- s is the sum of numerator powers (negative indices);
- d is the sum of dots (indices ≥ 2);
- t is the number of denominators.

	r	s	d	t	
$I_{1,2,1,0,0}$	4	0	1	3	11100_{2l} = Sector id 7
$I_{1,1,3,-1,0}$	5	1	2	3	
$I_{0,2,2,0,0}$	4	0	2	2	01100_{2l} = Sector id 6
$I_{0,1,2,-2,0}$	3	2	1	2	

Sector decomposition

Sector decomposition software

pySecDec (github.com/gudrunhe/secdec):

- Python core that generates C++ files.
- Can use different Monte-Carlo integration algorithms: Randomized Quasi-Monte Carlo (Qmc), Vegas/Suave/Divonne/Cuhre (Cuba), Cquad (Gsl).
- Optional GPU support (with Qmc).

Fiesta (bitbucket.org/feynmanIntegrals/fiesta):

- Mathematica core, generates/compiles to C++ behind the scenes.
- Custom Monte-Carlo integration on both CPU and GPU.

Sector decomposition with pySecDec

Usage overview (details: secdec.readthedocs.io):

1. Setup pySecDec by installing dependencies and running `make` in its release archive.
2. Use the Python module `pySecDec` to define the integral and generate the code for the integration library.
3. Compile the integration library.
4. Import the integration library from Python (or C++), call it to perform integration.

Differential equations

Method of differential equations

Consider a family of integrals depending on external momenta p_i and masses m_i:

$$I_{\nu_1,\nu_2,\ldots,\nu_N} \equiv \int \frac{\mathrm{d}^d l_1}{(2\pi)^d} \ldots \frac{\mathrm{d}^d l_L}{(2\pi)^d} \frac{1}{D_1^{\nu_1} \ldots D_N^{\nu_N}} = I_{\nu_1,\nu_2,\ldots,\nu_N}\left(\left\{p_i \cdot p_j, m_i^2\right\}\right).$$

With dimensional analysis one of the parameters can be scaled out, making remaining arguments dimensionless:

$$I\left(\left\{p_i \cdot p_j, m_i\right\}\right) = \left(p_1^2\right)^{\frac{d}{2}L - \sum_i \nu_i} I\left(\{x_i\}\right), \qquad \{x_i\} = \left\{\frac{p_i \cdot p_j}{p_1^2}, \frac{m_i^2}{p_1^2}\right\}.$$

Idea: if $\{x_i\} \neq \varnothing$, we can construct differential equations in x_i, and solve them instead of performing the loop integration directly.
(In practice: just set $p_1^2 = 1$, restore it in the end by dimensionality).

Constructing differential equations

Suppose IBP relations were solved, and we have a set of master integrals I_i.

1. Differentiate each I_i by one of the parameters, $x = m_a^2$ or $p_a \cdot p_b$:

$$\partial_{m_a^2} I = \int \frac{\mathrm{d}^d l_1}{(2\pi)^d} \cdots \frac{\mathrm{d}^d l_L}{(2\pi)^d} \, \partial_{m_a} \frac{1}{D_1^{\nu_1} \cdots D_N^{\nu_N}},$$

$$\partial_{p_a \cdot p_b} I = \left(G^{-1}\right)_{ia} p_i \cdot \left(\partial_{p_b} I\right), \quad \partial_{p_a \cdot p_a} I = \frac{1}{2} \left(G^{-1}\right)_{ia} p_i \cdot \left(\partial_{p_a} I\right),$$

   where G is the Gram matrix: $G_{ij} \equiv p_i \cdot p_j$.

2. Express the derivatives as integrals in the same family:

$$\partial_x I_i = \sum_k C_k \, I_{\nu_1^{(k)}, \nu_2^{(k)}, \ldots, \nu_N^{(k)}}.$$

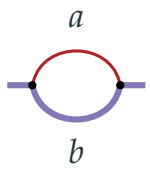
3. Use the IBP tables to reduce those integrals back to the master integrals, thus obtaining a linear differential equation system for I_i:

$$\partial_x I_i \overset{\text{IBP}}{=} \sum_j M_{ij} \, I_j.$$

Example: self-energy with one mass

Consider a family of self-energy with one mass integrals

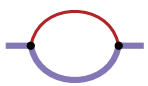
$$I_{a,b} \equiv \quad = \int \frac{\mathrm{d}^d l}{(2\pi)^d} \frac{1}{\left(q-l\right)^{2a} \left(l^2-m^2\right)^b}.$$



This family has two master integrals:

$$I_1 \equiv I_{0,1} = \quad = \int \frac{\mathrm{d}^d l}{(2\pi)^d} \frac{1}{l^2-m^2},$$

$$I_2 \equiv I_{1,1} = \quad = \int \frac{\mathrm{d}^d l}{(2\pi)^d} \frac{1}{\left(q-l\right)^2 \left(l^2-m^2\right)}.$$



Constructing differential equation system:

$$\partial_{m^2} \underbrace{\begin{pmatrix} I_1 \\ I_2 \end{pmatrix}}_{\mathbf{I}} = \begin{pmatrix} I_{0,2} \\ I_{1,2} \end{pmatrix} \overset{\text{IBP}}{=} \underbrace{\begin{pmatrix} \frac{-2+d}{2m^2} & 0 \\ \frac{2-d}{2m^2\left(m^2-q^2\right)} & \frac{-3+d}{m^2-q^2} \end{pmatrix}}_{\mathbb{M}} \underbrace{\begin{pmatrix} I_1 \\ I_2 \end{pmatrix}}_{\mathbf{I}}.$$

Solution via the epsilon-form

If we have a differential equation system for $\mathbf{I}$,

$$\partial_x \mathbf{I}(x,d) = \mathbb{M}(x,d)\,\mathbf{I}(x,d), \qquad \text{where } d = 4-2\varepsilon,$$

we can transform to a new basis $\mathbf{J}$ via the transformation matrix $\mathbb{T}$,

$$\mathbf{I}(x,\varepsilon) = \mathbb{T}(x,\varepsilon)\,\mathbf{J}(x,\varepsilon),$$

and for $\mathbf{J}$ the same differential equation system will look like

$$\partial_x \mathbf{J} = \mathbb{T}^{-1}\left(\mathbb{M}\mathbb{T} - \partial_x \mathbb{T}\right)\mathbf{J} \equiv \mathbb{M}'\,\mathbf{J}.$$

Idea: if this system is in an *ε-form* (has the dependence on ε factorized),

$$\partial_x \mathbf{J}(x,\varepsilon) = \varepsilon\,\mathbb{S}(x)\,\mathbf{J}(x,\varepsilon),$$

then the solution for $\mathbf{J}$ as a series in ε becomes trivial: [Henn '13]

$$\mathbf{J}(x,\varepsilon) \equiv \varepsilon^{k_0}\sum_{k=0}^{\infty}\varepsilon^k\,\mathbf{J}^{(k)}(x),$$

$$\sum \varepsilon^k\,\mathbf{J}^{(k)} = \sum \varepsilon^{k+1}\mathbb{S}\,\mathbf{J}^{(k)} \quad \Rightarrow \quad \mathbf{J}^{(k)}(x) = \int^x \mathbb{S}(x')\,\mathbf{J}^{(k-1)}(x')\mathrm{d}x' + \mathbf{C}^{(k)}.$$

Example, cont.: the epsilon-form

The ε-form can be achieved with a transformation to the following basis $\mathbf{J}$:

$$\mathbf{I} = \begin{pmatrix} (-1+2\varepsilon)\, m^2 & 0 \\ -1+\varepsilon & (-1+\varepsilon)\left(m^2 - q^2\right) \end{pmatrix} \mathbf{J}.$$

Differential equation system in $\mathbf{J}$ then has the form

$$\partial_{m^2} \mathbf{J} = \varepsilon \begin{pmatrix} -\frac{1}{m^2} & 0 \\ -\frac{1}{q^2}\frac{1}{m^2} - \frac{1}{q^2}\frac{1}{m^2-q^2} & -\frac{2}{m^2-q^2} \end{pmatrix} \mathbf{J}.$$

Accordingly, the solution is

$$\mathbf{J}^{(0)} = \mathbf{C}^{(0)},$$

$$\mathbf{J}^{(1)} = \mathbf{C}^{(1)} + \begin{pmatrix} -C_1^{(0)} \int^{m^2} \frac{\mathrm{d}m'^2}{m'^2} \\ \dots \end{pmatrix},$$

$$\mathbf{J}^{(2)} = \mathbf{C}^{(2)} + \begin{pmatrix} -C_1^{(1)} \int^{m^2} \frac{\mathrm{d}m'^2}{m'^2} + C_1^{(0)} \int^{m^2} \frac{\mathrm{d}m'^2}{m'^2} \int^{m'^2} \frac{\mathrm{d}m''^2}{m''^2} \\ \dots \end{pmatrix}.$$

Iterated integrals

The solution to differential equation in an ε-form always come as iterated integrals of the form of *multiple polylogarithms* (a.k.a Goncharov polylogarithms): [Goncharov '98]

$$G(w_1, w_2, \dots, w_n; x) \equiv \int_0^x \frac{\mathrm{d}t_1}{t_1 - w_1} \int_0^{t_1} \frac{\mathrm{d}t_2}{t_2 - w_2} \cdots \int_0^{t_n} \frac{\mathrm{d}t_n}{t_n - w_n}.$$

Special case for the trailing zeros:

$$G(\underbrace{0, \dots, 0}_{n}; x) \equiv \frac{1}{n!} \log^n(x).$$

Integration and differention, the simple case:

$$\int \mathrm{d}x \frac{1}{x - a} G(\vec{w}; x) = G(a, \vec{w}; x) + C, \quad \frac{\mathrm{d}}{\mathrm{d}x} G(w_1, \vec{w}; x) = \frac{1}{x - w_1} G(\vec{w}; x).$$

Software: GiNaC, Hpl, Harmpol, HyperInt, PolyLogTools, etc.

Example, cont.: the solution in GPLs

Rewriting the iterated integrals in terms of G:

$$\mathbf{J}^{(0)} = \mathbf{C}^{(0)},$$

$$\mathbf{J}^{(1)} = \mathbf{C}^{(1)} + \begin{pmatrix} -C_1^{(0)} G(0; m^2) \\ \dots \end{pmatrix},$$

$$\mathbf{J}^{(2)} = \mathbf{C}^{(2)} + \begin{pmatrix} -C_1^{(1)} G(0; m^2) + C_1^{(0)} G(0,0; m^2) \\ \dots \end{pmatrix}.$$

It is trivial to generate this answer to any required order. Note that the answer will only have $w_i \in \{0, q^2\}$. By rescaling the weights (or setting q^2 to 1), this can be turned into $w_i \in \{0, 1\}$, the *harmonic polylogarithms*.

Multiple polylogarithms, more properties

Differentiation in the general case when w_i may depend on x:

$$\frac{\mathrm{d}}{\mathrm{d}x} G(w_1, \dots, w_n; y) = \sum_i G(w_1, \dots, \cancel{w_i}, \dots, w_n; y) \frac{\mathrm{d}}{\mathrm{d}x} \log \frac{w_i - w_{i-1}}{w_i - w_{i+1}},$$

where $w_0 \equiv y$, and $w_{i+1} \equiv 0$.

Integration in the general case when w_i may depend on x is nontrivial, one must first represent G in a way that the integration variable only appears in the last argument. [Brown '08]

Automated via HyperInt (bitbucket.org/PanzerErik/hyperint): [Panzer '14]

```
$ maple
> read "HyperInt.mpl";
> fibrationBasis(Hlog(x,[y,x,y]),[x,y]);
Hlog(x,[y,y,y])-Hlog(x,[y,0,y])
```

Multiple polylogarithms and their relatives

The integral representation of multiple polylogarithms (G) is equivalent to the infinite sum representation (Li):

$$\mathrm{Li}_{m_1,\dots,m_n}(x_1,\dots,x_n) = \sum_{i_1>\cdots>i_n>0} \frac{x_1^{i_1}}{i_1^{m_1}} \cdots \frac{x_n^{i_n}}{i_n^{m_n}} =$$

$$= (-1)^n\, G\Big(\underbrace{0,\dots,0}_{m_1-1},\frac{1}{x_i},\dots,\underbrace{0,\dots,0}_{m_2-1},\frac{1}{x_1x_2\cdots x_n};1\Big).$$

- Note: there are conflicting conventions for the order of the indices in the Li summation. Above is the "physicist" notation (used in e.g. HPL, GiNaC, and the MZV datamine).
- The "mathematician" notation is reverse: $0 < i_1 < \cdots < i_n$; it was used by Goncharov, and is also used in HyperInt. The order of indices in the Multiple Zeta Values is also reversed there.

Multiple polylogarithms and their relatives, II

* *Logarithms* are GPLs with a single weight:

$$\log x = G(0;x), \qquad \log\left(\frac{a-x}{a}\right) = G(a;x).$$

* *Nielsen's generalized polylogarithms* are GPLs with $w_i \in \{0,1\}$:

$$S_{n,p}(x) = (-1)^p\, G(\underbrace{0,\ldots,0}_{n},\underbrace{1,\ldots,1}_{p};x).$$

* *Harmonic polylogarithms* (HPLs) are GPLs with $w_i \in \{0,\pm 1\}$:

$$H_{\ldots,+m,-n,0}(x) = (-1)^m G(\ldots,\underbrace{0,\ldots,0,1}_{m},\underbrace{0,\ldots,0,-1}_{n},0;x).$$

* *Two-dimensional HPLs* are GPLs with $w_i \in \{0,1,1-z,-z\}$.
* *Multiple Zeta Values* $\zeta_{\vec{w}}$ are just $H_{\vec{w}}(1)$ (in the "physicist" notation).

Fixing the integration constants

Differential equations only give the solution up to the integration constants. Finding these constants is the essential difficulty of the method. There are many ways.

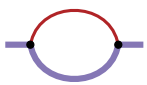
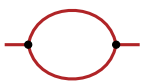
- By evaluating the integrals in a limit where they simplify.
  - Large mass limit. Small mass limit. [Smirnov '02]
  - Even massless integrals can be evaluated by adding masses to them, and connecting the large mass limit to the massless limit via the differential equations.
- Using the knowledge of the analytic properties of the integrals.
  - E.g.: enforcing regularity in the kinematic limits. [Gehrmann, Remiddi '00]
- From partial knowledge of the integrals values, such as a Mellin moment.
  - One can integrate over the semi-inclusive integrals to obtain the fully inclusive ones. [Gituliar '15]

Example, cont.: integration constants

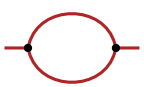
First, we consider I_1 (the vacuum bubble) "simple", and look it up in a book:

$$= -\frac{i\pi^{\frac{d}{2}}}{(2\pi)^d}\Gamma\left(1-\frac{d}{2}\right)\left(m^2-i0\right)^{\frac{d}{2}-1}.$$

For I_2: in the limit of zero mass the massive diagram becomes massless,

$$\lim_{m^2\to 0}$$  $$=$$  $$.$$

This massless integral we also know from a book:



$$= \frac{i\pi^{\frac{d}{2}}}{(2\pi)^d}\frac{\Gamma^2\left(\frac{d}{2}-1\right)\Gamma\left(2-\frac{d}{2}\right)}{\Gamma(d-2)}\left(-q^2-i0\right)^{\frac{d}{2}-2}.$$

Then, we can expand these in series' in ε, compare with the series from ε-form solution, and fix all $\mathbf{C}^{(k)}$.

See: `example.massive-self-energy.nb`.

The fundamental solution

A *fundamental solution* to $\partial_x \mathbf{I} = \mathbb{M}\,\mathbf{I}$ is an n by n matrix of independent solutions, such that any solution can be expressed as a linear combination of its columns.
A fundametal solution for a system in an ε-form, $\partial_x \mathbf{J} = \varepsilon\,\mathbb{S}\,\mathbf{J}$, can be constructed as a series in ε:

$$\mathbb{W} = \mathbb{1} + \varepsilon \int_{x_0}^{x} \mathrm{d}x'\,\mathbb{S}(x') + \varepsilon^2 \int_{x_0}^{x} \mathrm{d}x'\,\mathbb{S}(x') \int_{x_0}^{x'} \mathrm{d}x''\,\mathbb{S}(x'') + \dots .$$

The general solution is then just $\mathbb{W}$ multiplied by a vector of integration constants $\mathbf{C}$:

$$\mathbf{J}(x, \varepsilon) = \mathbb{W}(x, \varepsilon)\,\mathbf{C}(\varepsilon),$$

where the constants themselves are a series in ε,

$$\mathbf{C}(\varepsilon) \equiv \varepsilon^{k_0} \sum_{k=0}^{\infty} \varepsilon^k \mathbf{C}^{(k)}.$$

This is an alternative way to write down a solution for $\mathbf{J}$, with the benefit of being immediately extendable to the multivariate case.

The multivariate case

If differential equations in multiple variables are considered, and a combined ε-form is achieved,

$$\partial_{x_i} \mathbf{J}(\vec{x}, \varepsilon) = \varepsilon \, \mathbb{S}_i(\vec{x}) \, \mathbf{J}(\vec{x}, \varepsilon),$$

then writing down the solution in this case can be made easy by:

1. Choosing an integration contour along the axes x_i in an some order, for example from $(0, 0, \dots)$ to $(x_1, 0, \dots)$, then to $(x_1, x_2, 0, \dots)$, etc.
2. Writing down the fundamental solutions along each segment, $\mathbb{W}_i$. Because segments are chosen such that only x_i changes along each, $\mathbb{W}_i$ can be calculated the same as in single-variate case.

The general solution for $\mathbf{J}$ is then

$$\mathbf{J}(\vec{x}, \varepsilon) = \mathbb{W}_n(x_1, \dots, x_n, \varepsilon) \cdots \mathbb{W}_n(x_1, \varepsilon) \, \mathbf{C}(\varepsilon).$$

Overall this result will be a sum of terms of this form:

$$G(\text{arguments depending on } x_1, \dots, x_{n-1}; x_n) \cdots G(\text{constants}; x_1).$$

Lee algorithm

Lee algorithm for finding the epsilon-form

Overall idea: to go from a differential equation system

$$\partial_x \mathbf{I} = \mathbb{M}\,\mathbf{I}, \quad \text{where } \mathbb{M}(x,\varepsilon) = \sum_i \frac{\mathbb{A}_i(\varepsilon)}{(x-x_i)^{k_i}},$$

to an ε-form

$$\partial_x \mathbf{J} = \varepsilon\,\mathbb{S}\,\mathbf{J}, \quad \text{where } \mathbb{S}(x) = \sum_i \frac{\mathbb{S}_i}{x-x_i},$$

apply a series of simple basis transformation $\mathbf{I} = \mathbb{T}\,\mathbf{J}$, such that each brings the system a bit closer to an ε-form. [Lee '14]

1. If a higher pole is present (i.e. $k_i \neq 1$) then use a transformation that reduces the rank of $\mathbb{A}_i$, eventually eliminating it. ("Fuchsification").
2. Else, for the eigenvalues of $\mathbb{A}_i$ of the form $n + k\varepsilon$, use a transformation that shifts n by ± 1, eventually setting it to zero. ("Normalization").
3. If all $\mathbb{A}_i$ eigenvalues are proportional to ε, use a transformation that makes the whole $\mathbb{A}_i$ proportional to ε. ("Factorization").

Lee algorithm, fuchsification

Consider *a "balance" transformation between x_1 and x_2*:

$$\mathbb{T}(x,\varepsilon) = \overline{\mathbb{P}}(\varepsilon) + \frac{x - x_2}{x - x_1}\mathbb{P}(\varepsilon), \quad \text{with } \mathbb{P}^2 = \mathbb{P}, \text{ and } \mathbb{P} + \overline{\mathbb{P}} = \mathbb{1}.$$

If the matrix $\mathbb{M}$ has a pole at $x = x_1$ of power $n > 1$,

$$\mathbb{M} = \mathbb{A}_{-n}\left(x - x_1\right)^{-n} + \mathbb{A}_{-n+1}\left(x - x_1\right)^{-n+1} + \dots,$$

then either

1. there is such $\mathbb{P}$ and x_2 , that the transformed $\mathbb{A}_{-n}$ is of lower rank than $\mathbb{A}_{-n}$, while the leading expansion order around $x = x_2$ does increase beyound $n = 1$; or
2. the ε-form cannot be reached by any rational transformation.

Then, a series of balance transformations can decrease the rank of all $\mathbb{A}_{-n}$ with $n \neq 1$ to zero, transforming $\mathbb{M}$ into the *Fuchsian form*:

$$\mathbb{M}(x,\varepsilon) = \sum \frac{\mathbb{A}_i(\varepsilon)}{x - x_i}.$$

Lee algorithm, normalization

Differential equations for master integrals are observed to have a special feature: when transformed into Fuchsian form,

$$\mathbb{M} = \sum \frac{\mathbb{A}_i}{x - x_i},$$

the eigenvalues of $\mathbb{A}_i$ often have the form of $n + k\varepsilon$, where $n \in \mathbb{N}$.
Now, a balance can be found between x_1 and x_2 that does not spoil the Fuchsian form. Such a balance will shift one of the eigenvalues of $\mathbb{A}_1$ by $+1$, one of $\mathbb{A}_2$ by -1.
Then, a series of such balances can transform $\mathbb{M}$ into a *normalized Fuchsian form*: where all $\mathbb{A}_i$ have eigenvalues proportional to ε.

- Sometimes eigenvalues of the form $\frac{1}{2} + n + k\varepsilon$ are encountered. If they are present at up to three different $\mathbb{A}_i$, then it is possible to change the integration variable from x to such y, that the differential equation in y has all the eigenvalues in the form $n + k\varepsilon$.
- Sometimes eigenvalues are not linear in ε. This often means the master integral basis is linearly dependent.

Lee algorithm, factorization

Finally, once all the eigenvalues of $\mathbb{M}$ residues are proportional to ε, then either

1. the whole matrix can be made proportional to ε by a transformation that does not depend on x; or
2. the ε-form can not be reached.

Such transformation is searched for via an ansatz.
Note that on this step, and on all previous ones too, there exists multiple transformations that can achieve the desired form. As a result, the ε-form is not unique.

Lee algorithm, the multivariate case

If a system of differential equations in multiple variables is considered:

$$\partial_{x_i}\mathbf{I}(\vec{x},\varepsilon) = \mathbb{M}_i(\vec{x},\varepsilon)\,\mathbf{I}(\vec{x},\varepsilon),$$

then it is useful to have a single transformation $\mathbb{T}(x,\varepsilon)$ that transforms all of the differential equation systems into ε-form simultaneously,

$$\partial_{x_i}\mathbf{J}(\vec{x},\varepsilon) = \varepsilon\,\mathbb{S}_i(\vec{x})\,\mathbf{J}(\vec{x},\varepsilon), \qquad \text{with } \mathbf{I} = \mathbb{T}\,\mathbf{J}.$$

Single-variable Lee algorithm can be reused for this by:

1. Reducing $\mathbb{M}_1$ to an ε-form with $\mathbb{T}_1(x_1, x_2, \dots, \varepsilon)$.
2. Transforming all the equations with $\mathbb{T}_1$.
3. Reducing $\mathbb{M}_2$ to an ε-form with a such a $\mathbb{T}_2(x_2, \dots, \varepsilon)$ that is independent of x_1 and ε.
4. Transforming all the equations with $\mathbb{T}_2$. This will not spoil the ε-form in x_1 because $\mathbb{T}_2$ does not depend on x_1 or ε.
5. Repeating similarly for the rest of $\mathbb{M}_i$.

Epsilon-form software

Fuchsia (github.com/magv/fuchsia.cpp) [Gituliar, Magerya '17]

* Implements the Lee algorithm.
* Initial version in Python/SageMath, newer version in C++/GiNaC.
* Differential equation systems in multiple variables are supported.
* Can certify if a system is irreducible (with exceptions).
* Can suggest variable changes that will make it reducible.

Canonica (github.com/christophmeyer/CANONICA) [Meyer '17]

* Uses the rational ansatz method.
  Constructs an ansatz for the transformation matrix up to some powers of the variables, then solves for the coefficients.
* Written in Mathematica.
* Supports systems with multiple variables.

Epsilon (github.com/mprausa/epsilon) [Prausa '17]

* Implements the Lee algorithm.
* Doesn't support systems with multiple variables.

Summary

Summary

We have talked about:

* Computing loop amplitudes with Qgraf, Form, Color.h, Feynson.
* Mathematica as the hot glue.
* IBP reduction with Kira.
* Numerical evaluation with pySecDec.
* Differential equations, the ε-form, Fuchsia.

Summary

We have talked about:

* Computing loop amplitudes with QGRAF, FORM, COLOR.H, FEYNSON.
* MATHEMATICA as the hot glue.
* IBP reduction with KIRA.
* Numerical evaluation with PYSECDEC.
* Differential equations, the ε-form, FUCHSIA.

Thank you for your attention.