Form usage in pySecDec

Vitaly Magerya (CERN)

Liverpool, 2025

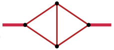
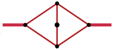
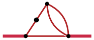
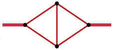
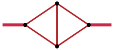
Plan of the talk

- Describe how we use FORM in pySecDec.
- Describe how we would like to use it.
- Ask if there are better ways.
- Compile a wishlist.

Bigger picture

Quantities of interest in high-energy physics: scattering amplitudes.

We compute these via:

- Feynman diagrams: $\mathscr{M}$ =  +  +  + ...
- Feynman integrals: $\mathscr{M} = C_1$  $+ C_2$  $+ C_3$  $+ C_4$  + ...
- IBP reduction: $\mathscr{M} = C'_1$  $+ C'_2$  $+ C'_3$  + ...
- Feynman integral evaluation:  = ?
  - Sector decomposition: pySecDec, Fiesta, sector_decomposition, etc.
  - Numerical differential equations: DiffExp, AMFlow, SeaSyde, Line, AmpRed, etc.
  - Mellin-Barnes representation: MB, Ambre, etc.
  - Tropical sampling: Feyntrop, Momtrop.
  - Etc.

Sector decomposition

$$I = (\text{Feynman parameterization}) = \int_0^1 \mathrm{d}x \int_0^1 \mathrm{d}y \left(3x^2 + 5y^2 + 7x^2y^2\right)^{-1+\varepsilon} = ?$$

How to evaluate?

* Expand the integrand in ε and integrate each order numerically?

$\Longrightarrow$ Not possible: the integrand diverges when both $x, y \to 0$.

Solution: [Heinrich '08; Binoth, Heinrich '00]

1. Isolate the divergence in x and y with sector decomposition:

$$I = \int \cdots \times \left(\theta\left(x > y\right) + \theta\left(y > x\right)\right) = \underbrace{\int_0^1 \mathrm{d}x \int_0^x \mathrm{d}y \cdots}_{\text{Sector 1}} + \underbrace{\int_0^1 \mathrm{d}y \int_0^y \mathrm{d}x \cdots}_{\text{Sector 2}}$$

* Normally done via geometric sector decomposition. [Bogner, Weinzierl '07; Kaneko, Ueda, '09; Schlenk, Zirke '16]

2. Rescale the integration region in each sector back to a hypercube:

$$\text{Sector 1} \overset{y=x\,z}{=} \int_0^1 \mathrm{d}x \int_0^1 \mathrm{d}z \; \underbrace{x^{-1+2\varepsilon}}_{\text{Divergent}} \; \underbrace{(3 + 5z^2 + 7x^2z^2)^{-1+\varepsilon}}_{\text{Finite}}$$

3. Subtract the divergence.

...

Divergence subtraction

3. Subtract the divergence, e.g.

$$\int_0^1 \mathrm{d}x\, x^{-1+k\varepsilon}\, I(x,\varepsilon) = \int_0^1 \mathrm{d}x\, \underbrace{x^{-1+k\varepsilon}\left(I(x,\varepsilon) - I(0,\varepsilon)\right)}_{\text{Finite}} + \underbrace{\int_0^1 \mathrm{d}x\, x^{-1+k\varepsilon}\; I(0,\varepsilon)}_{=\frac{1}{k\varepsilon} I(0,\varepsilon)},$$

so that Sector 1 becomes

$$\int_0^1 \mathrm{d}x \int_0^1 \mathrm{d}z\, x^{-1+2\varepsilon}\left(\left(3+5z^2+7x^2z^2\right)^{-1+\varepsilon} - \left(3+5z^2\right)^{-1+\varepsilon}\right) + \frac{1}{2\varepsilon}\int_0^1 \mathrm{d}z\, \left(3+5z^2\right)^{-1+\varepsilon}.$$

4. Expand the integrand in ε, e.g.:

$$\begin{aligned} x^{-1+k\varepsilon}\left(J(x)^{-1+\varepsilon} - J(0)^{-1+\varepsilon}\right) = {} & \frac{1}{x}\left(\frac{1}{J(x)} - \frac{1}{J(0)}\right)\varepsilon^0 + \\ & + \frac{k\log x}{x}\left(\frac{1}{J(x)} - \frac{1}{J(0)}\right)\varepsilon^1 + \frac{1}{x}\left(\frac{\log J(x)}{J(x)} - \frac{\log J(0)}{J(0)}\right)\varepsilon^1 + \\ & + \mathcal{O}\left(\varepsilon^2\right). \end{aligned}$$

5. Integrate each term in ε numerically.

Contour deformation

$$I = \text{(Feynman parameterization)} = \int \mathrm{d}^n\vec{x}\, \frac{U^\alpha\left(\vec{x}\right)}{F^\beta\left(\vec{x}, \ldots\right) + i0}$$

Problem: can't integrate numerically if $F = 0$ inside the integration region.
Solution: *deform $\vec{x}$ into the complex plane* to escape the pole:

$$\vec{x} \to \vec{x} + i\,\vec{\Delta}\left(\vec{x}\right) \qquad \text{and} \qquad \mathrm{d}^n\vec{x} \to \mathrm{d}^n\vec{x}\,\left|\frac{\partial\left(\vec{x} + i\,\vec{\Delta}\left(\vec{x}\right)\right)}{\partial\vec{x}}\right|$$

$$\Rightarrow \begin{cases} F \to F + i\,\Delta\,\partial_x F - \Delta^2\,\partial_x^2 F - i\,\Delta^3\,\partial_x^3 F + \mathscr{O}\left(\Delta^4\right), \\ \mathrm{Im}\,F \to \Delta\,\partial_x F - \Delta^3\,\partial_x^3 F + \mathscr{O}\left(\Delta^5\right). \end{cases}$$

Choose $\vec{\Delta}$ to enforce the $+i0$ prescription ($\mathrm{Im}\,F > 0$): $\vec{\Delta} = \lambda\,x\,(1 - x)\,\vec{\partial}_x F\left(\vec{x}\right)$.

- λ chosen heuristically: small enough so that $\mathrm{Im}\,F > 0$, but big enough to improve convergence.
- *Main computational cost: the evaluation of the Jacobian.*

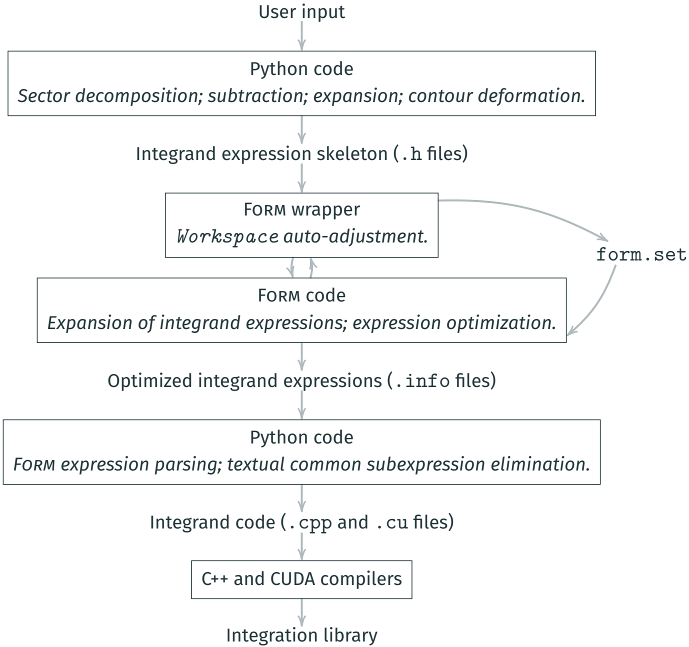
pySecDec

pySecDec: a Python library for *numerically evaluating parametric integrals* via sector decomposition and Randomized Quasi-Monte Carlo integration. [Heinrich et al ’23, ’21, ’18, ’17]

- Homepage: github.com/gudrunhe/secdec
- Installation: `python3 -m pip install pySecDec`.
- Primary use case:
  - *Evaluating a weighted sum of integrals many times* at different kinematic points.
- Mode of operation:
  - a user defines a weighted sum of integrals;
  - pySecDec prepares an integration library;
  - the user calls the integration library to get the value of the sums at given kinematic points.
- Other use cases:
  - Evaluating a single integral.
  - Expanding an integral in a small kinematic parameter (expansion by regions).

Demo: `pysecdec-example.py`.

FORM use workflow



Optimizing expressions

$$J = 2x^2y + 3xy^2 + \log\left(2x + 3y^4\right)$$

How to optimize this expression for evaluation speed?

Current approach:

- Use `#format O<n>` and `#optimize <expr>` on each argument, and then the whole expression, one at a time.
- Read in the result, dropping whitespace and "`; _+=`" sequences.
  - See `printing-example.frm`.
  - Setting `#:ContinuationLines` to 0 helps with "`; _+=`" (FORM 5!).
- Find "`pow(x,n)`" (via regular expressions) and expand into sequences of multiplications.
- Eliminate common subexpressions on textual basis.
  - I.e. transform "`x1=<text>;`" and "`x2=<text>;`" into "`x1=<text>;x2=x1;`".
- Clean up and write out C++.

Rat2c

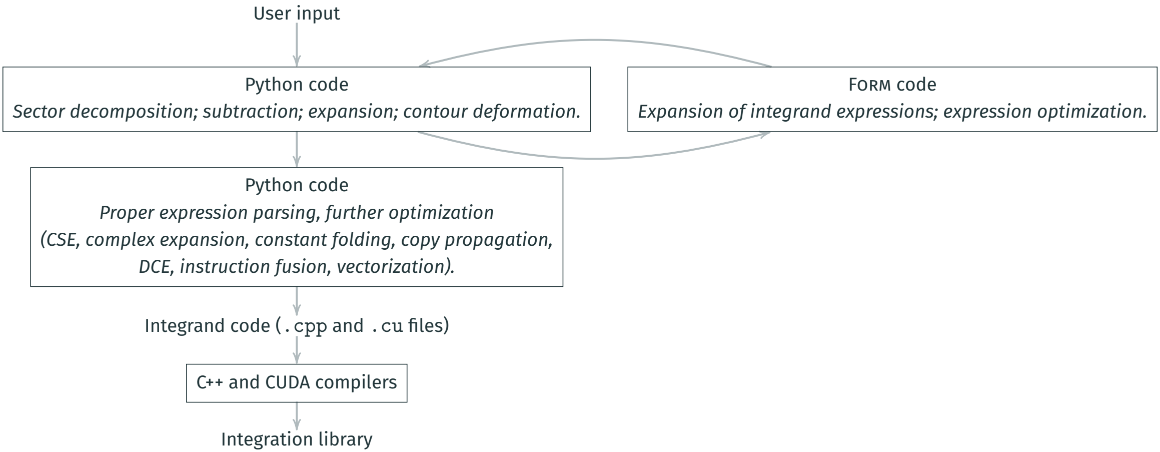
Rat2c: a Python program to convert one or more rational expressions into C using Form.

- Same Form output parsing & textual common subexpression elimination as in pySecDec.
- Homepage: github.com/magv/rat2c
- Example:

```
$ git clone https://github.com/magv/rat2c
$ echo 'x+y*x^2/2+z*x^3/y' | ./rat2c/rat2c -O4 -W1G -
#define inv(x) ((double)1/(double)(x))
#define quo(n,d) ((double)(n)/(double)(d))
void
function(
double *result,
const double x,
const double y,
const double z)
{
    double tmp1 = inv(y);
    double tmp2 = x*z*tmp1;
    tmp1 = quo(1,2)*y+tmp2;
    tmp2 = x*tmp1;
    tmp1 = 1+tmp2;
    result[0] = x*tmp1;
}
```

Demo: `rat2c-example.sh`.

New FORM use workflow



FORM as a library: #fromExternal and friends

FORM in slave mode:

- Start it with extra input and output pipes and a -pipe <r>,<w> argument.
- Read “<form pid>\n” from the output pipe.
- Write “<form pid>,<your pid>\n” back into the input pipe, and wait.
- In the FORM code, do #setExternal `PIPE1_', and then loop #fromExternal+.
- Each statement you write into the write pipe will be executed.
- Each #toExternal statement will write into the output pipe (which you should read from).

Implementations:

- github.com/tueda/python-form (Python);
- feyncalc.org/formlink (Matematica);
- pyform.py (Python, to become part of pySecDec).

Demo: pyform-example.py.

Optimizing expressions, the new way

$$J = 2x^2y + 3xy^2 + \log\left(2x + 3y^4\right)$$

How to optimize this expression for evaluation speed?

New approach:

- Define an expression. Use `ArgToExtraSymbol` to remember arguments of all functions.
- Use `#format O<n>` and `#optimize <expr>` on the current expression.
- Read out all newly defined extra symbols, and repeat for each of them.

Once the FORM part is done:

- Parse the resulting expressions, splitting them into individual arithmetic operations.
- Eliminate common subexpressions, fold constants, propagate copies, fuse multiply-add sequences, expand complex expressions, vectorize.
- Write out C++.

Demo: `codeopt-example.py`.

How well does this work for us?

Operation count for different expressions:

	O0	O1	O2	O3	O4	Via minors
4-loop massive banana, F	90	45	36	34	33	
7-loop massive banana, F	312	99	82	65	69	
4-loop massive banana, $\partial^2 F/\partial x_i/\partial x_j$	321	236	213	220	221	
7-loop massive banana, $\partial^2 F/\partial x_i/\partial x_j$	2725	1265	1170	1408	1356	
4x4 matrix determinant	60	40	28	31	29	28
5x5 matrix determinant	320	125	75	119	84	75
6x6 matrix determinant	1950	336	186	381	227	186
7x7 matrix determinant	13692	833	441	2461	735	441

Demo: `opcount-example.py`.

Wishlist

First priority:

- Make sure all buffers are auto-resizable, so no restarts are needed.
    - Never, ever, crash because of reasons that can be fixed automatically.

Really useful:

- Efficient input of very long expressions without manual massaging.

Would be nice:

- Ideas to improve nested multi-expression code optimization?
- A way to run `.clear` while in slave mode.
- Control over the printed line breaks and whitespace.
- A way to print a subrange of extra symbols.
- A way to run `#do x = {1,2}` with one or zero items.

Bonus: FORM usage in ALIBRARY

ALIBRARY: a Matematical library for computing Feynman amplitudes.

- Interface to QGRAF (diagrams), FEYNSON (diagram symmetries), GRAPHVIZ (diagram plotting), FORM (Dirac traces, index contraction, scalar product expansion, etc), COLOR.H (color tensor sums), KIRA or FIRE+LIRERED (IBP), pySECDEC.
- Homepage: github.com/magv/alibrary

FORM usage method:

- Export from Mathematica to FORM, run FORM code, import the result back.

Inside FORM code each transformation:

- finds unique factors it will act upon (`putInside` + `argToExtraSymbol`);
- hides the rest of the expression (`pushHide`), leaving only a sum of the unique factors;
- transforms the unique factors (`traceN`, `docolor()`, etc);
- creates a table for unique factor mapping (via `fillExpression`);
- puts the transformed factors back into the original expression (`popHide` + `id`).

See: `uniqbegin()` and `uniqend()` in `library.frm`.
Demo: `alibrary-example.m` (also `photon-propagator.m`).