New Features in FormCalc

Thomas Hahn

Max-Planck-Institut für Physik München
FormCalc is a matrix-element generator that turns FeynArts amplitudes up to 1-loop into a Fortran code for computing the partonic squared matrix element.

The generated code can be run with FormCalc’s own driver programs, or used with other ‘Frontends’, e.g. Monte Carlos.
FormCalc Internals

FormCalc

Mathematica

FORM

FeynArts amplitudes

Analytical results

Generated Code

SquaredME

RenConst

Fortran

Driver programs

Utilities library

T. Hahn, News Features in FormCalc – p.3
Overview

New Features presented in this talk:

- New FeynEdit tool,
- Mathematica interface,
- Abbreviations are split into tree + loop parts,
- Fierz identities in 4D,
- Fully analytic amplitudes,
- New functions for renormalization constants,
- Separate diagonalization package.
\begin{feynartspicture}(150,150)(1,1)
  \FADiagram{}
  \FAProp(6.,10.)(14.,10.)(0.8,){/ScalarDash}{-1}
  \FALabel(10.,5.73)[t]{$G$}
  \FAProp(6.,10.)(14.,10.)(-0.8,){/ScalarDash}{1}
  \FALabel(10.,14.27)[b]{$G$}
  \FAProp(0.,10.)(6.,10.)(0.,){/Sine}{0}
  \FALabel(3.,8.93)[t]{$\gamma$}
  \FAProp(20.,10.)(14.,10.)(0.,){/Sine}{0}
  \FALabel(17.,11.07)[b]{$\gamma$}
  \FAVert(6.,10.){0}
  \FAVert(14.,10.){0}
\end{feynartspicture}
Editing Feynman Diagrams

The elements of the diagram are easy to recognize and it is straightforward to make changes e.g. to the label text using any text editor. It is less straightforward, however, to alter the geometry of the diagram, i.e. to move vertices and propagators.

The new tool *FeynEdit* lets the user:

- copy-and-paste the \LaTeX\ code into the lower panel of the editor,
- visualize the diagram,
- modify it using the mouse, and finally
- copy-and-paste it back into the text.
Mathematica Interface

The new Mathematica Interface turns the generated stand-alone Fortran code into a Mathematica function for evaluating the cross-section or decay rate as a function of user-selected model parameters.

The benefits of such a function are obvious, as the whole instrumentarium of Mathematica commands can be applied to them. Just think of

```mathematica
FindMinimum[σ[TB, MA0], {{TB, 5}, {MA0, 250}}]
ContourPlot[σ[TB, MA0], {{TB, 5}, {MA0, 250}}]
...
```
Mathematica Interface - Input

The changes to the code are minimal.

Example line in run.F for Stand-alone Fortran code:

```
#define LOOP1 do 1 TB = 5, 50, 5
```

Change for the Mathematica Interface:

```
#define LOOP1 call MmaGetReal(TB)
```

The variable TB is ‘imported’ from Mathematica now, i.e. the cross-section function in Mathematica becomes a function of TB hereby.

The user has full control over which variables are ‘imported’ from Mathematica and which are set in Fortran.
Similar to the MmaGetReal invocations, the Fortran program can also ‘export’ variables to Mathematica.

For example, the line that prints a parameter in the stand-alone code is

```
#define PRINT1 SHOW "TB", TB
```

becomes

```
#define PRINT1 call MmaPutReal("TB", TB)
```

for the Mathematica Interface and transmits the value of TB to Mathematica.
Mathematica Interface – Usage

Once the changes to `run.F` are made, the program `run` is compiled as usual:

```
./configure
make
```

It is then loaded in Mathematica with

```
Install["run"]
```

Now a Mathematica function of the same name, `run`, is available. There are two ways of invoking it:

**Compute a differential cross-section at** \( \sqrt{s} = \text{sqrtS} \):

```
run[\text{sqrtS}, \text{arg1}, \text{arg2}, ...]
```

**Compute a total cross-section for** \( \text{sqrtS}_{\text{from}} \leq \sqrt{s} \leq \text{sqrtS}_{\text{to}} \):

```
run[\{\text{sqrtS}_{\text{from}}, \text{sqrtS}_{\text{to}}\}, \text{arg1}, \text{arg2}, ...]
```
Mathematica Interface – Data Retrieval

The output of the function `run` is an integer which indicates how many records have been transferred. For example:

\[
\text{Para}[1] = \{\text{TB }\to 5., \text{ MA0 }\to 250.\}
\]
\[
\text{Data}[1] = \{\text{DataRow}[\{500.\}, \{0.0539684, 0.\}, \{2.30801 \times 10^{-21}, 0.\}], \\
\text{DataRow}[\{510.\}, \{0.0515943, 0.\}, \{4.50803 \times 10^{-22}, 0.\}]\}
\]

Para contains the parameters exported from the Fortran code. Data contains:

- the independent variables, 
  here e.g. \(\{500.\} = \{\sqrt{s}\}\),

- the cross-sections, 
  here e.g. \(\{0.0539684, 0.\} = \{\sigma_{\text{tree tot}}, \sigma_{\text{1-loop tot}}\}\), and

- the integration errors, 
  here e.g. \(\{2.30801 \times 10^{-21}, 0.\} = \{\Delta \sigma_{\text{tree tot}}, \Delta \sigma_{\text{1-loop tot}}\}\).
Abbreviations

Abbreviations are perhaps the most powerful method in FormCalc to compactify and optimize the Fortran code.

\[ \text{AbbSum29} = \text{Abb2} + \text{Abb22} + \text{Abb23} + \text{Abb3} \]

\[ \text{Abb22} = \text{Pair1} \text{Pair3} \text{Pair6} \]

\[ \text{Pair3} = \text{Pair}[e[3], k[1]] \]

The full expression corresponding to \text{AbbSum29} is

\[
\begin{align*}
\text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[1]] \text{Pair}[e[4], k[1]] + \\
\text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[2]] \text{Pair}[e[4], k[1]] + \\
\text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[1]] \text{Pair}[e[4], k[2]] + \\
\text{Pair}[e[1], e[2]] \text{Pair}[e[3], k[2]] \text{Pair}[e[4], k[2]]
\end{align*}
\]
Categories of Abbreviations

- Abbreviations are *recursively defined* in several levels.
- When generating Fortran code, FormCalc introduces another set of abbreviations for the *loop integrals*.

In general, the *abbreviations are thus costly in CPU time*. It is key to a decent performance that the abbreviations are separated into different *Categories*:

- Abbreviations that depend on the helicities,
- Abbreviations that depend on angular variables,
- Abbreviations that depend only on $\sqrt{s}$.

Correct execution of the categories guarantees that *almost no redundant evaluations* are made and makes the generated code essentially as fast as hand-tuned code.
Splitting Abbreviations

The current version splits the abbreviations into such that are needed for the tree-level part and the rest:

OLD

Compute abbr

Compute $\mathcal{M}^{\text{tree}}$

Compute $\mathcal{M}^{1\text{-loop}}$

NEW

Compute abbr$_{\text{tree}}$

Compute $\mathcal{M}^{\text{tree}}$

Compute abbr$_{1\text{-loop}}$

Compute $\mathcal{M}^{1\text{-loop}}$

CPU-time (rough)

$\{\}$ 5 \%

$\{\}$ 95 \%

$\{\}$ .1\%

$\{\}$ .1\%

Application: separate phase-space integration of tree-level and one-loop component.

T. Hahn, News Features in FormCalc – p.14
The Fierz identities rearrange fermion chains by switching spinors, i.e.

$$\langle 1 | \Gamma_i | 2 \rangle \langle 3 | \Gamma_j | 4 \rangle = \sum c_{kl} \langle 1 | \Gamma_k | 4 \rangle \langle 3 | \Gamma_l | 2 \rangle$$

This is important in particular if one wants to extract certain predefined structures from the amplitude, most notably Wilson coefficients.

The latest FormCalc version offers a new option for the CalcFeynAmp function, e.g.

```
CalcFeynAmp[... , FermionOrder -> {2, 1, 3, 4}]
```

With this option, CalcFeynAmp tries to bring the spinor chains into the order $$\langle 2 | X | 1 \rangle \langle 3 | Y | 4 \rangle$$.
Fully Analytic Amplitudes

The ‘smallest’ object appearing in the output of CalcFeynAmp is a four-vector, i.e. FormCalc does not normally go into components. Those were inserted only in the numerical part. This has advantages: for example, the analytical expression does not reflect a particular phase-space parameterization.

Old method of obtaining analytical expression:

\[ M = \sum c_i F_i \quad \Rightarrow \quad |M|^2 = \sum c_i c^*_j (F_i F^*_j) \]

Thus: size of analytical expression for \(|M|^2\) scales as \(N^2\), rather than as \(N\) like \(M\).
Fully Analytic Amplitudes

New method of obtaining analytical expression (same as Fortran):

- **Set the external vectors** with the VecSet function (works almost exactly like the Fortran function), e.g.
  \[
  \text{VecSet}[1, m1, p1, \{0, 0, 1\}]
  \]

- **Evaluate your amplitude** with ToComponents, e.g.
  \[
  \text{ToComponents}[\text{amp}, "+-+-"]
  \]

This delivers an expression in terms of the phase-space parameters used in VecSet.

This is a very new function and its usefulness very likely depends on the size of the amplitude.
New Functions for Renormalization Constants

New functions have been introduced to simplify the definition of renormalization constants. For example, the entire renormalization section of the Standard Model now fits here:

\[
\begin{align*}
\text{RenConst}\{\text{dMf1}[t_\_, j_1_\_]\} & := \text{MassRC}[F[t, \{j_1\}]] \\
\text{RenConst}\{\text{dZfL1}[t_\_, j_1_\_, j_2_\_]\} & := \text{FieldRC}[F[t, \{j_1\}], F[t, \{j_2\}]][[1]] \\
\text{RenConst}\{\text{dZfR1}[t_\_, j_1_\_, j_2_\_]\} & := \text{FieldRC}[F[t, \{j_1\}], F[t, \{j_2\}]][[2]] \\
\text{RenConst}\{\text{dMZsq1}\} & := \text{MassRC}[V[2]] \\
\text{RenConst}\{\text{dMWsq1}\} & := \text{MassRC}[V[3]] \\
\text{RenConst}\{\text{dMHsq1}\} & := \text{MassRC}[S[1]] \\
\text{RenConst}\{\text{dZAA1}\} & := \text{FieldRC}[V[1]] \\
\text{RenConst}\{\text{dZAZ1}\} & := \text{FieldRC}[V[1], V[2]] \\
\text{RenConst}\{\text{dZZA1}\} & := \text{FieldRC}[V[2], V[1]] \\
\text{RenConst}\{\text{dZZZ1}\} & := \text{FieldRC}[V[2]] \\
\text{RenConst}\{\text{dZGO1}\} & := \text{FieldRC}[S[2]] \\
\text{RenConst}\{\text{dZW1}\} & := \text{FieldRC}[V[3]] \\
\text{RenConst}\{\text{dZGp1}\} & := \text{FieldRC}[S[3]] \\
\text{RenConst}\{\text{dZHI1}\} & := \text{FieldRC}[S[1]] \\
\text{RenConst}\{\text{dTH1}\} & := \text{TadpoleRC}[S[1]] \\
\text{RenConst}\{\text{dSW1}\} & := CW^2/SW/2 (\text{dMZsq1}/MZ^2 - \text{dMWsq1}/MW^2) \\
\text{RenConst}\{\text{dZe1}\} & := -1/2 (\text{dZAA1} + SW/CW \text{dZZA1})
\end{align*}
\]
Separate Diagonalization Package

The diagonalization routines included in FormCalc have been extended and made available as a separate package (physics/0607103).

- **HEigensystem** diagonalizes a Hermitian matrix,
- **SEigensystem** diagonalizes a complex symmetric matrix,
- **CEigensystem** diagonalizes a general complex matrix,
- **TakagiFactor** computes the Takagi factorization of a symmetric matrix (e.g. the neutralino mass matrix),
- **SVD** performs the Singular Value Decomposition.
Jacobi Algorithm

The Diag routines are based on the Jacobi algorithm. This is conceptually simple but scales less favourably than e.g. the QR method. Applicability range is thus small to medium-size matrices.

- rather compact code ($\sim 3$ kBytes each), therefore easy to adapt to own conventions,
- implemented in Fortran 77, but C/C++ and Mathematica interface included,
- LGPL license.
Summary and Availability

- The drawing tool FeynEdit will shortly be available from http://www.feynarts.de.
- The upcoming FormCalc version 5.3 with the new features
  - Mathematica interface,
  - Abbreviations are split into tree + loop parts,
  - Fierz identities in 4D,
  - Fully analytic amplitudes,
  - New functions for renormalization constants
will be available from http://www.feynarts.de/formcalc.
- The diagonalization package is available from http://www.feynarts.de/diag.