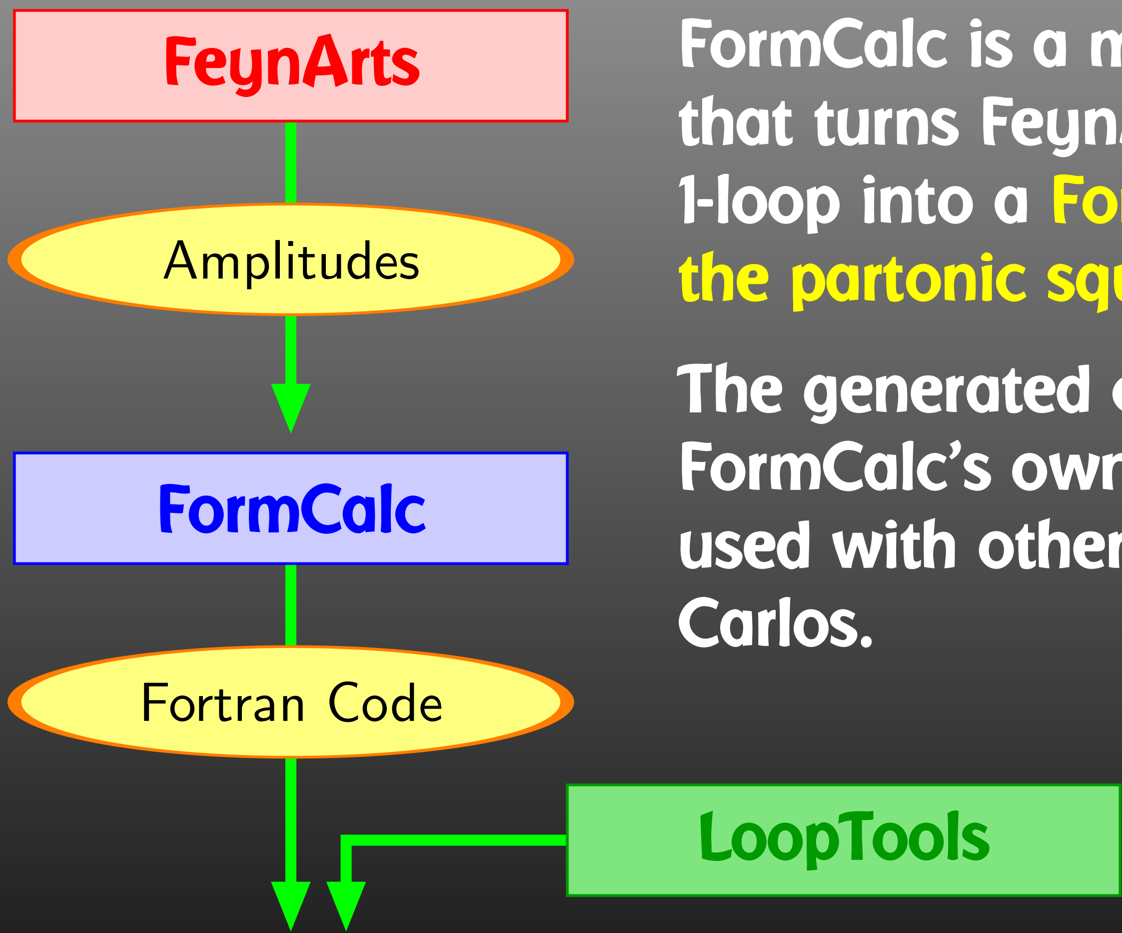
Feyn Tools for SUSY

Thomas Hahn

Max-Planck-Institut für Physik
München

The Feyn System

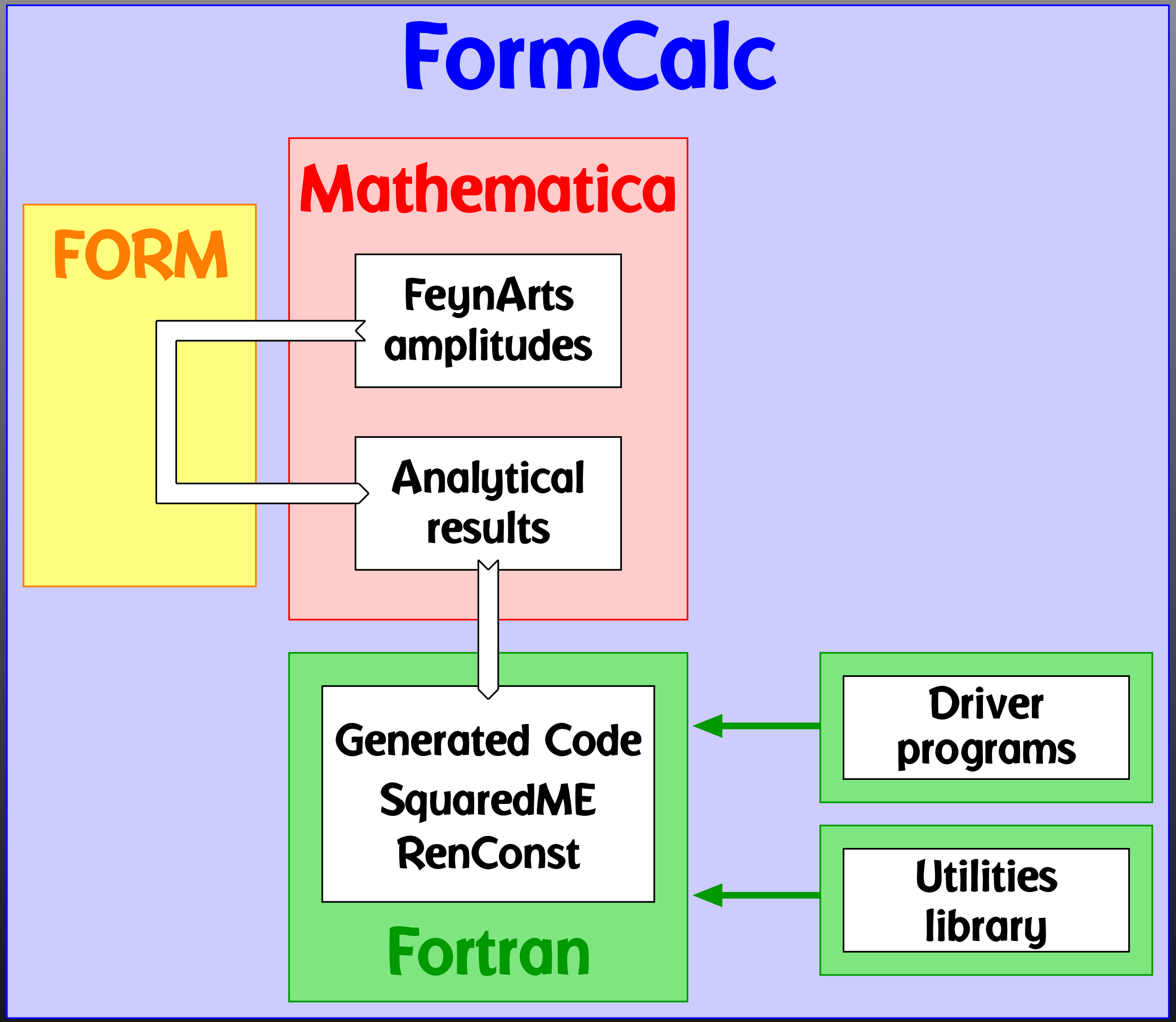


FormCalc is a matrix-element generator that turns FeynArts amplitudes up to 1-loop into a **Fortran code for computing the partonic squared matrix element.**

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

$|\mathcal{M}|^2 \longrightarrow$ Cross-sections, Decay rates, . . .

FormCalc Internals



Overview

New Features presented in this talk:

- Partial (Add-On) model files,
- New FeynEdit tool,
- Mathematica interface,
- Abbreviations are split into tree + loop parts,
- Fermion-Chain rearrangement in 4D,
- Fully analytic amplitudes,
- New functions for renormalization constants,
- Separate diagonalization package.
- SLHALib including latest SLHA 2 changes.

Partial (Add-On) Model Files

FeynArts 3.3 distinguishes

- Basic Model Files and
- Partial (Add-On) Model Files.

Basic Model Files, e.g. SM.mod, MSSM.mod, can be modified by Add-On Model Files, for example,

```
InsertFields[..., Model -> {"MSSMQCD", "FV", "HMix"}]
```

This loads the Basic Model File MSSMQCD.mod and modifies it through the Add-Ons FV.mod (non-minimal flavour violation) and HMix.mod (3×3 neutral Higgs mixing).

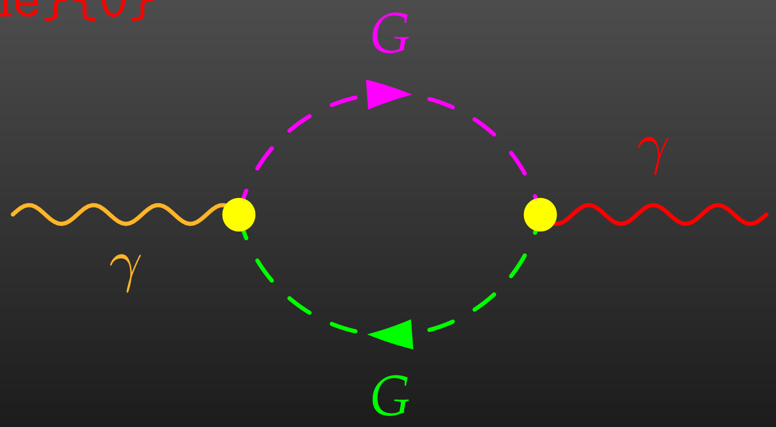
Model files can thus be built up from several parts.

The 'old' FVMSSM.mod exists for compatibility and just has

```
LoadModel[{"MSSMQCD", "FV"}]
```

FeynArts Paint Output

```
\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}
```

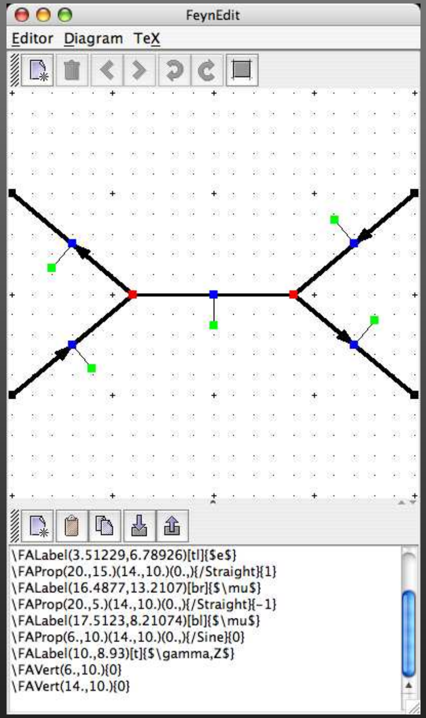


Editing Feynman Diagrams

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

```
FindMinimum[sigma[TB, MA0], {{TB, 5}, {MA0, 250}}]
ContourPlot[sigma[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.**

Mathematica Interface – Output

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 $\sqrt{s} = \mathtt{sqrtS}$:

```
run[sqrtS, arg1, arg2, ...]
```

Compute a total cross-section for $\mathtt{sqrtSfrom} \leqslant \sqrt{s} \leqslant \mathtt{sqrtSto}$:

```
run[{sqrtSfrom, sqrtSto}, arg1, arg2, ...]
```

Mathematica Interface – Data Retrieval

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

```
Para[1] = {TB -> 5., MA0 -> 250.}
Data[1] = {DataRow[{500.}, {0.0539684, 0.}, {2.30801 10^-21, 0.}],
           DataRow[{510.}, {0.0515943, 0.}, {4.50803 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. $\{0.0539684,\ 0.\} = \{\sigma_{\text{tot}}^{\text{tree}}, \sigma_{\text{tot}}^{\text{1-loop}}\}$, **and**
- **the integration errors,**
  here e.g. $\{2.30801\ 10\hat{}-21,\ 0.\} = \{\Delta\sigma_{\text{tot}}^{\text{tree}}, \Delta\sigma_{\text{tot}}^{\text{1-loop}}\}$.

Abbreviations

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

```
AbbSum29 = Abb2 + Abb22 + Abb23 + Abb3

    Abb22 = Pair1 Pair3 Pair6

        Pair3 = Pair[e[3],k[1]]
```

The full expression corresponding to `AbbSum29` is

```
Pair[e[1],e[2]] Pair[e[3],k[1]] Pair[e[4],k[1]] +
Pair[e[1],e[2]] Pair[e[3],k[2]] Pair[e[4],k[1]] +
Pair[e[1],e[2]] Pair[e[3],k[1]] Pair[e[4],k[2]] +
Pair[e[1],e[2]] Pair[e[3],k[2]] Pair[e[4],k[2]]
```

Categories of Abbreviations

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

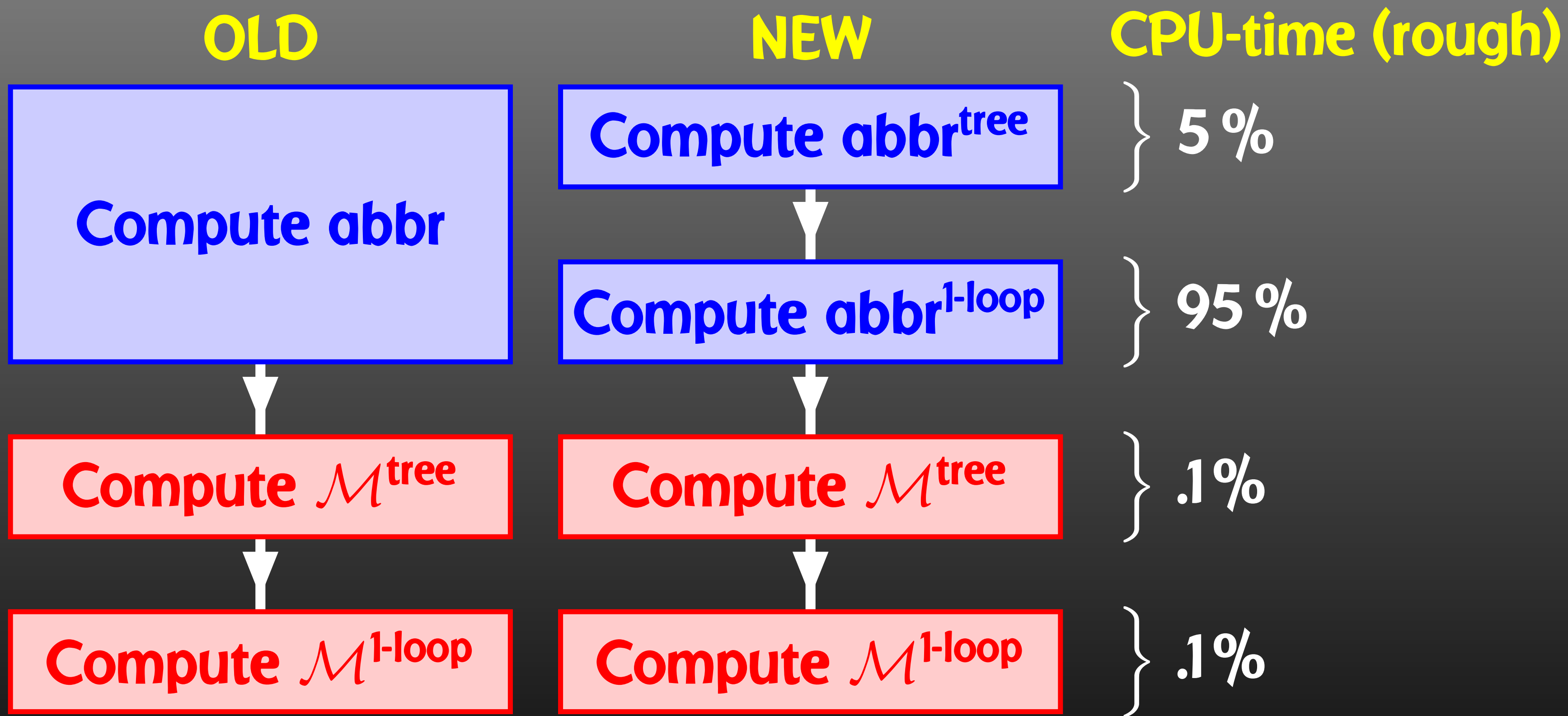
In general, the abbreviations are thus costly in CPU time.
It is key to a decent performance that the abbreviations are separated into different Categories:

- 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:



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

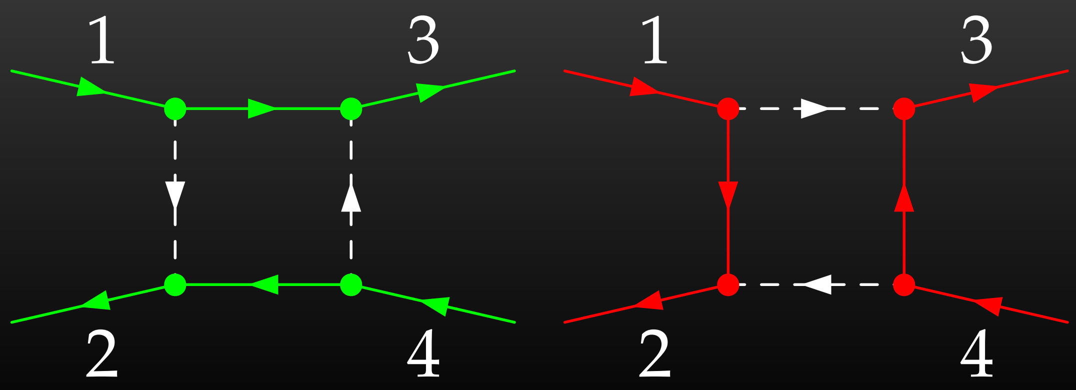
Fermion-Chain Rearrangement in 4D

Two new functions/options help in selecting and identifying given fermion structures. This is most obviously useful for the extraction of Wilson coefficients from an amplitude.

The new FeynArts function FermionRouting can be used to select diagrams according to their fermion structure, e.g.

```
DiagramSelect[...,
   FermionRouting[##] === {1,3, 2,4} & ]
```

selects only diagrams where external legs 1-3 and 2-4 are connected through fermion lines.



Fermion-Chain Rearrangement in 4D

FormCalc's CalcFeynAmp has the new **FermionOrder option** with which a **given ordering of the external spinors can be enforced** on spinor chains. For example,

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

brings the spinor chains into the order $\langle 2|\, X \,|1\rangle\, \langle 3|\, Y \,|4\rangle$ using Fierz and charge-conjugation identities.

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:

$$\mathcal{M} = \sum^{N} c_i F_i \quad \Rightarrow \quad |\mathcal{M}|^2 = \sum^{N^2} c_i c_j^* (F_i F_j^*)$$

Thus: size of analytical expression for $|\mathcal{M}|^2$ scales as N^2, rather than as N like $\mathcal{M}$.

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.

```
VecSet[1, m1, p1, {0, 0, 1}]
```

- **Evaluate your amplitude** with `ToComponents`, e.g.

```
ToComponents[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:

```
RenConst[ dMf1[t_, j1_] ] := MassRC[F[t, {j1}]]
RenConst[ dZfL1[t_, j1_, j2_] ] := FieldRC[F[t, {j1}], F[t, {j2}]][[1]]
RenConst[ dZfR1[t_, j1_, j2_] ] := FieldRC[F[t, {j1}], F[t, {j2}]][[2]]
RenConst[ dMZsq1 ] := MassRC[V[2]]
RenConst[ dMWsq1 ] := MassRC[V[3]]
RenConst[ dMHsq1 ] := MassRC[S[1]]
RenConst[ dZAA1 ] := FieldRC[V[1]]
RenConst[ dZAZ1 ] := FieldRC[V[1], V[2]]
RenConst[ dZZA1 ] := FieldRC[V[2], V[1]]
RenConst[ dZZZ1 ] := FieldRC[V[2]]
RenConst[ dZG01 ] := FieldRC[S[2]]
RenConst[ dZW1 ] := FieldRC[V[3]]
RenConst[ dZGp1 ] := FieldRC[S[3]]
RenConst[ dZH1 ] := FieldRC[S[1]]
RenConst[ dTH1 ] := TadpoleRC[S[1]]
RenConst[ dSW1 ] := CW^2/SW/2 (dMZsq1/MZ^2 - dMWsq1/MW^2)
RenConst[ dZe1 ] := -1/2 (dZAA1 + SW/CW dZZA1)
```

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.

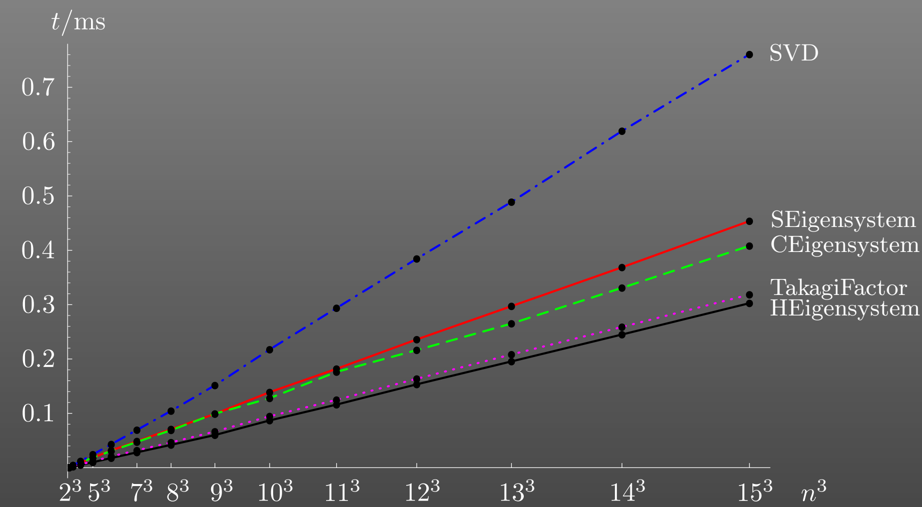
Jacobi Algorithm

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.

Timings on an AMD X2-5000:



- 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.

SLHA I/O Library

- The SUSY Les Houches Accord defines a common interface for SUSY tools.
- Reading/writing SLHA files not entirely straightforward.
- The SLHA I/O Library fills this gap:
  - Implemented as native Fortran 77 Library.
  - All data transferred in one double-complex array.
  - This array is indexed by preprocessor macros, e.g. `MinPar_TB` instead of `slhadata(20)`.
  - Main functions: `SLHARead`, `SLHAWrite`.
  - Implements the Latest (almost final) version of the SLHA2.
- Freely available at http://www.feynarts.de/SLHA.

Summary and Availability

- The drawing tool FeynEdit is available from http://www.feynarts.de.
- The current FormCalc version 5.3 has the new features
  - Mathematica interface,
  - Abbreviations are split into tree + loop parts,
  - Fermion-chain rearrangement in 4D,
  - Fully analytic amplitudes,
  - New functions for renormalization constants

  and is available from http://www.feynarts.de/formcalc.
- The diagonalization package is available from http://www.feynarts.de/diag.
- The SLHA Library is available from http://www.feynarts.de/slha.