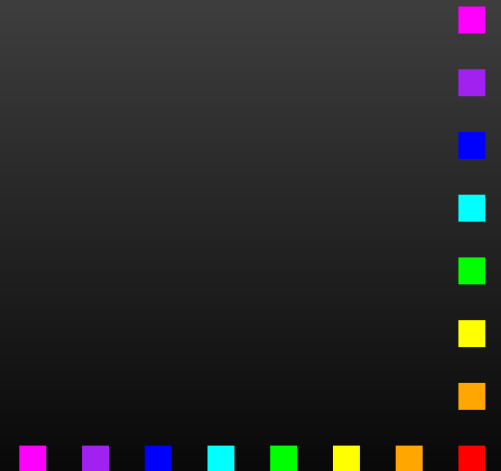


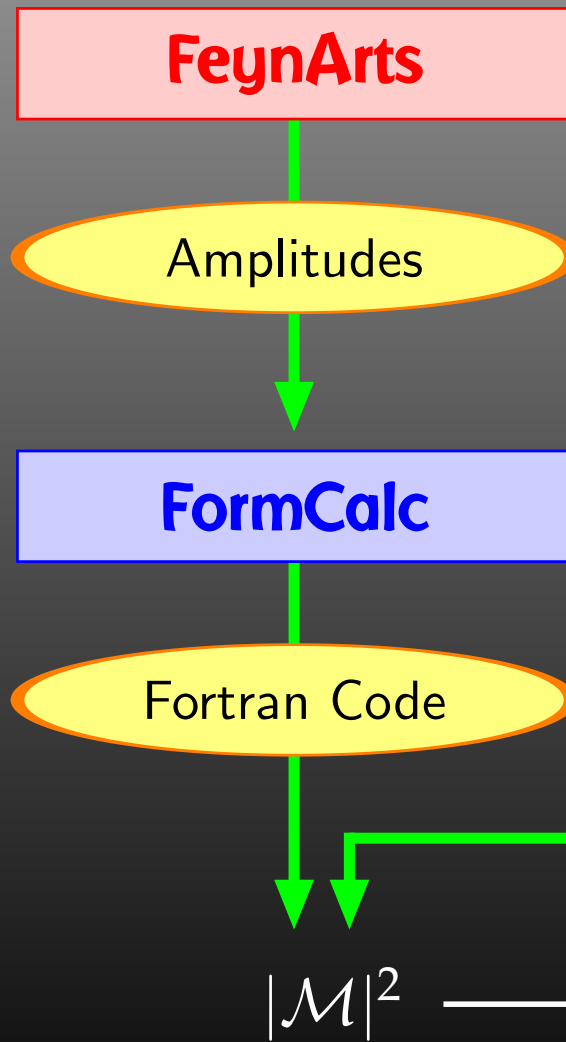
New Features in FormCalc

Thomas Hahn

Max-Planck-Institut für Physik
München



The System

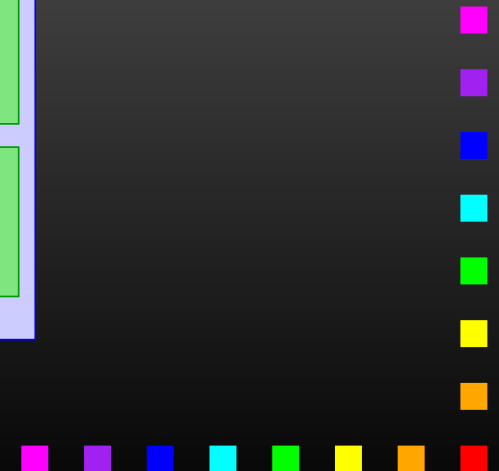
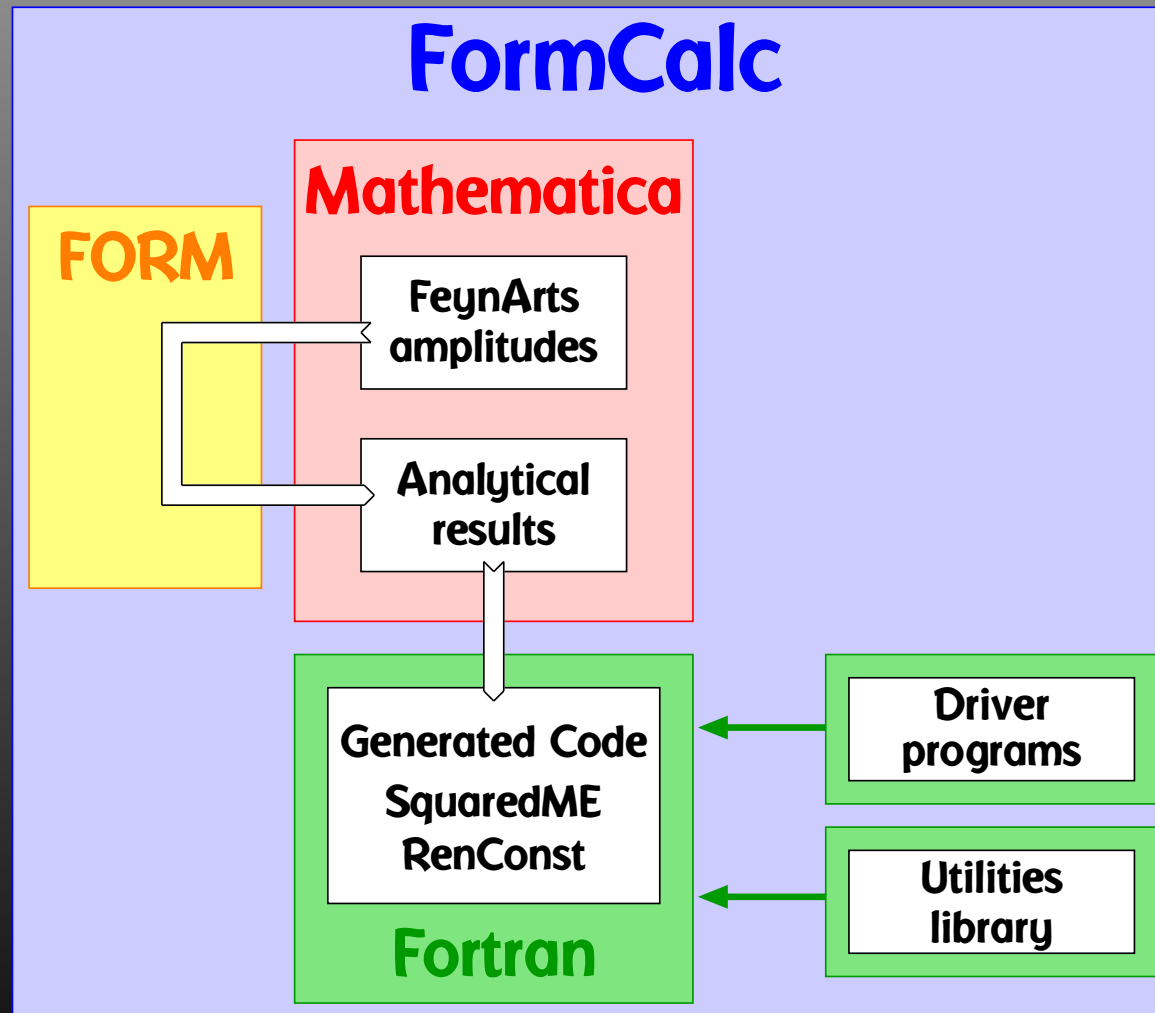


FormCalc is a matrix-element generator that turns FeynArts amplitudes up to 1-loop into a **Fortran code for computing the partonic squared matrix element.**

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



FormCalc Internals



Overview

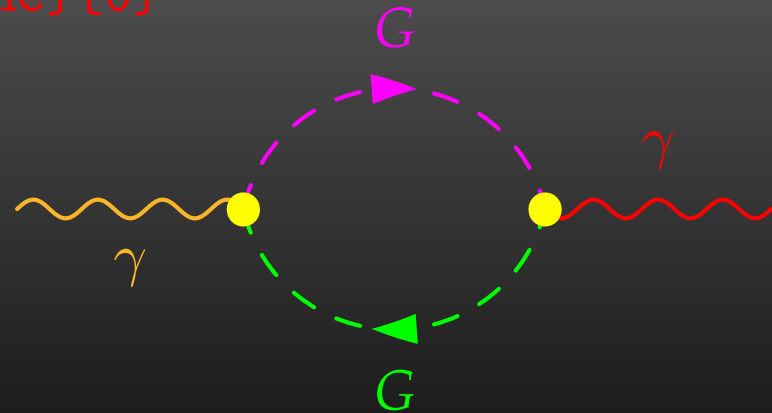
New Features presented in this talk:

- **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.**



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}
```



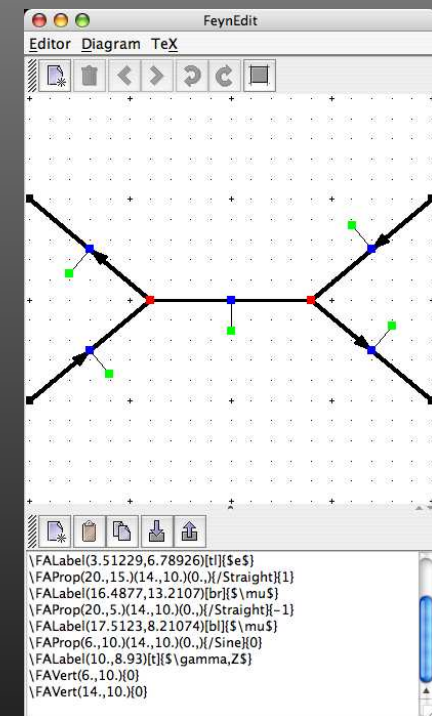
Editing Feynman Diagrams

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.**

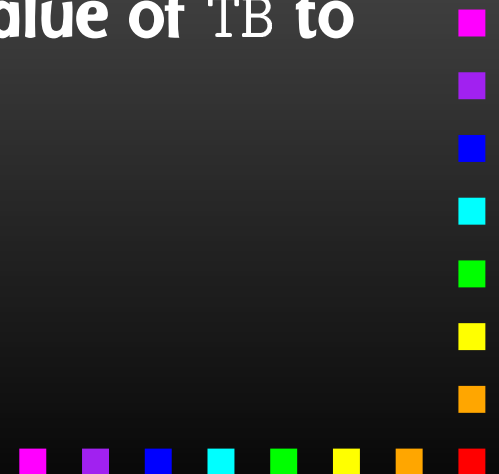
For example, the line that prints a parameter in the stand-alone code is

```
#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 $\sqrt{s} = \text{sqrtS}$:

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

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

```
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. $\{2.30801 \cdot 10^{-21}, 0.\} = \{\Delta\sigma_{\text{tot}}^{\text{tree}}, \Delta\sigma_{\text{tot}}^{\text{1-loop}}\}$.



Abbreviations

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

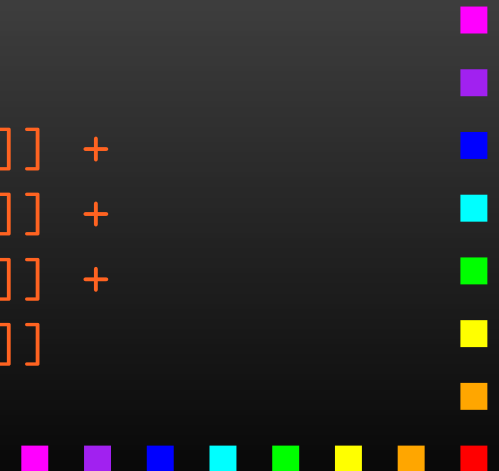
`AbbSum29 = Abb2 + Abb22 + Abb23 + Abb3`

`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**:

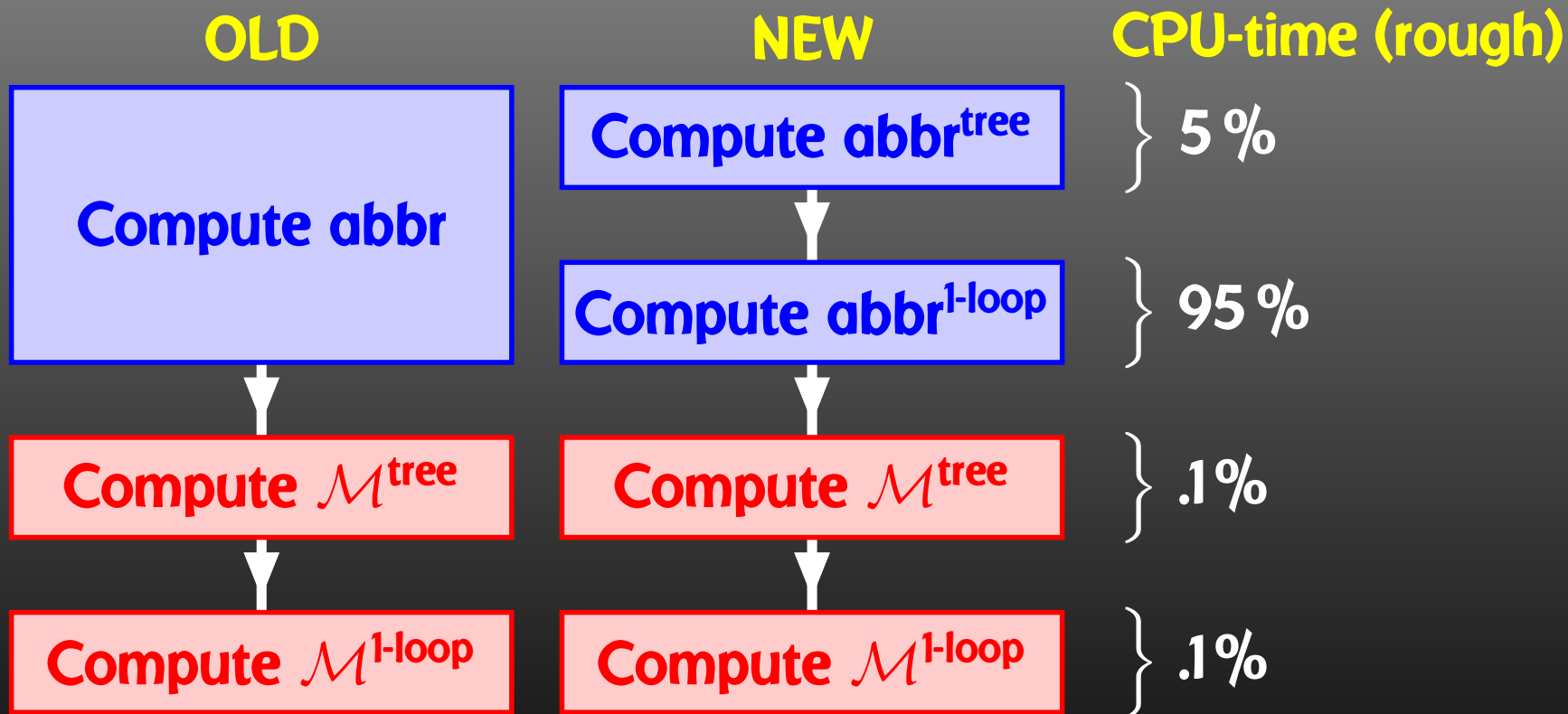
- **Abbreviations that depend on the helicities,**
- **Abbreviations that depend on angular variables,**
- **Abbreviations that depend only on \sqrt{s} .**

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.



Fierz Identities in 4D

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:

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

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:

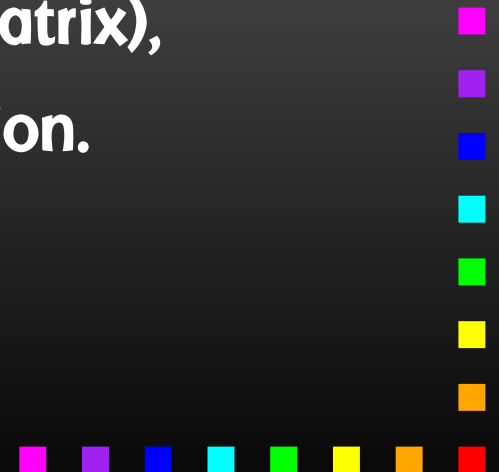
```
RenConst[ dMf1[t_, j1_] ] := MassRC[F[t, {j1}]]
RenConst[ dZfL1[t_, j1_, j2_] ] := FieldRC[F[t, {j1}], F[t, {j2}]] [[1]]
RenConst[ dZfR1[t_, j1_, j2_] ] := FieldRC[F[t, {j1}], F[t, {j2}]] [[2]]
RenConst[ dMZsq1 ] := MassRC[V[2]]
RenConst[ dMWsq1 ] := MassRC[V[3]]
RenConst[ dMHsq1 ] := MassRC[S[1]]
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

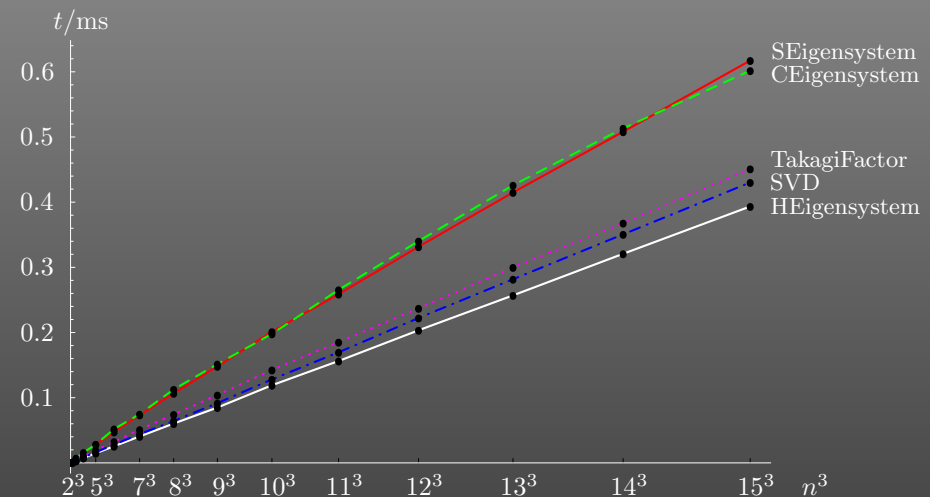
The Diag routines are based on the **Jacobi algorithm**.

This is conceptually simple but scales less favourably than e.g. the QR method.

Applicability range is thus **small to medium-size matrices**.

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

Timings on an AMD X2-5000:



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>.

