UG-FT-87/98
KA-TP-7-1998
hep-ph/9807565

Automated One-Loop Calculations in four and D dimensions

T. Hahn

Institut für Theoretische Physik, Universität Karlsruhe
D-76128 Karlsruhe, Germany

M. Pérez-Victoria

Dpto. de Física Teórica y del Cosmos, Universidad de Granada
E-18071 Granada, Spain

Abstract:
Two program packages are presented for evaluating one-loop amplitudes. They can work either in dimensional regularization or in constrained differential renormalization. The latter method is found at the one-loop level to be equivalent to regularization by dimensional reduction.

PACS: 07.05.Bx, 07.05.Wr, 11.15.Bt

Keywords: One-loop perturbative calculations, Computer algebra, Dimensional regularization, Dimensional reduction, Constrained differential renormalization.

1 Introduction

Little needs to be said nowadays about the importance of one-loop calculations. During the last years methods which (partially) automate these calculations have become available [1, 2, 3]. The automation is though nowhere as fully developed as for tree-level calculations [4, 5]. The reason is that loop calculations are in general more involved, and moreover a complete automation limits the possibilities to use a program beyond its designed scope.

In virtually all implementations of one-loop calculations on the computer, dimensional regularization [6, 7] has been employed for calculating renormalized amplitudes, and not without reason as it is a very elegant formalism that preserves most symmetries. However, dimensional regularization presents problems in chiral theories, related to the extension of γ_5 to arbitrary dimensions [8, 9, 10, 11]. In the Standard Model, a naive scheme—anticommuting γ_5 without further modifications—is usually used, but the situation is not completely clear [12]. More problematic are supersymmetric theories, where a variant, regularization by dimensional reduction [13], is commonly used in spite of possible inconsistencies at higher loops [14]. On the other hand, a new method in 4 dimensions, constrained differential renormalization (CDR), has recently been developed [15, 16]. It is a version of differential renormalization [17] that preserves gauge invariance at least to one loop [15, 16, 18]. There are some hints that it also preserves supersymmetry [19]. At any rate, it is clearly convenient to have at hand alternative methods implemented for automated one-loop calculations.

We have implemented the three mentioned methods in two program packages, called *FormCalc* and *LoopTools*, for the evaluation of one-loop amplitudes. It turns out that CDR and dimensional reduction are equivalent at the one-loop level (at least in the cases where the latter is well-defined). This is a new result and will be discussed in the next section. Actually, once the CDR coordinate-space expressions and procedure are translated into momentum space, both methods can be set up in an identical way. Therefore, our programs have in fact only two different options: to perform calculations in dimensional regularization or in dimensional reduction / CDR. We shall refer to these two possibilities as calculating in D or 4 dimensions, respectively.

Our programs do not fully automate one-loop calculations to the extent that they directly produce cross-sections from the process specification. Instead, *FormCalc* produces output which can easily be evaluated further, e.g. numerically in a Fortran program. One big advantage of *FormCalc* is that it leaves the user with a *Mathematica* expression which is considerably easier to modify than Fortran code. *LoopTools*, on the other hand, supplies all the functions needed for the numerical evaluation of the *FormCalc* output. We supply also a demonstration program to show how this numerical evaluation can be done.

The paper is organized as follows: In Section 2 we outline the regularization methods we have used in the programs, and discuss the equivalence between dimensional reduction and CDR. In Section 3 the implementation of the program packages is explained, where Section 3.1 describes the analytical and Section 3.2 the numerical part. In Section 4 we illustrate for the example of elastic Z–Z scattering the individual steps of a one-loop calculation using our packages. Section 5 lists the computer requirements needed to compile and run the packages, and their availability. Finally, an appendix collects the functions implemented for numerical evaluation.

2 Calculations in 4 and D dimensions

Dimensional regularization has proved to be the most convenient method for computing quantum corrections in gauge theories. The definition of the regularized expressions has two parts: analytic continuation of momenta (and other four-vectors) in the number of dimensions, D, and an extension to D dimensions of the Lorentz covariants (γ_μ, $g_{\mu\nu}$, etc.). The second part is achieved by treating the covariants as formal objects obeying certain algebraic identities [6]. The dimensionally regularized Feynman graphs are meromorphic functions of the complex parameter D, and the poles at $D = 4$ can consistently be subtracted to all orders in perturbation theory. Such a minimal subtraction scheme (MS) defines a renormalization fulfilling the usual requirements of a quantum field theory: causality, unitarity, field equations, Ward identities, etc. [6, 10]. Problems only appear for identities that depend on the 4-dimensional nature of the objects involved. This is the case for relations using the Levi-Civita tensor. In particular, the extension of γ_5 to D dimensions is problematic. The consistent prescriptions of Refs. [6, 8, 10] lead to spurious anomalies that have to be corrected with finite counterterms. We follow instead Ref. [12] and work with an anticommuting γ_5. We also maintain the usual relations for traces, such as $\mathrm{Tr}(\gamma_5\gamma_\mu\gamma_\nu\gamma_\rho\gamma_\sigma) = \varepsilon_{\mu\nu\rho\sigma}\mathrm{Tr}\mathbb{1}$. Even though they can be incompatible with an anticommuting γ_5 in D dimensions, the resulting ambiguities are expected to cancel in non-anomalous theories [12, 20]. For one-loop calculations one can proceed in the following manner (see Ref. [21] for details):

1. Calculate traces and simplify the Dirac algebra in D dimensions.
2. Write everything in terms of scalar and tensor integrals in D dimensions.
3. Decompose tensor integrals into Lorentz-covariant tensors constructed from the external momenta and the metric tensor. One is left with scalar integrals as coefficients of these tensors.
4. Calculate the scalar integrals and tensor coefficients in D dimensions, expand the whole diagram in $\varepsilon = D-4$, subtract the $\frac{1}{\varepsilon}$ poles and take the limit $\varepsilon \to 0$. Computer algebraically it is convenient to do this in two steps:
   (a) Add local terms for products of D times a divergent integral: $DI \to 4I + c$, with I the integral and c the coefficient of its $\frac{1}{\varepsilon}$ pole . Then, make $D = 4$ outside the integrals.
   (b) Calculate the scalar integrals and tensor coefficients in D dimensions, expand them up to order ε^0, and subtract their poles.

A modified version of dimensional regularization, designed to preserve supersymmetry and gauge invariance, was proposed by Siegel [13]. It is based on ***dimensional reduction*** from 4 to D dimensions: while the integration momenta are D-dimensional, as in usual dimensional regularization, all other tensors and spinors are kept 4-dimensional. Therefore one works with two kinds of objects: D-dimensional and 4-dimensional ones. For the validity of the field equations and gauge invariance, one must impose the identity

$$g_{\mu\nu}\hat{g}^{\nu\rho} = \hat{g}_\mu{}^\rho\,, \tag{1}$$

where $g_{\mu\nu}$ is the 4-dimensional metric ($g^\mu{}_\mu = 4$) and $\hat{g}_{\mu\nu}$ the D-dimensional one ($\hat{g}^\mu{}_\mu = D$). The Dirac algebra, including γ_5, is performed in 4 dimensions. Unfortunately, regularization by dimensional reduction is known to be inconsistent [14]. Nevertheless the inconsistencies arise at higher orders and the method has successfully been applied to many calculations in supersymmetric theories.[1] At one loop, the procedure described for dimensional regularization can be followed. The only differences are that the Dirac algebra is 4-dimensional and that a 4-dimensional metric tensor and Eq. (1) have to be introduced. Notice that $\hat{g}_{\mu\nu}$ only appears in the decomposition of the tensor integrals. These (and the tensor coefficients) are identical in dimensional regularization and dimensional reduction.

Finally, ***differential renormalization*** is a method of regularization and renormalization in coordinate space that cures UV divergences by substituting badly-behaved expressions by derivatives of well-behaved ones. The method has proved to be quite simple and convenient in a number of applications. A symmetric procedure of differential renormalization preserving gauge invariance at the one-loop level has recently been proposed. This so-called constrained differential renormalization proceeds in two steps:

1. The expressions read from Feynman diagrams are written in terms of a complete set of (singular) *basic functions*. The basic functions are products of propagators with derivatives acting on one of them. Functions are treated differently depending on whether their indices are self-contracted or not. All algebra is performed strictly in 4 dimensions.

2. These basic functions are substituted by their *renormalized* expressions. The renormalization of the singular basic functions has previously been fixed once and for all, such that a set of simple rules is respected (these rules are just natural extensions of mathematical identities among tempered distributions) [16]. In particular, the rules imply that renormalization does not commute with index contraction.

Although differential renormalization naturally works in coordinate space, it is possible to perform the reduction of Step 1 in momentum space and use the Fourier transforms of the renormalized basic functions in Step 2, which correspond to the tensor integrals in the dimensional methods. Expressions and manipulations in coordinate space can easily be translated into momentum space (see Appendix B of Ref. [16]).

As mentioned in the introduction, it turns out that CDR and dimensional reduction are equivalent methods for one-loop calculations. The equivalence can easily be understood once CDR has been translated into momentum space. First, the minimally subtracted D-dimensional tensor integrals are identical in the limit $D \to 4$ to the Fourier transforms of the corresponding renormalized basic functions of CDR (which we shall also call tensor integrals in the following), up to a redefinition of the renormalization scale.[2] The reason is that one-loop integrals in D dimensions satisfy the relations imposed by the CDR rules. Hence, a possible discrepancy can only arise in the initial conditions (namely, the renormalized value of the scalar one- and two-point functions), but this can be taken care of by the freedom

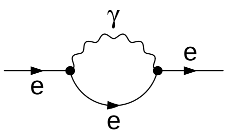
[1] Dimensional reduction can also be used in non-supersymmetric theories, but more care is required in order not to break unitarity [23].

[2] We have explicitly checked that the CDR tensor integrals in Appendix B of Ref. [16] coincide with the tensor integrals in dimensional reduction in the $\overline{\mathrm{MS}}$ scheme if the CDR renormalization scale $\bar{M}$ is redefined as $\log \bar{M}^2 = \log \mu^2 + 2$ where μ is the renormalization scale in the dimensional method.

to choose the renormalization scale. Second, all the algebra outside tensor integrals is considered 4-dimensional in both schemes. Third, the distinction of tensor integrals with contracted and uncontracted indices in CDR is imitated in dimensional reduction by Eq. (1) and the addition of local terms in Step 4a. To illustrate this, consider the renormalized tensor integral $C_{\mu\nu}$ (see Ref. [21] for its definition), which is the same in both methods (up to the freedom to choose the renormalization scheme). In CDR the tensor integral $C^{\mu}{}_{\mu}$ is considered independently; its renormalized value differs from $g^{\mu\nu}$ times the renormalized $C_{\mu\nu}$ by a local term [15]. In dimensional reduction we have

$$
\begin{aligned}
g^{\mu\nu} C_{\mu\nu} &= g^{\mu\nu} \left(\hat{g}_{\mu\nu} C_{00} + \sum_{i,j=1}^{2} p_{i\mu} p_{j\nu} C_{ij} \right) = \hat{g}^{\mu}{}_{\mu} C_{00} + \sum_{i,j=1}^{2} (p_i p_j) C_{ij} \\
&= D C_{00} + \sum_{i,j=1}^{2} (p_i p_j) C_{ij} = 4 C_{00} - \frac{1}{2} + \sum_{i,j=1}^{2} (p_i p_j) C_{ij} \,.
\end{aligned}
\tag{2}
$$

C_{00} and C_{ij} coincide in both methods. The extra local term, $-1/2$, is precisely what is needed to obtain $C^{\mu}{}_{\mu}$ from $g^{\mu\nu} C_{\mu\nu}$ in CDR [15, 16]. That the same occurs in the general case follows from the fact that in dimensional reduction one can contract the 4-dimensional metric with D-dimensional integration momenta before performing the integrals, and the resulting contracted tensor integrals also satisfy the CDR relations. Notice that conventional dimensional regularization introduces extra D's coming from $\hat{g}^{\mu}{}_{\mu}$'s outside the tensor integrals. Also, the Feynman rules in dimensional regularization contain D in some theories. Hence it can render different results, and not only in intermediate steps. A simple example where the results in D and 4 dimensions differ is the electron self-energy in QED:



$$
\stackrel{\text{dim.reg.}}{=} -\frac{e^2}{16\pi^2} \left[4 m_{\mathrm{e}} B_0(k^2, m_{\mathrm{e}}^2, 0) + 2 \not{k} B_1(k^2, m_{\mathrm{e}}^2, 0) + \not{k} - 2 m_{\mathrm{e}} \right] , \tag{3}
$$

$$
\stackrel{\text{CDR, dim.red.}}{=} -\frac{e^2}{16\pi^2} \left[4 m_{\mathrm{e}} B_0(k^2, m_{\mathrm{e}}^2, 0) + 2 \not{k} B_1(k^2, m_{\mathrm{e}}^2, 0) \right] . \tag{4}
$$

The functions B_0 and B_1 are defined in App. A.

Since dimensional reduction and CDR are equivalent at the one-loop level, one only needs a single implementation for the two methods. We have decided to use the CDR idea and add local terms when there are self-contracted indices after all the simplifications have been performed. This allows to work completely in 4 dimensions. Of course, the dimensionally regularized scalar functions and tensor coefficients are also used in this case.

3 Implementation of the program packages

The evaluation of one-loop diagrams using our packages proceeds in two steps:

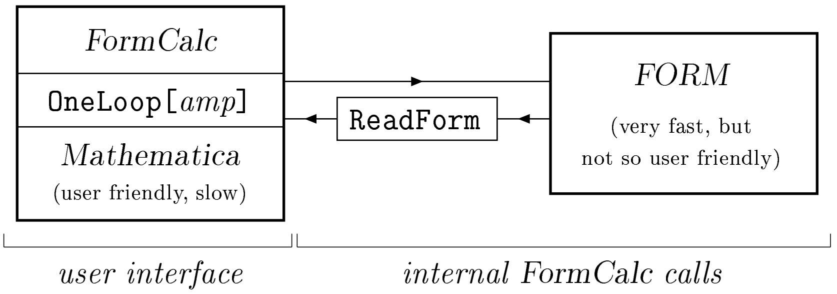
In *FormCalc* the symbolic expressions for the diagrams as obtained from *FeynArts* [24] are simplified algebraically such that the output can be used almost directly in a numerical program, e.g. in Fortran or C++.

LoopTools is the second of the packages and supplies the actual numerical implementations of the one-loop functions needed for programs made from the *FormCalc* output. It is based on the reliable package *FF* [25] and provides in addition to the scalar integrals of *FF* also the tensor coefficients in the conventions of Ref. [21]. *LoopTools* offers three interfaces: Fortran, C++, and *Mathematica*, so most programming tastes should be served.

3.1 *FormCalc*

FormCalc is a *Mathematica*-based program to calculate one-loop Feynman diagrams in either CDR or dimensional regularization. The program reads input from *FeynArts* and returns output in a way well suited for further numerical (or analytical) evaluation. *FormCalc* was designed to handle large numbers of diagrams, for example running the complete $W^+W^- \to W^+W^-$ amplitudes (about 1000 diagrams [26]) takes approximately 10 minutes on a Pentium PC.

The structure of *FormCalc* is rather simple: it prepares the symbolic expressions of the diagrams in an input file for *FORM*, runs *FORM*, and retrieves the results. This interaction is transparent to the user. *FormCalc* combines the speed of *FORM* with the powerful instruction set of *Mathematica* and the latter greatly facilitates further processing of the results. The following diagram shows schematically how *FormCalc* interacts with *FORM*:



The main function in *FormCalc* is `OneLoop` (the name is not strictly correct since it works also with tree graphs). It is used like this:

```
<< FormCalc.m;
$Dimension = 4;
amps = << myamps.m;
alldiags = OneLoop[amps]
```

The file `myamps.m` is assumed here to contain amplitudes generated by *FeynArts*. The dimension—`D` for dimensional regularization or `4` for dimensional reduction / CDR—is set with `$Dimension`.

Alternatively, if one wants to evaluate only a subset of diagrams,

```
somediags = OneLoop[ Pick[amps, {3, 5, {21, 29}}] ]
```

will calculate diagrams 3, 5, and 21 through 29, or for a single diagram it can be just

```
onediag = OneLoop[ amps[[9]] ]
```

Note that `OneLoop` needs no declarations of the kinematics of the underlying process; it uses the information *FeynArts* hands down. When amplitudes are loaded, *FormCalc* automatically puts the external particles on shell and for some common processes[3] introduces

[3]Currently processes with 4 or 5 external legs, i.e. $2 \to 2$, $2 \to 3$, $1 \to 3$, $1 \to 4$.

kinematical invariants (e.g. Mandelstam variables for a $2 \to 2$ process). The former can be disabled by setting `$OnShell = False` before loading the amplitudes.

For example, the QED electron self-energy in Eq. (3) was calculated with the following short program:

```
<< FormCalc.m;
$OnShell = False;
$Dimension = D;
OneLoop[<< electronSE.amp] //. Abbreviations[]
```

with the result

```
   EL^2 ME B0[Pair[k[1],k[1]], ME2, 0]
 - ----------------------------------- -
                4 Pi^2

     EL^2 B1[Pair[k[1],k[1]], ME2, 0] ga[k[1]]   EL^2 (-2 ME + ga[k[1]])
     ----------------------------------------- - -----------------------
                      8 Pi^2                             16 Pi^2
```

Similarly, Eq. (4) can be reproduced by putting `$Dimension = 4`. This results in

```
   EL^2 ME B0[Pair[k[1],k[1]], ME2, 0]
 - ----------------------------------- -
                4 Pi^2

     EL^2 B1[Pair[k[1],k[1]], ME2, 0] ga[k[1]]
     -----------------------------------------
                      8 Pi^2
```

Even more comprehensive than `OneLoop`, the function `ProcessFile` can process entire files. It splits the results up into a bosonic and a fermionic part; this is commonly needed for the numerical evaluation e.g. if one wants to sum over fermion generations. `ProcessFile` is invoked e.g. as

```
ProcessFile["vertex.amp", "results/vertex"]
```

It reads the *FeynArts* amplitudes from `vertex.amp` and produces `results/vertex.m` which contains the bosonic part and `results/vertexF.m` which contains the fermionic part.

`OneLoop` and `ProcessFile` return expressions where spinor chains, dot products of and Levi-Civita tensors contracted with polarization vectors have been collected and abbreviated. A term in such an expression may look like

```
C0i[cc1, MW2, S, MW2, MZ2, MW2, MW2] *
  (P12*S*(-8*a2*MW2 + 4*a2*MW2*S2 - 28*a2*CW^2*MW2*S2 +
        16*a2*CW^2*S*S2 + 4*a2*C2*MW2*SW^2) +
   O47*S*(-32*a2*CW^2*MW2*S2 + 8*a2*CW^2*S2*T + 8*a2*CW^2*S2*U) -
   P13*S*(-64*a2*CW^2*MW2*S2 + 16*a2*CW^2*S2*T + 16*a2*CW^2*S2*U) +
   O3*(-2*a2*MW2*T + a2*MW2*S2*T - 7*a2*CW^2*MW2*S2*T +
       4*a2*CW^2*S*S2*T + a2*C2*MW2*SW^2*T + 2*a2*MW2*U -
       a2*MW2*S2*U + 7*a2*CW^2*MW2*S2*U - 4*a2*CW^2*S*S2*U -
       a2*C2*MW2*SW^2*U))
```

The first line stands for the tensor coefficient function $C_1(M_W^2, s, M_W^2, M_Z^2, M_W^2, M_W^2)$ (see Appendix) which is multiplied with a linear combination of abbreviations like `O47` or `P12` with certain coefficients.

These coefficients contain the Mandelstam variables `S`, `T`, and `U` and some short-hands for parameters of the Standard Model, e.g. `a2` $= \alpha^2$ or `C4` $= \cos^{-4}\theta_W$. Replacing such commonly appearing factors by symbols makes expressions faster to handle for *Mathematica*. The coefficients are collected in a way optimized for numerical evaluation: for example, *Mathematica* will automatically add together the terms in `-2*a2*MW2*T + a2*MW2*S2*T - 7*a2*CW^2*MW2*S2*T` once the numerical values for the various parameters are substituted (i.e. the result is of the form `(a number)*T`), so it is not worth simplifying this expression further. If desired, *FormCalc* can nevertheless perform even these last simplifications to obtain the most compact analytical form.[4]

The abbreviations like `O47` or `P12` are introduced automatically and can significantly reduce the size of an amplitude. Consider `P13` in the excerpt of code above. It is composed of

```
P13 = O11 - O12 - O13 + O14 + O20
```

The `O`nn are in turn made up of

```
O11 = e23*k13*k43
O12 = e13*k23*k43
O13 = e13*k24*k43
O14 = e12*k31*k43
O20 = e23*k14*k43
```

and the `e`nn and `k`nn are defined as

```
e12 = Pair[e[1], e[2]]
e13 = Pair[e[1], e[3]]
k14 = Pair[e[1], k[4]]
...
```

`Pair[`a`, `b`]` denotes the scalar product of two four-vectors, and `e[`i`]` and `k[`i`]` the polarization vectors and momenta, respectively.

To get an idea of how advantageous the introduction of abbreviations can be, it is useful to compare the `LeafCount` of the expressions in *Mathematica*. The leaf count gives a measure of the size of an expression, more precisely it counts the number of subexpressions or "leaves" on the expression tree. `P13` has a leaf count of 1 since it is just a plain symbol. In comparison, its fully expanded contents have a leaf count of 113.

The definitions of the abbreviations can be retrieved by `Abbreviations[]` which returns a list of rules such that

```
result //. Abbreviations[]
```

gives the full, unabbreviated expression. It is of course necessary to save the abbreviations before terminating a session, e.g. with

```
Abbreviations[] >> abbr
```

[4] *FormCalc* wraps two functions, `o1` and `o2`, around coefficients at different levels in the amplitude. The default setting `o1 = o2 = Identity` can be changed e.g. to `o1 = o2 = Simplify`.

3.2 *LoopTools*

LoopTools supplies the functions needed for the numerical evaluation of the code produced by *FormCalc*. *LoopTools* follows the conventions of [21]. The complete reference to the *LoopTools* functions can be found in Appendix A.

LoopTools consists of three parts: a Fortran library, a C++ library, and a MathLink executable that can be used with *Mathematica* directly.

3.2.1 Using *LoopTools* with Fortran

To use the *LoopTools* functions in a Fortran program, the two files `tools.F` and `tools.h` must be included. `tools.F` contains actual Fortran code and should be included only once per program. `tools.h` contains the declarations and common blocks and must be included in every function or subroutine in which the *LoopTools* functions are used.

Before using any *LoopTools* function, the subroutine `bcaini` must be called. At the end of the calculation `bcaexi` may be called to obtain a summary of errors.

A very simple program would for instance be

```
#include "tools.F"

        program simple_program
#include "tools.h"

        call bcaini
        print *, B0(1000D0, 50D0, 80D0)
        call bcaexi
        end
```

Note that, as for all preprocessor commands, the `#` must stand at the beginning of the line.

Several default values can be superseded by defining preprocessor variables before including `tools.F`, e.g. the renormalization scale can be changed with `#define MUDIM ...` or the IR regulator mass can be changed with `#define LAMBDA ...`; some more technical options are described in the manual [27].

3.2.2 Using *LoopTools* with C++

To use the *LoopTools* functions in a C++ program, the file `ctools.h` must be included. Similar to the Fortran case, before making the first call to any *LoopTools* function, `bcaini()` must be called and at the end `bcaexi()` may be called to get a summary of errors.

In C++, the same simple program looks like

```
#include <fstream.h>
#include "ctools.h"

main()
{
  bcaini();
  cout << B0(1000., 50., 80.) << endl;
  bcaexi();
}
```

The renormalization scale and the IR regulator mass can be changed with the functions `set_mudim` and `set_lambda`, respectively.

3.2.3 Using *LoopTools* with *Mathematica*

The *Mathematica* interface is probably the simplest to use:

```
In[1]:= Install["bca"]

Out[1]= LinkObject[bca, 1, 1]

In[2]:= B0[1000, 50, 80]

Out[2]= -4.40593 + 2.70414 I
```

One-loop functions containing non-numeric arguments (e.g. `B0[1000, MW2, MW2]`) remain unevaluated.
Again, the renormalization scale and the IR regulator mass may be changed with the functions `Mudim` and `Lambda`, respectively.

4 An application to the Standard Model

We demonstrate in this section the basic steps of a real one-loop calculation using our packages. For an example, we choose the process ZZ $\to$ ZZ in the electroweak Standard Model [28].
A one-loop calculation generally includes

Step	Program	typical CPU time
1. Generate Diagrams	*FeynArts*	3 min.
2. Simplify analytically	*FormCalc*	10 min.
3. Produce Fortran code	(*NumPrep*)	3 min.
4. Compile with driver program	(`num.F`)	7 min.

The quoted execution times are for the full one-loop ZZ $\to$ ZZ calculation (about 500 diagrams) as run on a standard Pentium PC under Linux.

The numerical evaluation of the *FormCalc* output is not fully automated. The reason is that at one-loop level already many features appear which must be treated differently from process to process. The two programs *NumPrep* and `num.F` are supplied with the demo code to show how the numerical evaluation can be done. Although both were designed for $2 \to 2$ gauge-boson scattering processes, it should not be too difficult to adapt them at least to other $2 \to 2$ processes. (Basically, one has to supply the kinematics in `num.F`.)

In the following we present only a brief but characteristic excerpt of code for each step. The complete demo code is contained in the online distribution of *FormCalc*.

Step 1: Here the generation of the self-energy diagrams is demonstrated:

```
<< FeynArts.m

tops = CreateTopologies[ 1, 2 -> 2,              (* 1 loop, 2 -> 2 *)
  ExcludeTopologies ->                 (* create only self-energies *)
    {Tadpoles, WFCorrections, Triangles, AllBoxes} ]

inss = InsertFields[ tops,
  {V[2], V[2]} -> {V[2], V[2]},                          (* V[2] = Z *)
  Model -> "SM", InsertionLevel -> {Particles},
  Restrictions -> {NoGeneration2, NoGeneration3} ]

amps = CreateFeynAmp[inss];
ToFA1Conventions[amps] >> zzzz.self.amp
```

This code excerpt shows a typical application of *FeynArts*: create the topologies, insert fields into them (`V[2]` is a Z boson in *FeynArts* lingo), and create the amplitudes, i.e. apply the Feynman rules. The function `ToFA1Conventions` converts the full *FeynArts* 2.2 format back into the simpler *FeynArts* 1 format which is needed by *FormCalc*.

Step 2: The so-created self-energy diagrams are then simplified with *FormCalc*. Calculating in D and 4 dimensions yields equivalent results in the case of ZZ $\to$ ZZ, hence we have omitted the explicit definition of `$Dimension` here (the default is `D`).

```
<< FormCalc.m

ProcessFile["zzzz.self.amp", "self"];
Abbreviations[] >> abbr
```

Steps 3 and 4: The *FormCalc* results need to be converted into a Fortran program. (The numerical evaluation could, in principle, be done in *Mathematica* directly, but this becomes very slow for large amplitudes.) The simplest way to do this in *Mathematica* is something like `FortranForm[result] >> file.f`. With our demo code we supply a much more sophisticated program called *NumPrep* which goes well beyond simple translation into Fortran code. For example, it groups the one-loop integrals into angle-dependent and -independent integrals, so that the latter need to be calculated only once e.g. when integrating over the angle at a fixed energy.

NumPrep is too complex to describe here in detail. Instead, we give an example of how the final Fortran code looks like:

```
#include <defs.h>
        double complex function self()
        implicit logical (a-s,u-z)
        implicit double complex (t)
#include <vars.h>
        self = reso**2*(-5.747766329670332791488109665178O*ab61*O3 -
     -     3.696762828539980268636174052200372*ab62*O3 +
     -     ab59*(14.78705131415992107454469620880148 9 +
     -        0.0002222880990366838073437419146881997 3*MH2)*O3 +
     -     ab58*(22.99106531868133116595243866071195 1 +
     -        0.00044457619807336761468748382937639 95*MH2)*O3 +
     -     0.000666864297110051422031225744064599 2*ab60*MH2*O3 +
     -     0.002000592891330154266093677232193797 6*ab15*MH2**2*
     -      O3 - ab14*O3*(-215176.5845492641874055729913420395 0 -
     -        0.0002222880990366838073437419146881997 3*MH2**2 +
     -        7.393525657079960537272348104400745*S) - ...
```

Admittedly, this code looks ugly to the human eye, but note that it is highly optimized for numerical evaluation: apart from the `O3` which is one of the abbreviations already introduced by *FormCalc*, the one-loop integrals have been replaced by variables (e.g. `ab`nn for A and B functions). Also, the possibly resonant Higgs propagator $1/(s-M_{\mathrm{H}}^2)$ has been replaced by the variable `reso` so that it can be treated more easily in Fortran. These are just examples of what one can do with the *FormCalc* output.

Of course, *NumPrep* produces also the code to calculate the abbreviations and a `Makefile` to conveniently compile the code. The Fortran code produced by *NumPrep* needs in addition a driver program which supplies it with the necessary parameters, kinematics, etc. This driver program is called `num.F` and is included in the demo.

5 Computer requirements and availability

FormCalc and *LoopTools* should compile and run without change on any Unix-based platform. In particular, they have been tested on DEC Alpha, HP 9000, and Linux. *FormCalc* needs *Mathematica* 2.2 or above including the MathLink compiler (`mcc`) and *FORM* 2 or above.[5] *LoopTools* requires a Fortran-77 compiler, the GNU make utility, and the GNU C and C++ compilers (`gcc`, `g++`).

The programs *FormCalc* and *LoopTools* can be obtained via WWW from
`http://www-itp.physik.uni-karlsruhe.de/formcalc` and
`http://www-itp.physik.uni-karlsruhe.de/looptools`, respectively.

[5] *FormCalc* runs also with *FORM* 1, but cannot fully simplify spinor chains with external fermions then.

The packages contain a comprehensive manual giving installation instructions and a detailed description of every function.

FeynArts is available from

`ftp://ftp.physik.uni-wuerzburg.de/pub/hep/index.html`.

6 Acknowledgements

We thank Ansgar Denner for important discussions. T.H. thanks the University of Granada for the hospitality during preparation of this work. This work has been partially supported by DFG, under contract number Ku 502/8–1, and by CICYT, under contract number AEN96-1672, Junta de Andalucia, FQM101 and Ministerio de Educacion y Cultura.

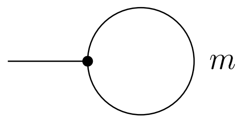
A Reference of the *LoopTools* functions

A.1 One-point functions

Function call	Description
`A0(ms)`	one-point function

The real argument `ms` (`double precision` in Fortran) is the mass squared:

$$\mathtt{ms} = m^2 \tag{5}$$

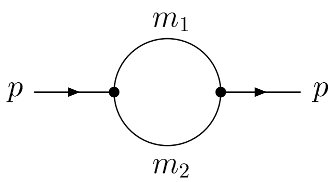


A.2 Two-point functions

Function call	Description
`B0(ps, m1s, m2s)`	scalar two-point function
`B1(ps, m1s, m2s)`	coefficient of p_μ
`B00(ps, m1s, m2s)`	coefficient of $g_{\mu\nu}$
`B11(ps, m1s, m2s)`	coefficient of $p_\mu p_\nu$

All arguments are real (`double precision` in Fortran) and are related to the momenta and masses as follows:

$$\begin{aligned} \mathtt{ps} &= p^2 \\ \mathtt{m1s} &= m_1^2 \\ \mathtt{m2s} &= m_2^2 \end{aligned} \tag{6}$$



A.3 Derivatives of two-point functions

Function call	Description
`DB0(ps, m1s, m2s)`	derivative of `B0`
`DB1(ps, m1s, m2s)`	derivative of `B1`
`DB00(ps, m1s, m2s)`	derivative of `B00`
`DB11(ps, m1s, m2s)`	derivative of `B11`

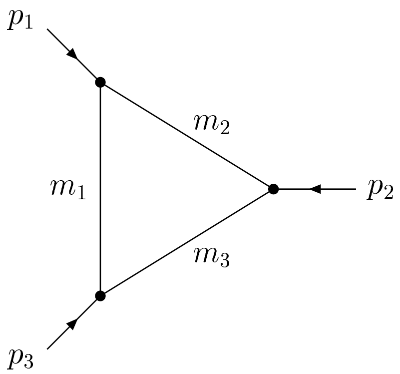
All derivatives are with respect to the momentum squared. The arguments are as in the case of the two-point functions.

A.4 Three-point functions

Function call	Description
`C0(p1s, p2s, p1p2s, m1s, m2s, m3s)`	scalar three-point function
`C0i(id, p1s, p2s, p1p2s, m1s, m2s, m3s)`	three-point tensor coefficients

Except for the `id` all arguments are real (`double precision` in Fortran) and are related to the momenta and masses as follows:

$$\begin{aligned} \texttt{p1s} &= p_1^2 \\ \texttt{p2s} &= p_2^2 \\ \texttt{p1p2s} &= (p_1 + p_2)^2 \\ \texttt{m1s} &= m_1^2 \\ \texttt{m2s} &= m_2^2 \\ \texttt{m3s} &= m_3^2 \end{aligned} \tag{7}$$



`C0i` is a generic function for all three-point tensor coefficients. A specific coefficient is selected with the `id` argument:

$$\begin{aligned} \texttt{C0i(cc0, ...)} &= C_0(\ldots) \\ \texttt{C0i(cc00, ...)} &= C_{00}(\ldots) \\ \texttt{C0i(cc112, ...)} &= C_{112}(\ldots) \quad \text{etc.} \end{aligned} \tag{8}$$

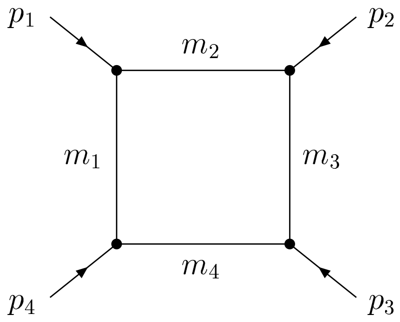
Since the indices are symmetric, the identifiers are assumed to be ordered, i.e. there is only `cc122` but not `cc212`.

A.5 Four-point functions

Function call	Description
`D0(p1s, p2s, p3s, p4s, p1p2s, p2p3s, m1s, m2s, m3s, m4s)`	scalar four-point function
`D0i(id, p1s, p2s, p3s, p4s, p1p2s, p2p3s, m1s, m2s, m3s, m4s)`	four-point tensor coefficients

Except for the `id` all arguments are real (`double precision` in Fortran) and are related to the momenta and masses as follows:

$$\begin{aligned}
\texttt{p1s} &= p_1^2 \\
\texttt{p2s} &= p_2^2 \\
\texttt{p3s} &= p_3^2 \\
\texttt{p4s} &= p_4^2 \\
\texttt{p1p2s} &= (p_1+p_2)^2 \\
\texttt{p2p3s} &= (p_2+p_3)^2 \\
\texttt{m1s} &= m_1^2 \\
\texttt{m2s} &= m_2^2 \\
\texttt{m3s} &= m_3^2 \\
\texttt{m3s} &= m_4^2
\end{aligned} \tag{9}$$



`D0i` is a generic function for all four-point tensor coefficients. A specific coefficient is selected with the `id` argument:

$$\begin{aligned}
\texttt{D0i(dd0, ...)} &= D_0(\dots) \\
\texttt{D0i(dd00, ...)} &= D_{00}(\dots) \\
\texttt{D0i(dd1223, ...)} &= D_{1223}(\dots) \quad \text{etc.}
\end{aligned} \tag{10}$$

Again, since the indices are symmetric, the identifiers are assumed to be ordered, i.e. there is only `dd1223` but not `dd3212`.

References

[1] R. Mertig, M. Böhm, and A. Denner, *Comp. Phys. Commun.* **64** (1991) 345;
R. Mertig, *Guide to FeynCalc 1.0*, Universität Würzburg (1992).

[2] T. Ishikawa *et al.*, KEK report 92-19 (1993).

[3] L. Brücher, *Nucl. Instr. Meth.* **A389** (1997) 327.

[4] E. Boos *et al.*, in: *New computing techniques in physics research II*, ed. D. Perret-Gallix, World Scientific, Singapore (1993).

[5] T. Stelzer and W. Long, *Comp. Phys. Commun.* **81** (1994) 357.

[6] G. 't Hooft and M. Veltman, *Nucl. Phys.* **B44** (1972) 189.

[7] C.G. Bollini and J. Giambiagi, *Nuovo Cim.* **12 B** (1972) 20;
J.F. Ashmore, *Nuovo Cim. Lett.* **4** (1972) 289;
G.M. Cicuta and E. Montaldi, *Nuovo Cim. Lett.* **4** (1972) 329.

[8] D.A. Akyeampong and R. Delbourgo, *Nuovo Cim* **17A** (1973) 578.

[9] W.A Bardeen, R. Gastmans and B. Lautrup, *Nucl. Phys.* **B46** (1972) 319.

[10] P. Breitenlohner and D. Maison, *Comm. Math. Phys.* **52** (1977) 11.

[11] G. Bonneau, *Phys. Lett.* **B96** (1980) 147.

[12] M. Chanowitz, M. Furman, and I. Hinchliffe, *Nucl. Phys.* **B159** (1979) 225.

[13] W. Siegel, *Phys. Lett.* **B84** (1979) 193.

[14] W. Siegel, *Phys. Lett.* **B94** (1980) 37.

[15] F. del Aguila, A. Culatti, R. Muñoz Tapia, and M. Pérez-Victoria, *Phys. Lett.* **B419** (1998) 263; F. del Aguila and M. Pérez-Victoria, *Acta Phys. Polon.* **B28** (1997) 2279.

[16] F. del Aguila, A. Culatti, R. Muñoz Tapia, and M. Pérez-Victoria, MIT-CTP-2705, UG-FT-86/98, hep-ph/9806451.

[17] D.Z. Freedman, K. Johnson, and J.I. Latorre, *Nucl. Phys.* **B371** (1992) 353.

[18] M. Pérez-Victoria, UG-FT-89/98.

[19] P.E. Haagensen, *Mod. Phys. Lett.* **A7** (1992) 893;
Y.S. Song, Ph.D. thesis;
F. del Aguila, A. Culatti, R. Muñoz Tapia, and M. Pérez-Victoria, *Nucl. Phys.* **B504** (1997) 532;
F. del Aguila, A. Culatti, R. Muñoz Tapia, and M. Pérez-Victoria, International Workshop on Quantum Effects in MSSM, Universitat Autònoma de Barcelona, September 1997, hep-ph/9711474.

[20] C. Schubert, *Nucl. Phys.* **B323** (1989) 478;
R. Ferrari, A.L. Yaouanc, L. Oliver and J.C. Raynal, *Phys. Rev.* **D52** (1995) 3036.

[21] A. Denner, *Fortschr. Phys.* **41** (1993) 307.

[22] J.A.M. Vermaseren, *Symbolic Manipulation with FORM*, CAN, Amsterdam (1991).

[23] D.M. Capper, D.R.T. Jones and P. van Nieuwenhuizen,*Nucl. Phys.* **B167** (1980) 479;
I. Jack, D.R.T. Jones and K.L. Roberts, *Z. Phys.* **C62** (1994) 161; **C63** (1994) 151.

[24] J. Küblbeck, M. Böhm and A. Denner, *Comp. Phys. Commun.* **60** (1991) 165;
H. Eck and J. Küblbeck, *Guide to FeynArts 1.0*, Universität Würzburg (1992);
H. Eck, *FeynArts 2.0—A generic Feynman diagram generator*, Dissertation, Universität Würzburg (1995).

[25] G.J. van Oldenborgh and J.A.M. Vermaseren, *Z. Phys.* **C46** (1990) 425.

[26] A. Denner and T. Hahn, hep-ph/9711302, to appear in *Nucl. Phys.* **B** (1998) .

[27] T. Hahn, *FormCalc and LoopTools user's guide*, available at `http://www-itp.physik.uni-karlsruhe.de/looptools`.

[28] A. Denner, S. Dittmaier and T. Hahn, *Phys. Rev.* **D56** (1997) 117.