

CalcHEP v.3.9.2
Calculator for High Energy Physics
A package for an automatic evaluation of
Feynman diagrams, integration over
multi-particle phase space, and event
generation.

A.Pukhov, A.Belyaev, N.Christensen*

March 5, 2025

Reference:

A. Belyaev, N. D. Christensen, and A. Pukhov,
Comput. Phys. Commun. **184** (2013), 1729-1769,
doi:[10.1016/j.cpc.2013.01.014](https://doi.org/10.1016/j.cpc.2013.01.014), [arXiv:[1207.6082](https://arxiv.org/abs/1207.6082) [hep-ph]].

*calchep@googlegroups.com

Contents

1	Preface	6
1.1	Introduction	6
1.2	Acknowledgments	9
2	Installation procedure	11
2.1	CalcHEP Web Site	11
2.2	Calcchep License	11
2.3	How to get the code	11
2.4	Compilation procedure	12
2.5	Compilation for High Precision calculations	14
2.6	User installation and start of the CalcHEP session	14
2.7	Potential problems in compilation	15
3	Elements of the user interface	18
4	Menu system for symbolic calculation	28
4.1	Model Choice and Manipulation	28
4.2	Numerical Evaluations	31
4.3	Process Input	33
4.4	Squared Diagrams and Symbolic Calculation	35
4.5	Output of results and launching of the numerical calculation	39
4.6	Switches	40
5	Numerical session	41
5.1	Sketch of the menu system.	41
5.2	Bookkeeping	43
5.3	Parton Distribution Functions	43
5.4	QCD coupling	45
5.5	Breit-Wigner propagator	46
5.6	Kinematical Functions	48
5.7	User Defined Functions	50
5.8	Cuts	51
5.9	Kinematics	52
5.10	Regularization	52
5.11	Monte-Carlo simulation	53
5.12	Event Generation	55
5.13	Simpson Integration	57
5.14	Two Particle Decays	58

6	Collecting Subprocesses	60
6.1	Distribution Summation	60
6.2	Event Mixing and LHEF	60
6.3	N-tuples.	63
7	Batch Mode	64
7.1	Blind mode	65
7.2	Shell Scripts	67
7.3	Batch interface	71
7.3.1	Structure and keywords of the batch file	73
7.3.2	Example of the batch file	85
7.3.3	Monitoring of the calchep batch session	87
7.3.4	Results Storage	89
8	Particle Interaction Model Implementation	92
8.1	Independent Parameters	92
8.2	Dependent Parameters	93
8.3	Particles	95
8.4	Interaction Vertices	97
8.5	External functions and libraries.	100
8.6	Propagators	101
8.7	Ghost and Goldstone fields propagators	102
9	Tools for model implementation.	105
9.1	The SLHAplus package	105
9.2	Effective Higgs γ - γ and glu - glu interactions.	108
9.2.1	Construction of effective vertexes.	108
9.2.2	QCD corrections to $h\gamma\gamma$	109
9.2.3	QCD corrections for h - glu - glu	109
9.3	LanHEP: automatic generation of models.	110
9.4	FeynRules	112
10	CalcHEP as a generator of matrix elements for other packages.	113
10.1	Choosing of model	113
10.2	Setting of parameters and calculation of constraints.	113
10.3	Testing of particle contents.	114
10.4	Decay widths and branching fractions.	115
10.5	Compilation of new processes.	116
10.6	Calculation of matrix elements.	117

11 CalcHEP output for Reduce, Mathematica and Form	118
11.1 General structure	118
11.2 <i>Reduce</i> examples	120
11.3 <i>Mathematica</i> examples	121
11.4 Form	123
11.5 <i>Reduce</i> program	123
Appendix	128
A L^AT_EX output	128
B Self-check of the CalcHEP package	128
B.1 Check of the built-in symbolic calculator	129
B.2 Comparison of results produced in two different gauges	129
C Ghost fields and the squared diagram technique for the t'Hooft-Feynman gauge	130
C.1 The problem	130
C.2 Incoming and outgoing ghosts	131
C.3 Massless vector-particle case	133
C.4 Summation of ghost diagrams in CalcHEP	134
C.5 Gauge symmetry and cancellations	135
D Feynman rules in CalcHEP	135
D.1 Lorentz part of diagram	135
D.2 Color factor	137
D.3 Common factors.	138
E Examples of model realization.	139
E.1 Implementation of QCD Lagrangian	139
E.2 Neutrino as a Majorana fermion	142
E.3 Leptoquarks	144
F Color string basis.	145
G Distribution functions and beam spectra	146
G.1 Backscattered photon spectrum	146
G.2 Weizsaecker-Williams approximation	146
G.3 ISR and Beamstrahlung	147

H	PDT - Particle Distribution Tables in CalcHEP.	149
H.1	CTEQ and MRST parton distributions	149
H.2	Format of parton distribution tables.	150
I	Monte Carlo phase space integration	152
I.1	Parameterization of multi-particle phase space	152
I.1.1	Parameterization via decay scheme	152
I.1.2	Polar vectors	154
I.1.3	Smoothing	155
I.2	Adaptive Monte Carlo integration package <i>Vegas</i>	156
I.2.1	Importance sampling	157
I.2.2	Stratified sampling	157
I.3	Generation of events	158
I.4	Format of event file.	159
J	Table of exit codes	160

1 Preface

1.1 Introduction

CalcHEP [1] is a package for the automatic calculation of elementary particle collisions and decays in the lowest order of perturbation theory (the tree approximation). The main idea of CalcHEP is to provide an interactive environment where the user can pass from the Lagrangian to the final distributions effectively with a high level of automation.

Other packages created to solve similar problems are *GRACE* [2–4], *HELAS* [5], CompHEP [6, 7], *FeynArts/FormCalc* [8–10], *MADGRAPH* [11, 12], HELAC-PHEGAS [13–15], O’MEGA [16], WIHIZARD [17], and SHERPA [18, 19].

The interactive session of CalcHEP is graphical and menu-driven and guides the user through the calculation by breaking the calculation up into a series of steps. At each step, CalcHEP presents the user with the available options allowing him/her to control the details of their calculation in an intuitive way. Moreover, at each menu, contextual help is available which explains the details of the current choices.

The batch session of CalcHEP is controlled by a set of scripts which perform common tasks noninteractively. After initializing the desired calculation, the user runs one of the scripts in the background. In some cases, parallelization is also supported.

Many models of particle interactions have been implemented in CalcHEP. Among them are the minimal supersymmetric extension of the standard model (MSSM) [20, 21], the next to minimal supersymmetric extension of the standard model (NMSSM) [22], the cp violating minimal supersymmetric extension of the standard model (CPVMSSM) [23], Little Higgs Models [24], a Lepto-quark model [25], a Technicolor model [26], a Higgsless Model [27, 28], and models with extra-dimensions [29–34]. Some of these models are available on the CalcHEP website for the users convenience.

A new model of particle interactions is implemented into CalcHEP by writing a set of pure text model files which contain all the details of the model including the properties of its particles, parameters and vertices. Although it is possible to do this by hand (especially for simple models), new models are typically implemented using a dedicated implementation package such as LanHEP [35] or FeynRules [36] which automatize the process of calculating the Feynman rules and writing the CalcHEP model files. This can be especially important for models with a large number of new particles and complicated Lagrangians.

The CalcHEP package consists of three parts which perform the symbolic, numerical and batch calculations. The first two parts are written in the *C*

programming language. The symbolic part produces C codes for squared matrix elements which are then used in the numerical calculations. The last part is written in Perl. It calls the routines of the first two parts and collects the results to obtain final cross sections and events for multichannel processes typical of modern collider physics.

The symbolic session of CalcHEP enables the user to interactively:

- Load a new model of particle interactions.
- Modify an existing model of particle interactions.
- Check a model for syntax errors.
- Read parameter information from an SLHA file [37, 38].
- Calculate dependent parameters.
- Calculate decay widths and branching ratios.
- Choose between Feynman and unitary gauge.
- Select a collision or decay process by specifying the incoming and outgoing particles.
- Specify particle exclusions for the diagrams.
- Generate Feynman diagrams.
- Display Feynman diagrams.
- Generate L^AT_EX output for the diagrams.
- Remove specific diagrams from the calculation.
- Generate and display squared Feynman diagrams.
- Remove specific squared diagrams from the calculation.
- Calculate analytic expressions for squared diagrams using the built-in symbolic calculator.
- Export the resulting squared diagram expressions to *Reduce*, *Mathematica*, or *Form* for further symbolic manipulation.
- Generate optimized C code for squared matrix elements.

- Compile the generated code.
- Launch the resulting numerical session.
- Generate numerical libraries of squared matrix elements for other packages.

The numerical session of CalcHEP enables the user to interactively:

- Convolute the squared matrix element with structure functions and beam spectra. The *CTEQ* and MRST parton distribution functions are supported via a link with LHAPDF [39]. Additionally, ISR and Beamstrahlung spectra of electrons, the laser photon spectrum, and the Weizsäcker-Williams photon structure functions are available [40] for muons, electrons, and protons.
- Modify physical parameters such as total energy, coupling constants, and masses involved in the process.
- Set the polarization of incoming massless particles.
- Set the QCD scale for the evaluation of the QCD coupling constant (and optionally its running) and for the parton distribution functions.
- Automatically calculate particle widths, including both 1→2 and 1→3 decay processes.
- Apply various kinematical cuts.
- Define the kinematic scheme (phase-space parameterization) for effective Monte Carlo integration.
- Introduce phase-space mapping to smooth sharp peaks in the squared matrix element or structure functions.
- Perform a Monte Carlo phase-space integration using *Vegas*.
- Generate unweighted events.
- Display distributions for various kinematic variables.
- Create graphical and L^AT_EX output for histograms.
- Save histogram data to a file for further analysis and/or plotting using gnuplot, PAW, Root, or Python.

Most of the features of the symbolic and numerical sessions are supported in the `batch` session. The instructions for batch computation are written in a text file. The batch interface reads these instructions and executes them in the background noninteractively.

The batch session enables the user to:

- Perform symbolic and numerical calculations available in interactive mode.
- View the progress of calculations through a series of HTML pages.
- Run calculations in parallel on a multicore machine or a computer cluster.
- Combine events from multiple production and decay processes.
- Write the final events to an LHE file [41] for further processing.
- Perform parameter scans over multiple physical variables.

1.2 Acknowledgments

The CalcHEP package includes code written by the CompHEP group. We would like to thank V. Ilyin, D. Kovalenko, A. Kryukov, V. Edneral, and A. Semenov for granting permission to use their code in CalcHEP.

Over the past decade, much of the CalcHEP development has been driven by the micrOMEGAs package [42, 43], developed by A. Pukhov in collaboration with G. Bélanger, F. Boudjema, and A. Semenov. Many new models for CalcHEP were implemented as part of this collaboration. The work of A. Semenov in developing the LanHEP program has been crucial for CalcHEP. We are very grateful to G. Bélanger, F. Boudjema, and A. Semenov for their collaboration. This work was supported by PICS-397 of CNRS, *Calcul en physique des particules*.

We would also like to thank A. Datta and K. C. Kong for their numerous suggestions, discussions, and testing of CalcHEP, as well as Patrik Svantesson for his recent help with debugging.

A. Belyaev acknowledges support from the STFC Consolidated Grant No. ST/J000396/1, as well as partial support through the NExT Institute.

A. Pukhov’s work on CalcHEP has been supported by the Royal Society grant JP090598 (2010–2013) and the Diamond Jubilee International Visiting Fellowship of the University of Southampton (2015–2016).

N. Christensen has been supported by the US National Science Foundation under grant PHY-0354226, the LHC-TI initiative of the US National

Science Foundation under grant PHY-0705682, the PITTsburgh Particle Physics, Astrophysics, and Cosmology Center, and the US Department of Energy under grant DE-FG02-12ER41832.

2 Installation procedure

2.1 CalcHEP Web Site

The CalcHEP code and manual can be found at the following Web site:

<http://theory.sinp.msu.ru/~pukhov/calchep.html>

2.2 Calcchep License

Non-profit Use License Agreement

This Agreement is to be held between the Authors of the CalcHEP program and any Party which acquires the program. On acquiring the program the Party agrees to be bound by the terms of this Agreement.

1. This License entitles the Licensee (one person) and the Licensee's research group to obtain a copy of the source code of CalcHEP and to use the acquired program for academic research and education or other non-profit purposes within the research group; or, it entitles the Licensee (a company, organization or computing center) to install the program and allow access to the executable code to the members of the Licensee for academic research and education or other non-profit use.
2. No user or site will re-distribute the source code or executable code to a third party in the modified form. Any re-distribution must provide the current licence for users.
3. This License does not permit any commercial (profit-making or proprietary) use or re-licensing or re-distributions. Persons interested in a for-profit use should contact the Authors.
4. The Authors of CalcHEP do not guarantee that the program is free of errors or meets its specification and cannot be held responsible for loss or consequential damage as a result of using it.

2.3 How to get the code

If you agree with the license above, you may download the CalcHEP code from the CalcHEP web site. The current filename is

`calchep_3.9.2.tar.gz`

which corresponds to the current CalcHEP version 3.9.2.

The next step is to unpack this file by doing

```
gtar -xzf calchep_3.9.2.tar.gz
```

As a result, a directory named `calchep_3.9.2` will be created. Below we shall refer to this directory as `$CALCHEP`¹. This directory contains the following subdirectories:

- `c_sources/` which is used for the source codes of the CalcHEP package.
- `lib/` which is used for the libraries generated during the CalcHEP compilation.
- `bin/` which contains the CalcHEP executable scripts and binary files.
- `include/` which contains some header files.
- `pdTables/` which contains tables of partons distribution functions.
- `help/` which contains text files used in the interactive session for the contextual help.
- `utile/` which contains auxiliary routines which are described in the `utile/README` file.
- `models/` which contains two realizations of the Standard Model in CalcHEP's format, one with a full and another with a diagonal CKM matrix.
- `work/` which is used to initialize a directory for the users calculations.

2.4 Compilation procedure

In order to compile the CalcHEP source code you need a *C* compiler, the X11 graphics library and the X11 include files. The compilation is launched by running

```
gmake
```

from the `calchep_3.9.2` directory. CalcHEP Makefiles are written for *gmake*. If the *gmake* command is absent on the users computer then *make* should also work.

If the compiler is detected and the sources are compiled successfully you will see the message:

```
"CalcHEP is compiled successfully and can be started "
```

¹CalcHEP scripts automatically set the environment variable `CALCHEP` which contains the path to the CalcHEP root directory `$CALCHEP`.

Otherwise the corresponding error message is printed on screen. See Section (2.7) for a discussion of possible problems. The size of the installed package is approximately 5Mb. To clean all the files created during compilation and any files created in `calchep_3.9.2/work/`, the user can issue the command

```
gmake clean
```

This command asks whether the user would like to delete the `FlagsForSh` file which contains user modifiable compiler names and compiler flags. The user may choose to keep this file if he/she wishes to use it in subsequent compilations.

Compilation tuning. The CalcHEP compilation procedure consists of two steps. During the first step, CalcHEP looks for compilers and compiler flags. The results are written in the `FlagsForSh` file in the form of *bash* assignment instructions. For instance:

```
CC=gcc
CFLAGS="-g -signed_char -Wall"
```

If the resulting parameters satisfy the CalcHEP requirements then the file `FlagsForMake`, which contains the same assignments, but written in *make* format is created. This file is included in all `Makefile` files used during CalcHEP compilations. In this way, the user can tune the file `FlagsForSh` to fit his/her computing environment and recompile CalcHEP. The `FlagsForSh` file contains comments which explain the available parameters. The option to save the `FlagsForSh` file during the *gmake clean* procedure is implemented to save the users modifications, if desired, for the recompilation. The command

```
gmake flags
```

generates the `FlagsForMake` file and stops.

Both *C* and *Fortran* compilers are defined in `FlagsForSh`. The *Fortran* compiler is not used for CalcHEP compilation, but it can sometimes be required for compilation of programs used by CalcHEP interaction models. For example, the CalcHEP implementation of the MSSM needs either *SuSpect*, *Isajet* or *SoftSUSY* to calculate the particle spectrum. All of these programs require a *Fortran* compiler. So, although CalcHEP does not require a *Fortran* compiler, some problems are expected in some models in the absence of a *Fortran* compiler.

2.5 Compilation for High Precision calculations

By default CalcHEP uses the `double` numerical type to store the initial and intermediate parameter values and `double precision` functions to work with them. The user can, optionally, choose to compile CalcHEP for high precision calculations.

The `long double` type is part of the C99 standard and realized on all modern *C* compilers, however, one has to note that, usually, the `long double` type is implemented with 80 bit precision. In this case, calculations will be as fast as with the standard `double` type, but the increase in precision is not significant. To enable the `long double` type, the compiler option `-D_LONG_` needs to be added to the `FlagsForSh` file and CalcHEP needs to be recompiled. If CalcHEP has already been compiled, `gmake clean` needs to be run first, but `FlagsForSh` should be kept. If CalcHEP has not been compiled yet, `gmake flags` should be run first, in order to create `FlagsForSh`.

The Intel *C* compiler has a `_Quad` type (quadruple precision) for 128 bit real numbers. To use this type in CalcHEP calculations, the compiler option `-D_QUAD_` has to be added to `FlagsForSh`. One should note that, currently, only the Intel compilers support `_Quad` type and that the Intel compilers require further options. Here is an example of the `CFLAGS` line of `FlagsForSh` for the Intel compiler:

```
CFLAGS="-D_QUAD_ -fPIC -fsigned-char -Qoption,cpp,--extended_float_type"
```

To implement other numeric types, the user should edit the file `include/nType.h`.

2.6 User installation and start of the CalcHEP session

After compilation of the CalcHEP package, the user should install a work directory where they perform their calculations. This is created with the `mkUsrDir` script which takes as its (only) argument the directory name where the user would like to do his/her calculations.

```
./mkUsrDir <dir.Name>
```

The directory name can include path information as appropriate. Here is an example that creates a work directory named `work` in the user's home directory:

```
./mkUsrDir ~/work
```

The user can create several work directories for different calculations if they like. The `mkUsrDir` script will create the directory `<dir.Name>` and copy or link the following directories and files to it:

- `bin` is a symbolic link to the `$CALCHEP/bin` directory and contains all the scripts and binaries required for calculations with CalcHEP.

- **models** is where the particle interaction model files belong. It is initialized with the default models contained in `$CALCHEP/models`, but the user can add further models to this directory.
- **tmp** is where CalcHEP stores temporary files during symbolic calculations.
- **results** is where the output of the symbolic session is written. In particular, this is where the symbolic session creates the numerical code `n_calchep` for the user to perform their numerical calculations.
- **calchep.ini** is a text file which allows the user to specify his/her preferences for the graphical user interface. Among the options available in this file are the text font, whether to use color or black and white and whether to use a sound to signal certain events.
- **calchep** is the shell script which is normally used to start the symbolic session. It is invoked as `./calchep`.
- **calchep_batch** is the shell script which is normally used to start a batch session. It is invoked as `./calchep_batch <batch.file>` where `<batch.file>` is the name of the file which contains the batch instructions. An example batch file is stored in `$CALCHEP/utile/batch_file`. The user can copy it and modify as necessary for their calculation. When run, the batch program creates the following directories:
 - **Processes** is where the binaries for the individual processes are generated and stored and where the numerical calculations are performed.
 - **Events** is where the event files (in LHE format) and distributions are stored.
 - **html** is where a system of html files are stored that inform the user of the progress and results of the calculation. This directory also contains a rich set of help files which explain the details of how to use the batch system. These files can be opened in the users web browser. The file `html/index.html` contains links to all the other html files.

2.7 Potential problems in compilation

X11: The most frequent compilation problem is due to the absence of the X11 include files on the user's computer. Usually, these files are stored in the

directory `/usr/include/X11/`, but CalcHEP checks other locations as well. If the X11 header files are not found, CalcHEP still compiles, however, it only runs in non-interactive mode. If the user attempts to launch `./calchep` in this case, it will immediately close and print the following error message to the shell

```
Error: You have launched the interactive session for a version
of CalcHEP that has been compiled without the X11 library.
Presumably, the X11 development package is not installed on
your computer.
```

Nevertheless, all the non-interactive functionality, including the batch session, should still work. If the user would like to use CalcHEP in interactive mode, he/she should install the following additional package:

<code>libX11-devel</code>	<code>for</code>	Fedora/Scientific, Darwin(MAC)
<code>libX11-dev</code>	<code>for</code>	Ubuntu/Debian
<code>xorg-x11-devel</code>	<code>for</code>	SUSE

After installing the X11 development libraries, the user should recompile CalcHEP by issuing the commands:

```
gmake clean
gmake
```

When `gmake clean` asks whether to remove `FlagsForSh`, the user should answer yes, since CalcHEP needs to regenerate it with the X11 library information.

Limits on size of textual strings. Some CalcHEP models need very long lines in the model files for their implementation. Typically, this happens when the model is generated automatically by LanHEP or FeynRules. Although some of the dependent parameter definitions could, in principle, be split over several lines, these packages write the entire expression to a single line. When the line is too long for the interactive session of CalcHEP, it will print the following error message to the shell:

```
Error in model file  ./models/*****
Length of record exceeds the maximum defined by
the parameter  STRSIZ=4096
which is defined in c_sources/chep_crt/include/syst.h
```

The user can solve this problem by using the compiler option `-DSTRSIZ=<needed_number>` where `<needed_number>` is the line length required by the model. This option should be set in the `FlagsForSh` file and CalcHEP should be recompiled:


```
gmake clean
gmake
```

However, this time, `FlagsForSh` should not be removed.

3 Elements of the user interface

In this section, we would like to discuss the general elements of the CalcHEP graphical user interface. Among these elements are the *on-line help*, the *menu*, *messages*, the *string editor*, the *table editor*, the *diagram viewer* and the *plot viewer*. The user can control them using the *Arrow* keys, the *Enter* key, the *Esc* key², the *Backspace* key, the *PgUp/PgDn*³ keys and mouse clicks⁴.

Additionally, the user can control whether colors and sounds are used and can choose the font used in the graphical user interface. The user can set all of these preferences by editing the *calchep.ini* file located in his/her work directory. This file contains three lines, one for each of these settings. The color and sound are set by simply toggling between **on** and **off**. The font is set by specifying the full font string of the desired font. Other fonts available on the users system can be obtained by the command **xlsfonts**, however, only non-proportional fonts should be used for CalcHEP. We recommend fonts from the Courier family. The default value should give the user some guidance.

1. On-line Help. During interactive symbolic and numerical sessions, the user can press the *F1* key to be shown context sensitive help. This help is in the form of a textual message explaining the currently available options to the user. If the text is longer than can be seen at one time in the graphical user interface, there will be a *PgDn* symbol in the bottom-right corner of the help screen. By pressing the *PgDn* key (or by clicking on the *PgDn* symbol), the user can advance the text to the next page. The user can close the help window by pressing the *Esc* key or by clicking the asterisk in the top-left corner of the help screen.

2. Menu. The details of a calculation in CalcHEP are controlled by a series of menus which allow the user to set the properties of the calculations. These menus appear on the right side of the graphical user interface as a vertical list of the available options (see Fig. 1.) The current menu item is highlighted and the user can move between the items by pressing the *Up* and *Down* keys or by clicking on the desired menu item with his/her mouse. Once the desired menu item is highlighted, the user can activate it by pressing the *Enter* key or by clicking on it again with the mouse.

²Use the '**Ctrl** [**]**' sequence if the *Esc* key is missing on your keyboard

³On some keyboards *Prev* and *Next* replace *PgUp* and *PdDn*.

⁴CalcHEP is only sensitive to the release of the left mouse button. It is not sensitive to the press of the mouse button.

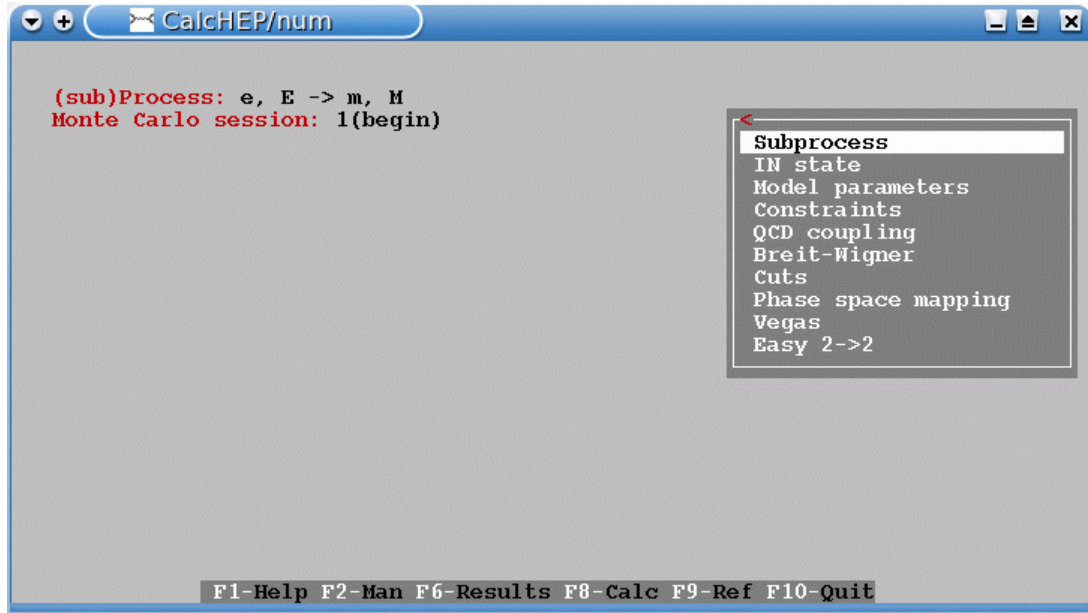


Figure 1: An example of the CalcHEP graphical user interface with a menu displayed.

If the menu is too long to fit in the graphical user interface, *PgUp* and/or *PgDn* will appear at the respective top and/or bottom of the menu. The user can scroll to the other available menu items by pressing the *PgUp*/*PgDn* keys or by clicking on the *PgUp*/*PgDn* symbols on the menu.

As the user's calculation progresses and he/she moves through the menus, he/she can always return to a previous menu by pressing the *Esc* key or by clicking on the *<* symbol at the top-left corner of the menu border.

When a menu is present, CalcHEP is also sensitive to the Function keys *F1*, *F2*, ..., *F10*⁵. A list of currently active Function keys is displayed at the bottom of the graphical user interface and depends on the currently active menu. The typical Function keys that are available are:

- F1- Help : displays a help message about the highlighted menu item.
- F2- Manual : displays information for using the graphical user interface.
- F3- Models : displays the current model of particle interactions.
- F4- Diagrams : displays the Feynman diagrams for the current process.
- F6- Results : displays and allows the user to delete the output files.
- F9- Ref : displays the CalcHEP website, the required citation for

⁵On Macintosh operating systems these keys can be activated within CalcHEP by holding the *Fn* key down while pressing them.

: using the CalcHEP package, and other acknowledgements.
 F10- Quit : quits the CalcHEP session.

The function key functionality can also be activated by clicking the mouse on the function key name at the bottom of the graphical user interface. They can also be activated by pressing the numeric key corresponding to the Function key. For example, '3' will be interpreted as $F3$ and so on. The only caveat is that '0' initiates the $F10$ key.

Another helpful feature of the menu is the menu search. Some menus can be quite long. For example, some model parameter lists are much longer than can be shown in one screen. In situations such as this, the user can press the f , F or \hat{F} key and a text box will open. The user can then type the desired menu item and press enter. The menu will immediately skip ahead to, and highlight, the menu item matching the users string.

3. Messages. At various times, CalcHEP displays a message on the screen. There are two kinds of messages. The first is informational and does not require a response from the user. The informational message ends with “*Press and key*”. The user can continue his/her work by pressing any key or by clicking on any part of the graphical user interface with the mouse. The second kind of message is in the form of a dialog box. This message ends with ($Y/N?$) and the user is required to make a choice by pressing the Y or the N key. The user can also click on the Y or the N in the dialog with the mouse.

4. String Editor. At times (e.g. the input of a new process) the user is required to enter a textual string. At these moments, CalcHEP provides a text box where the user can enter and modify his/her text. If text has already been entered in this text box in the past, CalcHEP will often remember this string and present it as the default value in the text box for convenience. If the user's first input is a letter, number or other textual symbol, the old string will be removed and a new string will be started. If, on the other hand, the first input is non-textual (such as a mouse click, a tab or an arrow key), the old string will be kept and the user will be able to modify it. To edit this text string, the user can use the *left/right arrow* keys or the mouse to move the cursor to the desired position. The *Delete* and *Backspace* keys both remove a character to the left and move the cursor back one position to the left. When the user is finished with his/her text string, he/she can press enter to accept it. If the user wishes to cancel, he/she can press the *Esc* key.

A1	A2	A3	A4	>	Factor	< >	Lorentz part
G	G	G			GG		$ m1.m2*(p1-p2).m3+m2.m3*$
G	G	G.t			$GG/\sqrt{2}$		$ m1.M3*m2.m3-m1.m3*m2.M3$
W+	W-	A			-EE		$ m1.m2*(p1-p2).m3+m2.m3*$
W+	W-	Z			$-EE*CW/SW$		$ m1.m2*(p1-p2).m3+m2.m3*$
W+	W-	Z	Z		$-(EE*CW/SW)^2$		$ 2*m1.m2*m3.m4-m1.m3*m2.$
W+	W+	W-	W-		$(EE/SW)^2$		$ 2*m1.m2*m3.m4-m1.m3*m2.$
W+	W-	A	Z		$-EE^2*CW/SW$		$ 2*m1.m2*m3.m4-m1.m3*m2.$
W+	W-	A	A		$-EE^2$		$ 2*m1.m2*m3.m4-m1.m3*m2.$
h	W+	W-			$EE*MW/SW$		$ m2.m3$
h	Z	Z			$EE/(SW*CW^2)*MW$		$ m2.m3$
h	h	h			$-(3/2)*EE*Mh^2/(MW*SW)$		$ 1$
h	h	h	h		$(-3/4)*(EE*Mh/(MW*SW))^2$		$ 1$
h	h	Z	Z		$(1/2)*(EE/(SW*CW))^2$		$ m3.m4$
h	h	W+	W-		$(1/2)*(EE/SW)^2$		$ m3.m4$
M	m	h			$-EE*Mm/(2*MW*SW)$		$ 1$
L	l	h			$-EE*ML/(2*MW*SW)$		$ 1$
C	c	h			$-EE*Mc/(2*MW*SW)$		$ 1$
B	b	h			$-EE*Mb/(2*MW*SW)$		$ 1$
T	t	h			$-EE*Mt/(2*MW*SW)$		$ 1$
E	e	A			-EE		$ G(m3)$
M	m	A			-EE		$ G(m3)$

Figure 2: An example of a CalcHEP table.

5. Table Editor. CalcHEP uses a *table* structure to store information about the model's parameters, particles, and vertices as well as for the cuts and distributions of a numerical calculation. For each of these cases, one row stores the information for one parameter, one particle, one vertex, one cut and one distribution respectively. CalcHEP has a table editor which allows the user to view and, at times, to modify the contents of these tables.

An example of a table can be seen in Fig. 2. The top line of a table window displays the title of the table. Below this, the table is surrounded by a frame box. The columns of the table are separated by vertical lines. The first table row contains the column names and below this, the table data is displayed. One cell (the intersection of a row and a column) is highlighted at a time. This is the current cell and contains a cursor if the table is open for editing. The current line number is shown in the top-right corner of the window.

To change the position of the highlighted cell and/or the cursor, the user can use the arrow keys, the *Tab* key, and the click of the mouse. Pressing any printing symbol will enter that symbol in the current cell at the cursor position, if in edit mode. The *PgUp* and *PgDn* keys allow the user to scroll the table up and down. The *F1* and *F2* keys provide help information about the current table and the table editor respectively. Exiting the table is achieved by pressing the *Esc* key.

There are some further auxiliary commands which can help the user when working with tables. These commands are achieved by holding down the *Control* key (^) while pressing another key or by clicking on the command label displayed on the table border with the mouse. These commands are:

- **Xgoto** (^X) allows the user to enter a position and then moves the cursor to that position in the current cell.
- **Ygoto** (^Y) allows the user to enter a line number and then moves the cursor to that line. Entering \$ causes CalcHEP to move to the end of the table.
- **Find** (^F) allows the user to enter a string or comma separated list of strings to be searched for in the table. CalcHEP will search the table for the search string(s) and move the cursor to the position of the next occurrence of the string(s), starting from the current position of the cursor. If the user enters a comma separated list of strings, CalcHEP will search for a row containing an instance of each of the strings, where the order of the strings is not important. Spaces are matched as well as printing characters. Pressing ^F again, after a search, will result in the cursor moving, again, to the next occurrence of the string. A new search string can be entered by changing the position of the cursor first and then pressing ^F. The original purpose of table searching was to facilitate finding particular vertices in the vertex table, however, it is available in all tables.
- **Write** (^W) allows the user to enter a filename and then writes the contents of the current (highlighted) cell to that file. If the content of the cell is very long then new line symbols are inserted automatically.

The above commands are available in both modes of the Table Editor and are displayed along the bottom border of the table. The following commands are available only if the table is open for editing and appear along the top border of the table:

- **Clr** (^C) clears the contents at the cursor position and to the right of the cursor position in the current cell.
- **Read** (^R) allows the user to enter a filename and then reads the contents of that file and enters it into the current (highlighted) cell. Spaces and new-line symbols are ignored. The size of the cell is increased automatically to accomodate the new string.

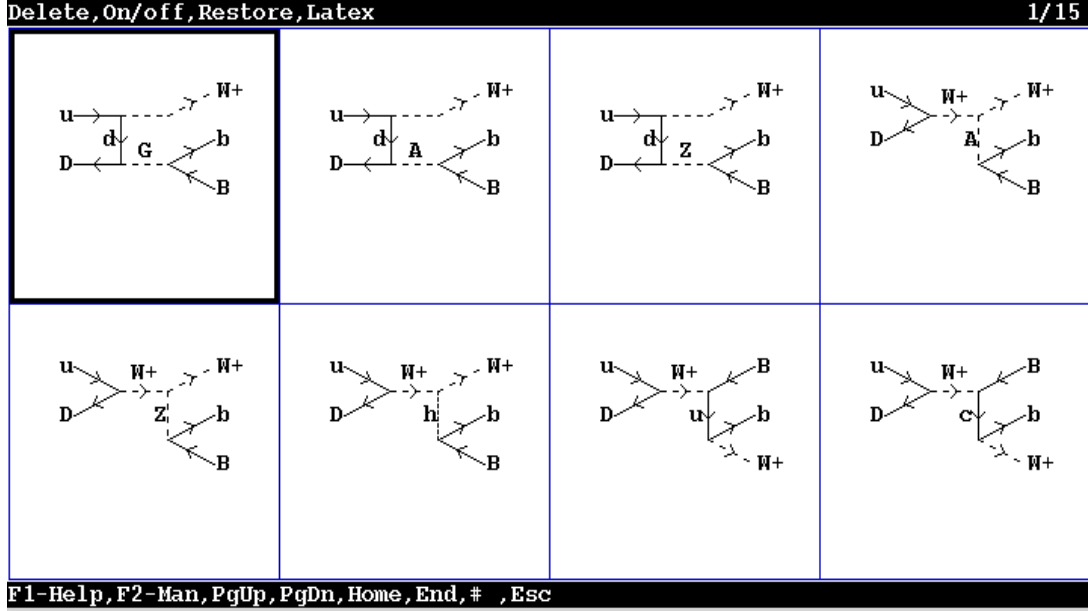


Figure 3: An example of the CalcHEP Diagram Viewer.

- **Del** (^D) removes the current row of the table and stores it in the buffer. Pressing the *Enter* key, creates a new row with buffer contents.
- **Size** (^S) allows the user to change the width of the current column. Column sizes are, also, increased automatically if the user enters a string which is too wide for the current column.
- **ErrMess** (^E) redisplayes the most recent error message associated with editing a table.
- One can use the *Enter* key to creates a new row.

6. Diagram Viewer. The Diagram Viewer is designed to display multiple Feynman diagrams at a time. The viewer splits the window into rectangular cells and puts one diagram in each cell. Each cell is enclosed by a frame while the current diagram is highlighted by having a thicker frame than the others. The index of the current diagram is displayed at the top-right of the window along with the total number of diagrams. An example can be seen in Fig.3.

The number of diagrams which can be displayed simultaneously depends on the size of the window. The user can increase this number by increasing the size of the window using his/her window manager (typically by dragging an edge or corner of the window).

The currently highlighted diagram can be changed by using the *Arrow* keys and/or the mouse. Furthermore, the diagrams can be scrolled up and down by pressing the *PgUp* and *PgDn* keys. The *Home* and *End* keys move to the beginning and end of the diagrams respectively. To move directly to the diagram with index *n*, the user may press the *#* key, type *n* and press enter. Exiting the Diagram Viewer can be done by pressing the *Esc* key. Alternatively, these commands can be accomplished by clicking on the command labels located at the bottom-left of the Diagram Viewer with the mouse.

The Diagram Viewer may, also, have some optional functions which depend on the context. These are:

- **Delete (D)** which allows the user to turn off all the diagrams.
- **On/off (O)** which allows the user to toggle the currently highlighted diagram between on and off.
- **Restore (R)** which allows the user to turn on all the diagrams.
- **Latex (L)** which allows the user to write all the diagrams (that are not turned off) to file in L^AT_EX axodraw [44] syntax.
- **Ghosts (G)** which displays the current diagram along with all other diagrams which are related to the current one by replacing gauge bosons with their associated ghosts and Goldstone bosons. This command is only available for squared diagrams.

These commands can be invoked by pressing the key marked in parentheses or by clicking on the label along the top-left of the Diagram Viewer window.

7. Plot Viewer. The Plot Viewer is designed to display histograms (see Fig. 4) and continuous curves (see Fig. 5). After being launched, the Plot Viewer displays the plot and waits for a signal from the keyboard or the mouse. If the mouse is clicked inside the plot, the x-coordinate of the mouse is displayed at the bottom of the window. Additionally, the value of the function or histogram at that x-coordinate value is shown. In the case of a two-dimensional histogram density plot, both the x-coordinate and the y-coordinate along with the histogram height are shown.

If a key is pressed, a menu appears (see Fig. 5) allowing the user to control some aspects of the plot as well as to export the plot data. The available options are:

- **Y-max** which allows to set the maximum height of the plot.

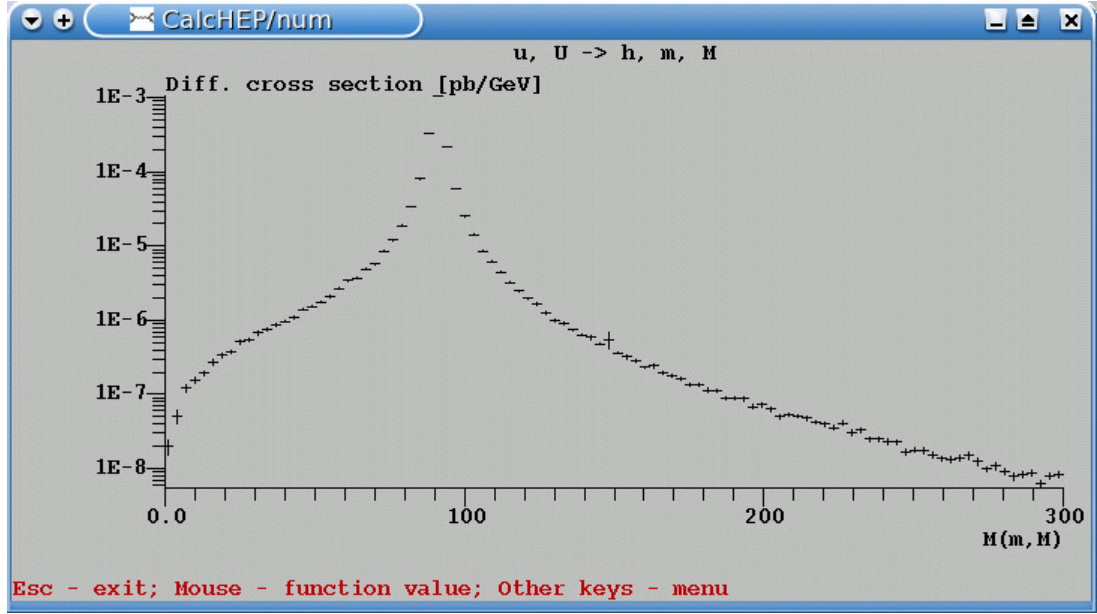


Figure 4: An example of a histogram plot in CalcHEP.

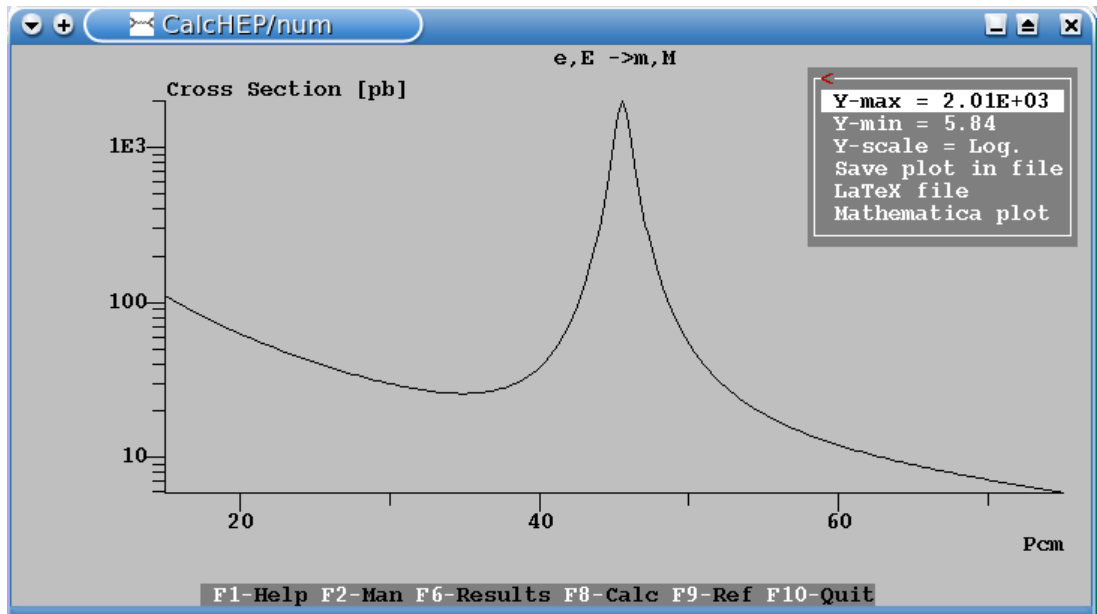


Figure 5: An example of a continuous plot in CalcHEP for a $2 \rightarrow 2$ process.

- **Y-min** which allows to set the minimum height of the plot.
- **Y-scale** which allows whether the vertical axis should be linear or logarithmic. However, note that the logarithmic scale is only available if the lower limit is positive and the ratio of upper and lower limits is greater than ten.
- **Save plot in file** which allows to save the plot data to the file `plot_#.txt` where `#` is an integer. This file also includes example plot instructions for gnuplot and PAW. Additionally, the file `plot_#.gnu` is created which has gnuplot instructions and can be used by issuing the command `gnuplot < plot_#.gnu`. The file `plot_#.kumac` is also created and contains PAW instructions which can be used by issuing the command `paw -b plot_#.kumac`. Alternatively, the user can load the data, interactively, in either gnuplot or PAW.
- **Math file** which allows to save the plot data to the file `plot_#.math` in Mathematica syntax. It can be read into a Mathematica session using `Get[<file>]` where `<file>` is replaced with the file name and path in parantheses. This file, also, includes example plotting commands which the user can copy and modify to suit his/her purpose.
- **LaTeX file** which allows to save the plot in the form of a `LaTeX` axodraw [44] file. The axodraw style file is required for compilation and can be found in the `$CALCHEP/utile` directory.

In the case of a two-dimensional histogram, the Plot Viewer represents the differential cross-section of each bin by a solid black rectangle where the height and width of the black rectangle is given by

$$S = 0.9 * (binSize) * (F/Fmax)^{P/2} \quad (1)$$

where S is the height or width of the rectangle, $(binSize)$ is the size of the vertical or horizontal bins, F is the differential cross-section for that bin, $Fmax$ is the maximum differential cross-section in the histogram and P is a parameter chosen by the user to achieve the best resolution. It defaults to the value 1. If a two-dimensional histogram is being viewed, the first three menu options are replaced with:

- **S=F^P** which allows to choose the scaling of the size of the rectangle relative to the differential cross-section as described above. Large values for P tend to increase the resolution for large histogram bins but reduce the resolution for small histogram bins. Small values for P do the opposite.

When finished viewing the plot, the user can exit the Plot Viewer by pressing the *Esc* key.

In addition to using the Plot Viewer in the interactive numerical session, the user can also view plots that have been saved to file by using the executable `plot_view` located in the `$CALCHEP/bin` directory. This command takes one option which is the file where the plot data has been exported as in

```
$CALCHEP/bin/plot_view plot_#.txt
```

This is a stand-alone version of the Plot Viewer.

9. File Search Engine. When CalcHEP requests a file or directory from the user (such as when importing a new model), it opens a text box where the user can enter the filename. The user can enter the full file name, including the path, or he/she can enter part of the path and end his/her text with `/*`. When this is done, CalcHEP opens a menu where the contents of the directory are listed (see Fig. 6.) The user can use the menu functionality to choose the directory or file he/she wants, after which, the file or directory name in the text box will be updated accordingly. This can be continued until the desired file or directory is found. If the final target is a directory, the user must finally remove the `*` and press enter to choose the directory. The initial path can also be started with the environment variables `$CALCHEP/`, `$WORK/`, `~/` and `~<user_name>/`.

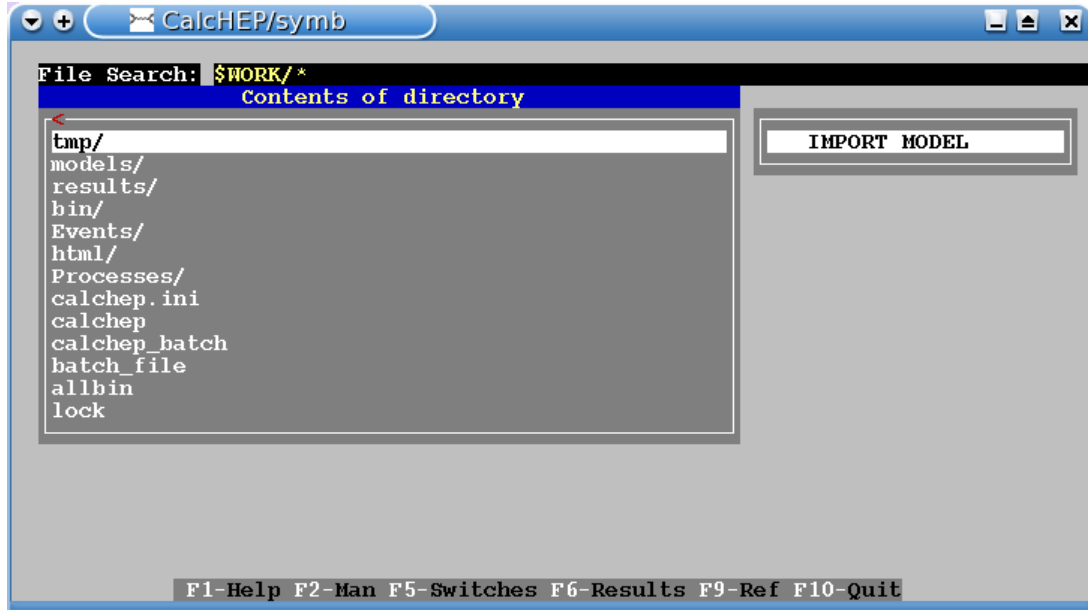


Figure 6: An example of the CalcHEP File Search Engine.

4 Menu system for symbolic calculation

The flow of menus in the symbolic calculation session is presented schematically in Fig. 7.

4.1 Model Choice and Manipulation

Menu 1. This menu presents a list of available models and allows the user to choose among them for his/her calculations. Additionally, at the bottom of this list is the entry **IMPORT MODEL** which allows the user to import a new model of particle interactions into CalcHEP. When **IMPORT MODEL** is chosen, the File Search Engine will open allowing the user to specify the directory where the new model is stored. It will then allow the user to choose among the models in that directory and choose a new name for the model if desired, after which the model will be imported and appear on the list of models in this menu.

Menu 2. The first item on this menu allows the user to enter a physical process and will be explained further in Subsection 4.3.

The second item on this menu is **Force Unit. Gauge** and allows the user to use Unitary gauge in his/her calculation, even if the model is implemented in t'Hooft-Feynman gauge. Generally, we recommend to use t'Hooft-

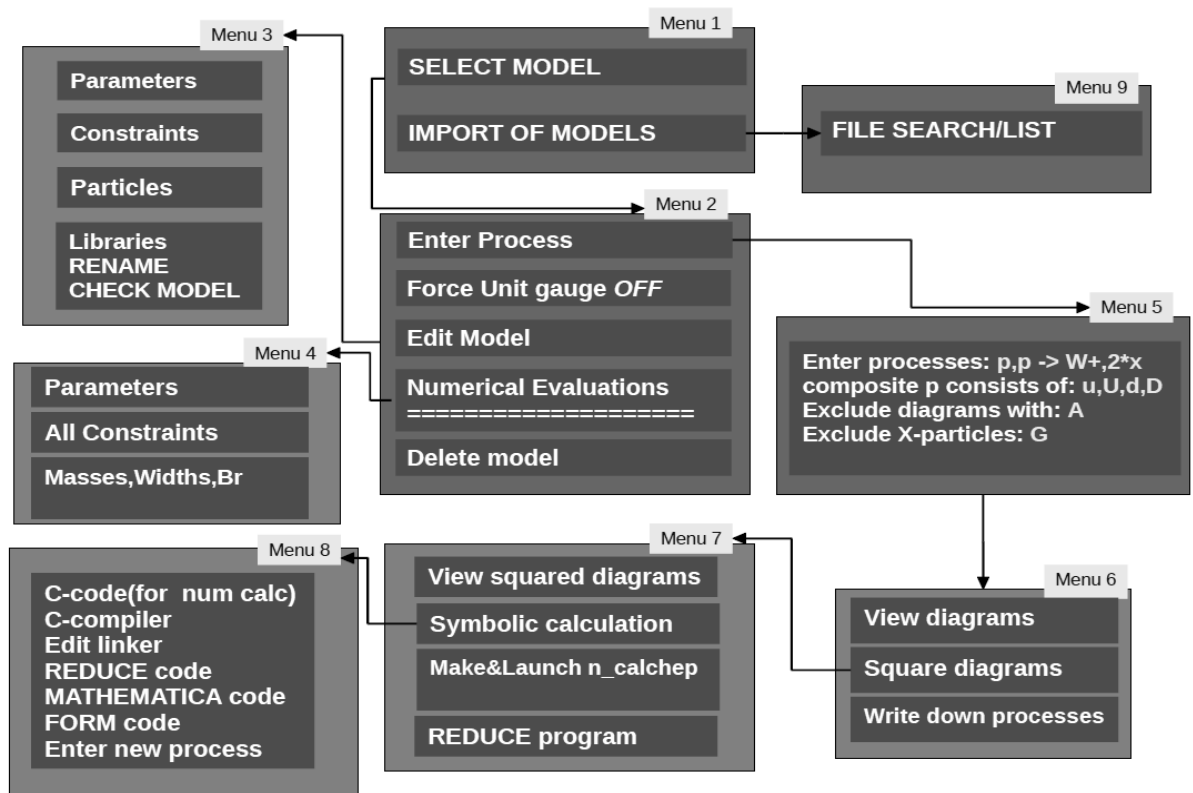


Figure 7: Menu flow for the interactive symbolic session.

Feynman gauge, whenever possible, as the ultra-violet cancellations between diagrams are much better in this case. Further information can be found in Section C.

The third item on this menu is **Edit Model** and allows the user to view and modify the current model and will be described further when we discuss **Menu 3** below. We, also, note that after the user reaches **Menu 5** (where a process is entered), he/she can still view the model by pressing the **F3** key but cannot modify it.

The fourth item on this menu is **Numerical Evaluation** and allows the user to view the value of the dependent variables as well as the masses, widths and branching ratios of the particles. Further details can be found in Subsection 4.2.

The final menu item is **Delete model** which removes the current model and returns CalcHEP to **Menu 1** where a new model can be chosen. Before removing the model, a warning dialog appears and allows the user to cancel the model deletion.

Menu 3. The model details are stored in five text files located in the **models** subdirectory of the users work directory. These model files can be edited with any text editor. However, the user must not include any **Tab** symbols and must keep the column structure of the file unchanged. We recommend that the user take advantage of the built-in model editor which can detect mistakes in the user's input.

This menu items appearing for this menu are:

- **Parameters** : Edit the independent parameters of the model.
- **Constraints** : Edit the dependent parameters of the model.
- **Particles** : Edit the particle properties of the model.
- **Vertices** : Edit the Feynman interaction vertices of the model.
- **Libraries** : Edit the prototypes for external functions and the external libraries which should be linked to the numerical code.
- **RENAME** : Edit the name of the model.
- **CHECK MODEL** : Checks whether the model passes a set of tests described below.

The detailed format and requirement for these files is described in Section 8. An explanation of the **Table Editor** which is used to modify the model files can be found in Section 3.

When the user is satisfied with his/her modifications, he/she can press the **Esc** key. When this occurs, CalcHEP first asks whether the user would like to save his/her modifications. If the user answers **N**, CalcHEP returns to **Menu 2** and the modifications are lost. If, however, the user answers **Y**, CalcHEP first performs the following series of tests on the model (which can also be initiated by choosing the **CHECK MODEL** menu item):

- Do all particle and parameter names satisfy the CalcHEP naming requirements.
- Are all numbers entered correctly.
- Have all parameters been declared before they are used in any expressions.
- Are all particles used in a vertex defined.
- Are the algebraic expressions correct.
- Are all the Lorentz indices correctly contracted.
- For any vertex, is its conjugate vertex included.

If all these tests are passed after pressing the **Esc** key and choosing to save the model, the model is saved to disk and CalcHEP returns to **Menu 2**.

If, on the other hand, any of the tests are failed, CalcHEP stops the tests after the first detected error and displays a message describing this error along with the table and position in that table where the error can be found. In this case, the user remains in **Menu 3** and is allowed to fix the mistake in his/her input. The error message can be viewed again by pressing **^E** in any of the model tables. After the user fixes the mistake, he/she can press the **Esc** key and try to save the model again.

4.2 Numerical Evaluations

After choosing **Numerical Evaluations** on **Menu 2**, the user is taken to **Menu 4** where numerical evaluations of the dependent parameters, masses and widths can be done. The default independent parameters are those defined in the model files. However, the user can change the values of the independent parameters which are used for these numerical evaluations by choosing **Parameters** on this menu. This will cause CalcHEP to display a menu which lists all the independent parameters and allow the user to change them one by one. Furthermore, CalcHEP displays **READ.FROM.FILE** at the

top of this menu. If the user chooses this, CalcHEP opens the **File Search Engine** and allows the user to choose a file. Afterwards, it reads the file and updates all the independent parameters used for the numerical evaluations. This parameter file must be written in the form of two columns separated by whitespace. The first columns must contain the parameter name and the second column must contain the numerical value. Each parameter must be on a separate line. Here is an example for the SM:

```

      EE    3.1223E-01
alfSMZ    1.172E-01
      SW    4.81E-01
      Ml    1.777
      Mtp   175
      MZ    91.1884
      Mh    120

```

These new independent parameter values are only used for the numerical evaluations done in this menu. The default values used for other numerical calculations (via **Enter Process** on **Menu 2**) are those defined in the model files.

Once the user is satisfied with the values of the independent parameters, he/she can initiate the numerical evaluation of the dependent parameters by choosing **All Constraints**. By default, CalcHEP will only calculate the independent parameters up to and including any dependent masses. If the user would like further independent parameters calculated, he/she can add the keyword **%Local!** to the dependent parameter definitions. CalcHEP will then calculate all independent parameters up to **%Local!**. An example of how to include the **%Local!** keyword is:

```

Constraints
Name      | Expression      |
.....
%Local!   |                  |

```

where the line with the ... represents a list of dependent parameter definitions.

The calculated dependent parameters will appear in a menu which the user can scroll through. The user can also choose one of the dependent parameters in this menu in order to view its dependence on the independent parameters. This is accomplished by displaying another menu that allows the user to choose the independent parameter and then choose the beginning value, the ending value and the number of points to evaluate. The results will be plotted on screen in the **Plot Viewer**.

The final menu item of Menu 4 is **Masses, Widths, Branch..** This item will bring up a new menu which lists all the particles in the model. Choosing any particle in this menu will cause CalcHEP to calculate its mass (if dependent) as well as its width and branching ratios (if any) and display them onscreen along with other particle information. Additionally, the user may choose **ALL PARTICLES** at the top of this menu which will cause CalcHEP to calculate all the dependent masses, widths and branching ratios and write them in a file in the **results** subdirectory of the work directory with the filename **decaySLHAN.txt** where N is an integer. The format of this file follows the SLHA [37] convention and should be suitable for other programs that follow this convention.

When calculating the decay widths and branching ratios, CalcHEP first calculates the contribution from $1 \rightarrow 2$ decays. If the resulting width is zero, it then calculates the contribution from $1 \rightarrow 3$ decays. If still zero, it calculates the contribution from $1 \rightarrow 4$ decays. However, if the model is defined in terms of an SLHA file and that file contains the widths and branching ratios, CalcHEP does not calculate them, but uses the values specified in the SLHA file.

4.3 Process Input

After choosing **Enter process** on Menu 3 the user is presented with the **Process Input** screen (see Fig. 8) where he/she can enter the physical process he/she would like to calculate. At the top of this screen, CalcHEP displays a list of the model particles. Each entry contains the particle name followed by the antiparticle name in parentheses and ends with the full descriptive name for the particle. If the list of particle is too long to fit on the screen, the user may press the *PgUp* and *PgDn* buttons to view the other particles.

Below the particle list, CalcHEP displays the prompt **Enter process:** and presents the user with a text entry box where he/she can enter his/her desired process. The syntax for this entry is:

P1 [,P2] -> P3, P4 [,P5...]

where the incoming particles and outgoing particles are separated by \rightarrow and P1...P5 are (anti)particle names. The total number of (anti)particles should not exceed 6. For example, the input **u, U -> G, G** specifies the annihilation of a u-quark and an anti-u-quark into two gluons.

In place of (anti)particle names, the user can enter **N*x** after the \rightarrow , where N is an integer. CalcHEP replaces this with all possible combinations of N particles and antiparticles from the X-particles list. The default is for this list to contain all the particles and antiparticles from the model. For example,

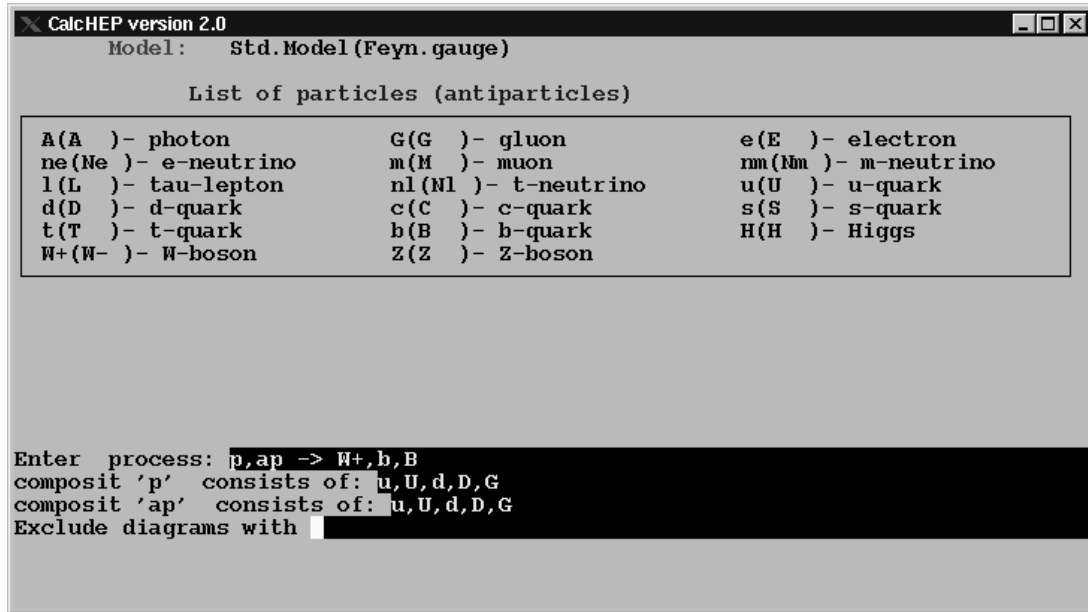


Figure 8: Example of the input of a process in the interactive symbolic session.

the input $u, U \rightarrow G, G, 2 \times$ specifies the annihilation into two gluons plus any other two particles from the model. The user can, however, limit which particles are included in the X-particles list. If $N \times$ is used, CalcHEP presents the user with the text **Exclude X-particles:** followed by a text entry box where the user can list any desired particle and antiparticle limitations. The syntax for this entry is:

Exclude X-particles : P1>n1 [,P2>n2, ...]

where P_1, P_2, \dots are particle names and n_1, n_2, \dots are quantity limits. This instructs CalcHEP to remove diagrams with more than n_1 particles of type P_1 , n_2 particles of type P_2 , and so on, in the part of the final state specified by $N \times$. The specification $P > 0$ can be shortened to P and forbids the appearance of the particle P among the X-particles.

The user may also enter an alias for multiple particles, such as a p for the partons in a proton, j for the particles that produce jets, or l for leptons. The user can use any short name he/she likes as long as it is different than the names of the (anti)particles defined in the model. When an alias is used in the process, CalcHEP requests its definition. For example, if the user enters the process $p, ap \rightarrow W^+, b, B$, and there is no p or ap defined in the model, CalcHEP will display the prompt **composit 'p' consists of:** followed by a text entry where the user can specify which particles and/or antiparticles he/she would like included in the alias definition of p . In this

example, the same is done for the `ap` entry. This specifies the collision of any particles in the definition of `p` against any particles in the definition of `ap` and producing the particles `W+,b,B`. Aliases can be used both for the incoming and the outgoing particles.

When the initial state particles are massless, the user may request polarized beams. The way this is accomplished is by adding the `%` symbol to the end of a massless particle name. For example, entering the process

```
e%,E%→A,A
```

will cause CalcHEP to generate the code for the annihilation of polarized e^+ and e^- beams to produce two photons. In the current version, we only consider initial states which consist of mixture of left and right polarizations. We do not, currently, support linear polarizations.

After the process has been entered, CalcHEP allows the user to enter any particles he/she would like excluded from the internal lines of the diagrams. It does this by displaying the text **Exclude diagrams with:** followed by a text input box. The syntax for this entry is as follows

```
Exclude diagrams with : P1>n1 [,P2>n2,...]
```

where `P1,P2,...` are particle names and `n1,n2,...` specify the maximum number of internal lines that can contain these particles. In other words, diagrams where the number of internal lines containing `P1` is greater than `n1` or where the number of internal lines containing `P2` is greater than `n2` (and so on) will not be constructed. For example,

```
Exclude diagrams with : W+>1
```

will cause CalcHEP to only construct diagrams with zero or one `W+` internal line. The input `P>0` can be shortened to `P` and is understood by CalcHEP to mean that `P` can not appear in any internal lines. If the user leaves this text entry blank, the full set of diagrams will be constructed.

At any time during the process entry, the user may press the *Esc* key to return to the previous input and the *F1* key to get process input help. After the process entry is complete, CalcHEP generates all the Feynman diagrams satisfying the user's constraints. If no diagrams are allowed, CalcHEP displays a warning message and returns the user to the beginning of the process entry to try again. If one or more diagrams are constructed, CalcHEP advances to Menu 5, which we describe next.

4.4 Squared Diagrams and Symbolic Calculation

Menu 5. This menu appears on the screen after the construction of the Feynman diagrams along with information about the number of diagrams and subprocesses generated.

The first menu item is **View diagrams** and allows the user to view a graphical representation of the generated Feynman diagrams via the **Diagram Viewer**. In addition to viewing the diagrams, as described in Section 3, the user can remove some of the diagrams before they are squared and can generate L^AT_EX output for the diagrams which are not removed.

If more than one subprocess is generated, CalcHEP will first present the user with a list of the subprocesses when he/she chooses **View diagrams**. Each subprocess will be listed along with the number of diagrams for that subprocess. The user can move among the subprocesses by using the *PgUp* and *PgDn* keys or by using the mouse. If the **F7** key is pressed while on this menu, all the diagrams for the highlighted subprocess are removed. On the other hand, if the **F8** key is pressed, all the diagrams are restored for the highlighted subprocess. If the **Enter** key is pressed while on this menu (or the mouse clicks on the highlighted process), the **Diagram Viewer** will open with the diagrams for the highlighted process.

We will now list some details of the visual representation of Feynman diagrams used in CalcHEP.

- Incoming particles are drawn on the left side of the diagrams, while the outgoing particles are shown on the right.
- CalcHEP uses dotted lines for scalar particles (spin 0), dashed lines for other bosonic particles (spins 1 and 2) and solid lines for fermionic particles (spins 1/2 and 3/2).
- Charged particles are represented by lines with arrows. The arrow indicates the direction of the particle (not the anti-particle) propagation.
- Incoming and outgoing particles are labeled by their names at the end of their lines. Virtual particles are labeled by their names at the middle of their lines. If a particle is not self-conjugate, the particle's name is used for the labeling (not the anti-particle's name).
- In the case of scattering processes, the first scattering particle enters at the top of the diagram while the second scattering particle enters at the bottom.
- CalcHEP produces only one representative diagram for a set of diagrams which can be transformed into one another by replacing identical outgoing particles. For example, CalcHEP creates only one diagram for the SM $e^+, e^- \rightarrow \gamma, \gamma$ process, whereas a textbook would present

two diagrams. The reason for this is that CalcHEP has not yet assigned the momenta in the diagrams, so the representative diagram is sufficient.

- At this stage, CalcHEP does not generate diagrams with the Fadeev-Popov ghosts or the Goldstone bosons associated with gauge symmetry breaking. These fields are restored when the diagrams are squared. This can be done because the vertices with the Fadeev-Popov ghosts and Goldstone bosons are related to the vertices with the gauge bosons. After squaring, each squared diagram with a gauge boson in it gives rise to a set of squared diagrams with the gauge bosons replaced with the Fadeev-Popov ghosts and Goldstone bosons as determined by the Feynman rules. In some cases, there is no gauge boson vertex corresponding to a vertex with Fadeev-Popov ghosts or Goldstone bosons (such as the G_0^4 vertex where G_0 is the Goldstone boson eaten by the Z boson). In these cases, CalcHEP produces a diagram as if the corresponding gauge boson vertex existed (such as a Z^4 vertex). After squaring, this squared diagram simply represents the ones with the Fadeev-Popov ghosts and Goldstone bosons as determined by the Feynman rules. (The actual Z^4 diagram is dropped while the ones with the Fadeev-Popov ghosts and/or Goldstone bosons are kept. Further details can be found in Sections 8.7, 8.4, and E.
- Vertices with a non-trivial color structure (for example, the four-gluon vertex of the SM) are implemented by means of an unphysical tensor auxiliary field. The vertices involving this auxiliary field are treated in the same way as the Fadeev-Popov ghosts and Goldstone bosons. The Feynman diagrams involving these auxiliary fields are not constructed at this point. They are restored after the diagrams are squared. See Sections 8.7, 8.4, E for further explanation.

The second menu item is **Squaring** and causes CalcHEP to create squared diagrams. CalcHEP uses these squared diagrams for subsequent calculations of squared matrix elements. See Section C for the details.

The **Write down processes** menu item creates the file `list_prc.txt` in the `results` subdirectory. This file contains a list of the constructed subprocesses.

Menu 6. The **View squared diagrams** menu item is similar to the **View diagrams** of the previous menu, however, it displays the squared diagrams. Each squared diagram is a graphical representation of AB^* , where A and B are Feynman diagrams constructed in the previous step.

We summarize some features of the squared diagrams in CalcHEP :

- CalcHEP does not construct both AB^* and BA^* . Instead, it only generates AB^* and calculates its contribution to the squared matrix element as $2\text{Re}(AB^*)$. This results in smaller, more efficient code.
- CalcHEP constructs only one representative of a set of squared diagrams which can be transformed into one another by permutations of identical outgoing particles. The needed symmetrization for these particles is performed during the symbolic or numerical calculations of the squared matrix element. Again, this results in smaller, more efficient code.
- Each squared diagram represents a set of squared diagrams where some physical particles are replaced by their associated ghosts, Goldstone bosons and/or auxiliary fields in all possible ways according to the Feynman rules defined for the model. This set of squared diagrams can be viewed at this stage by pressing the **G** key while the desired squared diagram is highlighted.

Just as for the diagrams of the previous menu, the squared diagrams can be deleted by the user while in the **Diagram Viewer**. Furthermore, if the squared diagrams have already been calculated, each diagram will contain one of **CALC**, **ZERO**, **Out of memory** or **Del**. They mean, respectively, that the squared diagram is calculated, is identically zero, the calculation ran out of memory, or the squared diagram was deleted by the user.

The second menu item is **Symbolic calculation** and instructs CalcHEP to begin the symbolic calculation of the squared matrix element using the generated squared diagrams. This is done by the built-in symbolic calculator. During this calculation, CalcHEP displays the current status of the calculation, which includes which diagram is currently being worked on and how many are left.

The main goal of the CalcHEP package is to generate *C* code that numerically calculates the squared matrix element. The next menu item **Make&Launch n_calchep**, performs the symbolic calculation as described in the previous paragraph. It then writes the *C* code for those squared diagrams, compiles it and executes the resulting interactive numerical code for the generated squared diagrams. It also advances to the next menu.

The next menu item is **Make n_calchep** which causes CalcHEP to perform the symbolic calculation of the squared matrix elements, write the *C* code and compile it. However, in distinction to **Make&Launch n_calchep**, it does not execute the resulting code. Also, in distinction, it closes the

graphical user interface and performs these steps in the background. When it is finished, it prints the message `n_calchep is created` to stdout and quits. The executable can be found in the `results` subdirectory of the user's work directory. A `lock` file is stored in the user's work directory to prevent CalcHEP from having multiple instances running at the same time and interfering with each other.

The **Reduce program** menu item creates a version of the squared diagrams that is formatted for the *Reduce* program [45]. Each squared diagram is put in a separate file `pm.n.red` where `m` is the subprocess number and `n` is the squared diagram number. These files are not used further by CalcHEP, but can be useful when the user would like a symbolic expression for the squared matrix element. Moreover, they can be used to check the results of the CalcHEP symbolic calculator. CalcHEP includes some tools for checking the symbolic calculator using *Reduce*. More details can be found in Appendix B.

4.5 Output of results and launching of the numerical calculation

Menu 7. This menu occurs after the symbolic calculations have been performed. The first menu item is **C-code** which causes CalcHEP to write *C* code for the squared diagrams to the `results` subdirectory of the work directory. After the *C* code has been written, the user can execute the second menu item **C-compiler** which cause CalcHEP to compile the *C* code and create the executable `n_calchep`. If the compilation is successful, it will launch the resulting interactive numerical session which should appear on the user's screen. If there are problems with the linking, the user can modify the libraries linked by using the **Edit Linker** menu item. Any changes made using **Edit Linker** will be added to the model definition for later use. Details of the numerical interactive session are covered in Section 5.

This menu has three more items allowing the user to export the squared matrix element expressions to formats appropriate for other programs. Each writes the expressions to files in the `results` subdirectory of the work directory. The **REDUCE code**, **MATHEMATICA code** and **FORM code** menu items write the symbolic expressions to *Reduce*, *Mathematica* and *Form* formats respectively. Further manipulations of the symbolic expressions can be performed in those programs as desired by the user. The user could, for example, sum over the squared diagrams, perform substitutions, evaluate the expression numerically or calculate the total cross section. Using these expressions in these external programs is reasonable when the number and size

of the diagrams are small. Further details about this output can be found in Section 11.

4.6 Switches

There are some switches which influence the results of the symbolic calculator and *C* output. They are controlled by a menu which can be obtained by pressing the F5 key. The items on this menu are:

- **Number of QCD colors = 3/inf** : This switch has two possible values: 3 or inf. If this switch is set to 3, CalcHEP performs the usual $SU(3)$ quantum chromodynamic calculations including all the terms. If this switch is set to inf, on the other hand, CalcHEP only calculates the leading term in the large N_c expansion. This removes many interference diagrams which only contribute at higher order and reduces the size of the code. (Of course, the numerical value of N_c is still taken as 3 in the final results.) The default is 3.
- **Diagrams in C-output ON/OFF** : This switch determines whether CalcHEP writes an ascii image of the diagram in the *C* code of the squared diagrams. This can be useful when analyzing the *C* code. The size of the code can be reduced by turning this off. (However, the size of the ascii image is usually small compared to the rest of the code.) The default is ON.
- **Widths in t-channels OFF/ON**: This switch determines whether CalcHEP includes the particle width in the propagator on t-channel lines (where there is no chance that the particle will go on shell.) The default is OFF which means that these t-channel widths are not included. The user may change this to ON in which case the user must also turn this feature on in the Breit-Wigner menu of the numerical session described in the next section.

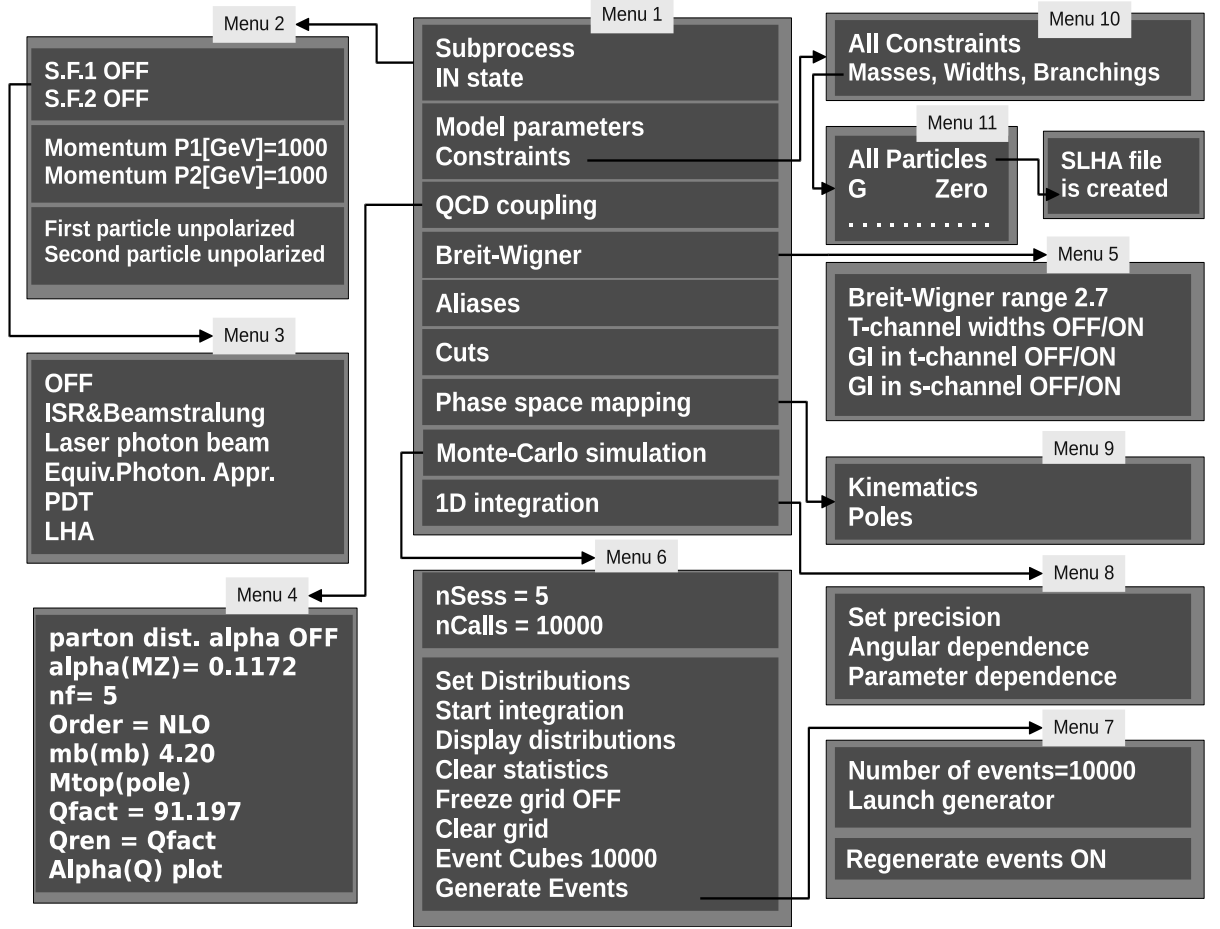


Figure 9: Schematic diagram of the menu system for the numerical session.

5 Numerical session

5.1 Sketch of the menu system.

In Section 4, we described the process of generating and compiling C code for a collision or decay process for any particle physics interaction model. In this section, we describe how to use the resulting executable to calculate the collision cross section or the decay width in interactive mode. In Fig. 9, we present a schematic view of the menu system of the interactive numerical session. An example of the first menu (Menu 1) is shown in Fig. 1. We will now describe each of these menu items.

The first menu item is **Subprocess** which allows the user to choose which subprocess to work on if more than one was generated during the symbolic session. The current subprocess is displayed at the top of the numerical

session screen (see Fig. 1.) If a combination of the subprocesses is desired, we direct the user to the tools described in Sections 6 and 7.

The next menu item is **In state** which allows to enter the momenta of the incoming particles, their polarizations and their parton distribution functions. We note that, in order to set the polarizations, the incoming particles must be massless and the % symbol had to be used in the symbolic session (see Section 4.3.) If this was done, the polarization can be set anywhere between the maximum and minimum helicity value for the particle. (For example, for a fermion, the helicity must be set between $-1/2$ and $1/2$. For a vector boson, the helicity must be set between -1 and 1 , and so on.) The parton distribution functions will be described further in Subsection 5.3.

The next menu item is **Model parameters** which allows to modify the numerical value of the independent parameters which are used in the numerical calculations. The following menu item **Constraints** is the same as in the symbolic session and has been described in Subsection 4.2. However, the QCD strong coupling **GG** is not included under either of these menus since it depends on the scale of the interactions. Its value is controlled by the next menu item **QCD coupling** which is described in Subsection 5.4.

The **Aliases** menu function is intended for declaration of alias names joe particle sets. For example, one can define alias *Jet* for quarks and gluon:

Jet	d,D,u,U,s,S,c,C,b,B,G
-----	-----------------------

Aliases can be used for definition of phase space cuts and histograms. The cut which contains an alias name will be checked for each particle of the set. As well as a new point will be added to histogram for each particle of the set. The **Cuts** menu item allows to set cuts on the Monte Carlo phase space integration and event generation. Details of the cuts specification can be found in Subsection 5.8. The **Breit-Wigner** menu item allows to modify the behavior of the propagators for the unstable particles. Further details can be found in Subsection 5.5. The **Phase space mapping** menu item opens up into a menu with two items. They are **Kinematics** and **Regularization** and are described in Subsections 5.9 and 5.10, respectively. They allow to modify the mapping of phase space to improve Monte Carlo integration.

The menu item **Monte-Carlo simulation** allows to run the *Vegas* [46,47] Monte Carlo integration of the multiparticle phase space to determine the collision cross section or the decay width. The **Monte-Carlo simulation** menu also allows to generate kinematic distributions and generate events. It is described in further detail in Subsection 5.11.

For 2→2 processes with fixed energies of incoming particles the phase space integral is one-dimensional and can be integrated using traditional Riemann approach. In this case, one can chose the **1D integration** which is the last item in the menu. This option is described in Subsection 5.13. For

the 1->2 case we have zero dimension phase space and this option allows a fast summation over channels and calculation of branchings.

5.2 Bookkeeping

Each time any parameters are changed which affect the numerical calculation, CalcHEP increases the session number by one and clears the statistics. The current session number is displayed at the top of the interactive session screen. These parameters not only include the dependent model parameters but also include the choice of subprocess, incoming momenta, parton distribution functions, QCD coupling and cuts. During a *Vegas* session, the full set of parameters for the current session is stored in the file `session.dat` located in the `results` subdirectory. This file changes to match the current session. If the user quits the interactive session and restarts it later, CalcHEP will read the parameters from the `session.dat` file. The user can then continue from where he/she left off.

The full set of parameters for each session is also stored in the file `prt_N` where `N` is the session number. This file, also, contains the results of the Monte Carlo integration. It is useful if the user would like to determine what parameters he/she used in an earlier calculation and what the results were. When CalcHEP generates events, they are stored in the file `events_N` where `N` is the session number. Moreover, distributions are stored in `distr_N` where `N` is the session number. Other results are written by CalcHEP to files with the session number `N` as part of the file name and will be described in later sections.

5.3 Parton Distribution Functions

The first items in the `In state` menu are `S.F.1` and `S.F.2` which control the structure functions of the first and second incoming particles respectively. Each of these menu items opens a new menu which allows to choose whether the user wants the structure functions `OFF` or whether he/she wants the `PDT` structure functions or the `LHAPDF` structure functions. The `LHAPDF` [39] sets require separate installation described below. After making this choice, CalcHEP presents the user with a list of the available structure functions and then finally allows to choose any free parameters of the structure functions. Only structure functions allowed for the incoming particles are listed.

Examples of the list of parton distribution functions and of setting the properties of a `LHAPDF` structure function are presented in Fig. 5.3. The `PDT` structure functions are stored in the directory `$CALCHEP/pdTables`. CalcHEP comes with a set of `PDT` tables. These include structure functions

for initial state radiation (ISR) for incoming electrons, Weizsaecher-Williams structure functions for photons and structure functions for backscattering laser photons which are described further in Section G. Additionally, a set of parton distribution functions for the proton and anti-proton are included as shown on the left of Fig. 5.3. New PDT tables can be added as described in Appendix H.

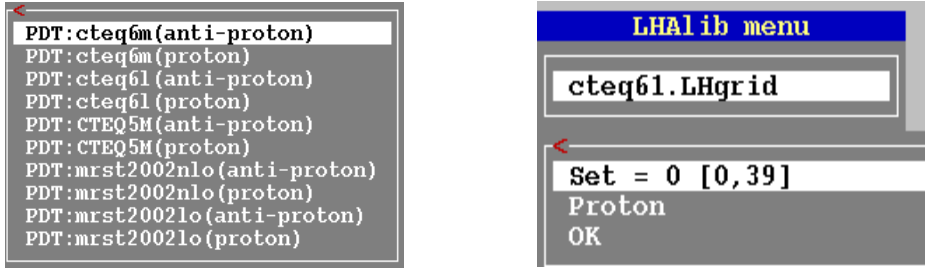


Figure 10: PDT structure functions(left) and LHA settings(right)

To use the LHAPDF structure functions, the LHAPDF library must be installed and linked to the model. The way this is handled in CalcHEP is that it comes preinstalled with a dummy version of the LHAPDF routines which inform it that the LHAPDF structure functions are not used. When the user installs and links the true LHAPDF structure functions, they are used in place of the dummy version that comes with CalcHEP. The desired structure functions along with the LHAPDF libraries according to the LHAPDF instructions (see [39]).

To use the LHAPDF structure function in the CalcHEP, first of all, one has to add

```
-L<path_to_lhapdf> -lLHAPDF
```

line to Libraries model item. It is enough to compile executable `results/n_calchep`, but may be not enough to launch it. If your `<path_to_lhapdf>` is disposed in system area, then problems are not expected. An executable has to know a location of shared libraries used at time of launching of CalcHEP. Standard paths like `/usr/lib` or `/usr/local/lib` are checked by default. An arbitrary library location can be passed to executable via environment `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path_to_lhapdf>`. You can define this enviroment before you start to work with CalcHEP, include it to you startup file (`.batchrc`) or add it to `./calchep` and `./calchep_batch` scripts.

In case of adding it to `./calchep` it is better to use the other variable — `LD_RUN_PATH` which provides the generated `n_calchep` with the property to be launched independently without setting environment variables. For

`./calchep_batch` run we recommend to define `LD_LIBRARY_PATH` variable.

5.4 QCD coupling

The value of the strong coupling constant depends on the scale of the calculation. CalcHEP runs the strong coupling constant. The parameters of this running can be set in the `QCD coupling` menu and include:

- `alpha(MZ)` : the strong coupling value at M_Z .
- `nf` : the maximum number of quark flavors in the running. At small energies CalcHEP takes into account threshold effects caused by charm and bottom quarks and changes `nf`. If `nf=6` then the top quark mass threshold is also taken into account. This is the reason `mb(mb)` and `Mtop(pole)` are included in this menu.
- `order` : the loop order of the running. Choices are `LO`, `NLO`, and `NNLO`.
- `mb(mb)` : the pole mass of the bottom quark.
- `Mtop(pole)` : the pole mass of the top quark.
- `Q[GeV]` : the scale of the calculation. More details can be found below.

The QCD scale typically depends on the momenta of the particles. In CalcHEP, it can be defined as an algebraic expression which includes floating point numbers, model parameters and the following primitive phase space functions:

- `Sij` : gives $(p_i + p_j)^2$, the invariant mass squared of particles `i` and `j` which must satisfy `i, j ≤ nin + nout`.
- `Mij` : gives $\sqrt{(p_i + p_j)^2}$, the invariant mass of particles `i` and `j` which must satisfy either `nin < i, j ≤ nin + nout` or `i, j ≤ nin`.
- `Ti` : gives $\sqrt{(p_i^x)^2 + (p_i^y)^2}$, the transverse momentum of particle `i`.
- `Mi` : gives M_i the mass of particle `i`.
- `Wij` : gives the transverse mass of particles `i` and `j` which must satisfy `i, j > nin`. The definition of the transverse mass can be found in Section 5.6.
- `Zi` : gives $\sqrt{Ti^2 + Mi^2}$, the transverse energy of particle `i`.

For example, a popular choice of scale based on the Mandelstam variables is $\sqrt{2stu/(s^2 + t^2 + u^2)}$ and can be realized by the following function in CalcHEP :

```
Q[GeV] = sqrt(2*S12*S13*S14/(S12^2+S13^2+S14^2))
```

The `min` and `max` functions can also be used with an arbitrary number of arguments as can standard mathematical functions. Whatever function is defined for the scale of the calculation, CalcHEP sets the minimum value as 1 GeV.

The last menu item in the `QCD coupling` menu is `Alpha(Q) plot` which allows to see a plot of the strong coupling over a range of QCD scales.

When parton distribution functions are used for the initial state particles, it is preferable to use the strong coupling constant defined in the parton distribution function instead of the internal CalcHEP value. This can be controlled by using the `parton dist. alpha` item of the `QCD coupling` menu. It can take the value `ON` or `OFF`. The default is `ON` (the strong coupling constant defined in the parton distribution function is used.) However, if the parton distribution functions are not being used or if they do not define the strong coupling constant, CalcHEP displays `!ON` in place of `ON` and uses its own internal value of the strong coupling constant.

5.5 Breit-Wigner propagator

The propagator denominator of a particle at tree level is given by

$$\frac{1}{p^2 - m^2} ,$$

where m is the particle mass and has a pole at $p^2 = m^2$. If this pole is inside the phase space volume being integrated over, it causes the integral to diverge. At higher order, the propagator denominator is modified to become [48]:

$$\frac{1}{p^2 - m^2 - i\Gamma(p^2)m} .$$

where $\Gamma(m^2)$ is the width of the particle (the inverse of the particle's mean lifetime). This removes the pole and renders the integral convergent. Since the $\Gamma(p^2)$ terms only dominates this propagator denominator near the pole (and is a small correction far from the pole), this propagator is well approximated by replacing $\Gamma(p^2)$ with $\Gamma = \Gamma(m^2)$ which gives the Breit-Wigner propagator denominator

$$\frac{1}{p^2 - m^2 - i\Gamma m} .$$

This is the propagator denominator used in CalcHEP. However, because we are using a width, which comes from higher order corrections, in a tree-level calculation, it has the potential to violate gauge invariance in the calculation and ruin the large cancellations that sometimes occur between diagrams. There are three different regimes to consider. In the first regime, the particle is exactly on shell. In this regime the calculation is exactly gauge invariant and there is no problem. In the second regime, the particle is off shell, but not very far from on shell. In this regime, the process is still dominated by the resonant diagrams and the effect of gauge invariance breaking is still small. In the third regime, the width is not needed to regularize the integral. So, gauge invariance can be satisfied by not including the width.

The **Breit-Wigner** menu allows to adjust the properties of the propagator denominators used in the phase space integrals. The first menu item is **BreitWigner range** which allows to set the regions where the width is used, by adjusting the value of R . CalcHEP then uses the width in the propagators for $|p^2 - m^2| < Rm\Gamma$. No width is used for $|p^2 - m^2| > \sqrt{R^2 + 1}m\Gamma$. And, in the intermediate region, the propagator is replaced with a constant that interpolates between the two regions. We find that the default value of $R = 2.7$ leads to a difference of $\sim 0.2\%$ in the integral of the squared propagator between the modified propagator described in this paragraph and the propagator with a constant width for the entire momentum range.

The second menu item is **T-channel widths** and allows to turn the width on in the t-channel propagators. The default is to not include these since they are not required to regularize the integration and these propagators never go on shell. Note that, by default, the symbolic session does not include the widths in the t-channel propagators. In order to turn this on, the user must also turn it on during the symbolic session (see Section 4.6.)

The last two menu items (**GI in t-channel** and **GI in s-channel**) control whether CalcHEP applies another method of restoring gauge invariance described in [49, 50]. The diagrams which do not contain the resonant propagator are multiplied by the factors

$$\frac{(p^2 - m^2)^2}{(p^2 - m^2)^2 + (\Gamma m)^2} \quad (2)$$

And in diagrams which contain only the single power of propagator, this propagator is replaced by

$$\frac{(p^2 - m^2)}{(p^2 - m^2)^2 + (\Gamma m)^2} \quad (3)$$

expression.

This modification corresponds to the symbolic summation of all diagram contributions at a common denominator expression with subsequent substitution of the width term into the factored denominator. The trick allows to keep all gauge-motivated cancellations. As a defect of the trick it should be mentioned that the factor (2) kills contributions of non-pole diagrams in the $p^2 = m^2$ point [51].

5.6 Kinematical Functions

CalcHEP allows a very large class of kinematic functions that can be used for cuts and distributions. Firstly, it defines many popular kinematical functions which we describe in this section. Secondly, it allows the user to code any kinematical function in a *C* code file (see Section 5.7.) In this way, any cut and/or distribution can be achieved.

The built-in kinematical functions are called with the syntax

`Name[^, _](P1[,P2,P3...])`,

where **Name** is one capital letter, the `^` and `_` are optional (and will be described below) and **P1**, **P2**, etc. are (anti)particles. The available functions are:

- A** `A(P1[,P2,...])` : gives the angle between **P1** and the combined momentum $p_{P2} + p_{P3} + \dots$. If only one particle is specified, as in `A(P1)`, the angle between **P1** and the first incoming particle is returned. The angle is given in degrees.
- C** `C(P1[,P2,...])` : gives the cosine of the angle defined above for `A(P1[,P2,...])`.
- J** `J(P1,P2)` : gives the jet cone angle between **P1** and **P2**. The jet cone angle is defined as $\sqrt{\Delta y^2 + \Delta \varphi^2}$, where Δy is the difference in pseudo-rapidity and $\Delta \varphi$ is the difference in azimuth angle between **P1** and **P2**.
- E** `E(P1[,P2,...])` : gives the energy of the combined momentum $p_{P1} + p_{P2} + \dots$.
- M** `M(P1[,P2,...])` : gives the invariant mass of the combined momentum $p_{P1} + p_{P2} + \dots$.
- P** `P(P1,P2[,P3,...])` : first boosts into the cms frame of **P1**, **P2**[,**P3**,...] and then takes the cosine of the angle between **P1** (in the cms frame) and the boost direction.

T $T(P1[,P2,\dots])$: gives the transverse momentum of the combined momentum $p_{P1} + p_{P2} + \dots$.

Y $Y(P1[,P2,\dots])$: gives the rapidity of the combined momentum $p_{P1} + p_{P2} + \dots$.

N $N(P1[,P2,\dots])$: gives the pseudo-rapidity of the combined momentum $p_{P1} + p_{P2} + \dots$.

W $W(P1[,P2,\dots])$: gives the transverse mass of the particle set $S = \{P1,P2,\dots\}$ given by

$$\sqrt{\left(\sum_{i \in S} \sqrt{m_i^2 + (p_i^T)^2}\right)^2 - \left(\sum_{i \in S} \vec{p}_i^T\right)^2}$$

where m_i and p_i^T are the mass and transverse momentum respectively of the i th particle.

For example, $M(e,e)$ returns the invariant mass of an electron and a positron in the final state.

The keyword **Jet** can be used instead of a (anti)particle name and is an alias for a gluon or any of the first 5 quarks. If the kinematical lists the same particle more than once, such as in the case of $J(\text{Jet}, \text{Jet})$ or $M(e, e)$, CalcHEP constructs the kinematical variable for distinct final state particles. So, $J(\text{Jet}, \text{Jet})$ means the jet cone angle of two distinct **Jet** particles with different momenta and $M(e, e)$ means the invariant mass of two electrons with different momenta.

For some processes and kinematical variables, there are multiple ways the final state particles can be assigned. For example, consider the process $p, p \rightarrow A, A, A$ and the kinematical variable $E(A)$ which could be applied to any of the photons. Sections 5.8 and 5.11 describe how this is handled for cuts and distributions. Here we describe the use of the \wedge and $_$. For any kinematic function, \wedge causes the highest value to be returned while $_$ causes the lowest value to be returned. For our three photon example, $E^\wedge(A)$ returns the highest of the three energies and $E_ (A)$ returns the lowest of the three energies.

There are two additional types of kinematical variables that do not follow the patterns described so far. The first is **M12** which returns the invariant mass of the two initial state particles $\sqrt{(p_1 + p_2)^2}$. The second is user defined kinematical variables. User defined functions always begin with **U**. The rest of the name can be anything the user likes. For example, the user could define the function **xyz** in which case the kinematical variable would be written as **Uxyz**. Further details can be found in Section 5.7.

5.7 User Defined Functions

There are two *C* code functions that the user can write to modify the results of the numerical session. The first prototype is

```
double usrFF(int nIn,int nOut,double*pvect,char**pName,int*pCode)
```

which multiplies the squared matrix element for each phase space point. This could, for example, be used to implement a K factor to approximate the effect of loops. If the user does not define this function, CalcHEP uses a dummy version of this function which always returns 1.

The second function prototype is

```
double usrfun(char*name,int nIn,int nOut,double*pvect,char**pName,int*pCode)
```

which allows the user to write his/her own kinematical functions to be used in the cuts and distributions. When the user enters U<name> for a cut or a histogram, CalcHEP calls `usrfun(<name>,...)`. For example, if the user enters `Ua1b2` in a cut table, CalcHEP calls `usrfun("a1b2",...)`. It is then the responsibility of the user to write the code that calculates the kinematical variable and returns the value. In the absence of user code, CalcHEP contains a dummy version of this function which prints an error message to `stderr` and terminates the numerical session.

After the user writes these functions, the user must add the full path to his/her code in the **Libraries** table of the model. (Details can be found in Section 4.1.) If applicable, the environment variables `$CALCHEP` and `$WORK` can be used as part of the path which are defined by CalcHEP. (Other environment variables can also be used.)

We will now describe the other parameters of these functions. `nIn` and `nOut` are the number of incoming and outgoing particles, respectively. `pvect` is a one-dimensional array that contains the momenta of the external particles. The *j*th component of the *i*th particle's momentum is given by $p_i^j = \text{pvect}[4*i+j]$. The energy is the 0th component $E_i = \text{pvect}[4*i]$ and the momentum along the axis of collision is given by $p_{zi} = \text{pvect}[4*i+3]$. `pName[i]` and `pCode[i]` give the name and PDG code of the *i*th particle, respectively.

Two more functions which the user may find helpful in writing his/her code are

```
int findval(char*name, double *value);  
int qnumbers(char*pname,int*spin2,int*charge3,int*cdim);
```

The first gives the value of the model parameter specified by its name. Both independent and dependent parameter values can be obtained in this way. If the parameter given by `name` is found in the model, then `findval` returns

0 and fills `value` with the numerical value of the parameter. The `qnumbers` function gives the particle's quantum numbers. The particle is specified by name and the pointers `spin2`, `charge3` and `cdim` are filled with twice the particle's spin, three times the particle's charge and dimension of the particle's color representation. This function returns the particle's PDG code. If the parameter name cannot be found, it returns 0.

The `$CalcHEP/utile` directory contains examples of the `usrFF()` and `usrfun()` functions which can be modified to suit the needs of the user.

5.8 Cuts

Cuts can be entered by choosing the `Cuts` menu item. This opens a table where the cuts can be defined. There are four columns in this table. The first column can contain a `%`, a `!` or can be empty. A `%` means that the cut should be ignored. A `!` means the inverse cut (for each particle combination). The second column accepts a kinematical function (see Section 5.6.) The third and fourth columns take the minimum and maximum values, respectively, for the kinematical variable. If the third or fourth columns are empty then the minimum or maximum limits, respectively, are not applied. These limits may contain numerical values, model parameters, standard algebraic expressions (including `+`, `-`, `*`, `/` and `^`) and functions defined in the *C* math library (such as `sqrt()`, `sin()` and `cos()`.)

If the process contains identical outgoing particles, the cut is applied to each particle combination. For example, for the process $pp \rightarrow AA$, the cut

Parameter	> Min bound <	> Max bound <
E(A)	20	100

is equivalent to

Parameter	> Min bound <	> Max bound <
E^(A)		100
E_(A)	20	

On the other hand, the same cut, but with an exclamation mark in the first column

Parameter	> Min bound <	> Max bound <
E(A)	20	100

demands the absence of photons with energy in the 20 - 100 GeV interval. For processes with several identical particles, a cut marked by `!` does not mean

the mathematical negation of the condition without the exclamation mark. It means the negation of each individual cut for each particle combination.

If a cut contains particles which are not included in the current subprocess, they are ignored until the user starts to work with a subprocess which does include them.

5.9 Kinematics

The **Kinematics** menu subitem of the **Phase space mapping** menu item allows to display and change the phase space parameterization used in the Monte Carlo integration. The way this is done in CalcHEP is that it continually splits the remaining particles into two sets until each particle set contains one particle. In this framework, the multi-particle phase space is parameterized by the invariant masses of each particle set and by the two-dimensional spherical angles of the $1 \rightarrow 2$ splitting [52, 53]. Since the choice of the kinematics influences the phase space mapping, it also affects the convergence of the Monte Carlo integration. In other words, a mapping choice that is related to the physical problem is more likely to converge efficiently than one that is not. See Appendix I.1 for further details.

Upon entering the kinematics menu item, the current phase space mapping scheme is displayed along with a dialogue asking the user whether he/she would like to change it. If the user answers Y, CalcHEP will request the splittings one at a time. For each splitting, CalcHEP presents the user with the particle set that requires splitting (or 12 if the first splitting which means all the final state particles). The user then splits these particles into two groups and enters them separated by a comma. For example, suppose we are considering a $2 \rightarrow 4$ process. The default splitting is:

12 -> 3 , 456
 456 -> 4 , 56
 56 -> 5 , 6

However, the user might change this to:

12 -> **34** , 56
 34 -> 3 , 4
 56 -> 5 , 6

where the **34** is what the user enters.

5.10 Regularization

Generally speaking, the Monte Carlo integration of squared matrix elements does not converge well because of the presence of the singular propagators. Even after including the widths, the convergence may not be optimal. It can

be improved by doing a phase space transformation which smooths the sharp peaks of the squared matrix element (see Appendix I.1 for further details.) The resonances can come in the forms

$$\frac{1}{p^2 - m^2} \quad (4)$$

$$\frac{1}{(p^2 - m^2)^2} \quad (5)$$

$$\frac{1}{(p^2 - m^2)^2 + (m\Gamma)^2} \quad (6)$$

where m , Γ and p are the mass, width and momentum of the virtual particle, respectively.

In order for CalcHEP to regularize these propagators, it needs to know the position of the resonances. These can be entered via the **Regularization** subitem of the **Phase space mapping** menu item which opens a table editor allowing to enter the position of these resonances. This table has four columns allowing to specify the momentum, mass, width and power of the resonance denominator.

The momentum of the resonant particle can be expressed as a sum of the external momenta. The user can type the numbers of the particles which should be added together to get the resonant momentum. Initial state momenta are added to each other and final state momenta are added to each other, but initial and final state momenta are subtracted from each other. For example, the entry of **12** means the resonance occurs at $(p_1 + p_2)^2 = m^2$, the entry of **34** means the resonance occurs at $(p_3 + p_4)^2 = m^2$ and the entry of **134** means the resonance occurs at $(p_1 - p_3 - p_4)^2 = m^2$.

The mass and width of the resonance are entered in the second and third columns. They can include numerical values, model parameters and algebraic expressions. Typically they should simply be the model parameters which specify the mass and width of the resonant particle.

The last column is for the power of the denominator of the propagator that appears in the squared matrix element which can be either 1 or 2. Of course, in a squared matrix element, resonant propagators appear to the second power. However, there are times that the gauge cancellations allow the exponent to be effectively decreased to 1. This typically only happens when the resonance is stable (its width is 0). If the resonant particle is unstable and its width is nonzero, CalcHEP uses 2 and ignores the user's input.

5.11 Monte-Carlo simulation

The integration of the multiparticle phase space is done by the *Vegas* Monte Carlo routine [46, 47] (see Section I.2 for further details.) The **Monte-Carlo**

simulation menu allows the user to control the integration. It, also, allows the user to set up histograms to be filled during the integration (explained below) and generate unweighted events (see the Subsection 5.12.)

Vegas runs N_{sess} sessions. After each session, if the grid is not frozen, *Vegas* improves the grid so that the integral of the next session converges more efficiently. The number of sessions run by *Vegas* is controlled by the menu item **nSess**. The greater this number, the more likely *Vegas* is to find a satisfactory grid. Typical values for **nSess** are between 5 and 10. The default is 5.

During each session, *Vegas* calculates the integrand N_{calls} times. This is controlled by the menu item **nCalls**. Greater values of N_{calls} give better estimates for the integral and allow for better improvement of the grid. The optimal value depends on the process being analyzed. Processes with more final state particles typically need larger values of N_{calls} whereas processes with fewer final state particles converge with smaller N_{calls} . The default value of **nCalls** is 10000.

Vegas begins calculating the integral and improving the grid when the menu item **Start integration** is chosen. During the integration, the status of each session is displayed along with the integration results of previous sessions. These results include the integral estimate, the uncertainty estimate and the estimated efficiency of the event generator if the grid is frozen. When *Vegas* finishes, it makes a final estimate of the total integral, uncertainty and χ^2 . It is usually a good idea to achieve uncertainties of approximately 1% or better for individual sessions. After *Vegas* finishes, the user can adjust the *Vegas* parameters and/or run *Vegas* again until satisfactory results are obtained. Unless the statistics are cleared, the new *Vegas* results are combined with the previous results.

After the grid is improved, it is a good idea to clear the statistics before calculating the final integral and distributions. This is done via the **Clear statistics** menu item. The grid is usually well adjusted when the Monte Carlo uncertainty stabilizes at or below approximately 1%. Once the grid is improved, the user can, optionally, freeze the grid via the **Freeze grid** menu item so that *Vegas* does not further adjust it. Another benefit of freezing the grid is that the event generator will be prepared during the *Vegas* sessions. In some cases, it is desirable to start with a fresh grid. This can be done by the **Clear grid** menu item.

CalcHEP has facilities to generate kinematical distributions during the *Vegas* sessions. The user can specify which distributions he/she would like by choosing **Set Distributions**. A table with 6 columns will open. The first column is where the user specifies the kinematical variable to be histogrammed. The kinematical variables available are described in Subsec-

tion 5.6. The second and third columns are the minimum and maximum values of the histogram. If a 1-dimensional distribution is desired, the last three columns should be left blank. If a 2-dimensional distribution is desired, another kinematical variable, a minimum and a maximum can be entered in the last three columns. Multiple distributions can be entered, one per line. The minima and maxima can contain numbers, model parameters, algebraic expressions and standard math functions from the *C* math library. If there are multiple ways the final state particles fit the kinematical variable, each is added to the distribution (the distribution is a sum of each possibility.)

The user can continue to run *Vegas* until he/she is satisfied with the uncertainties in the distribution. The distributions can be viewed by entering the **Display Distributions** menu. A list of the distributions will be presented and the user can choose the one he/she would like to view. After allowing the user to choose the number of bins for the histogram, CalcHEP will display it using the **Plot Viewer** described in Section 3. An example of a distribution is presented in Fig. 4.

The distribution data is stored in the file `distr_N` where *N* corresponds with the CalcHEP session number (not the *Vegas* session number.) The user can view the distributions from previous CalcHEP sessions by use of the `show_distr` program located in the `bin` subdirectory of the user's work directory. For example,

```
$CALCHEP/bin/show_distr distr_1
```

would display the distributions from the CalcHEP session number 1.

5.12 Event Generation

CalcHEP can generate unweighted events. These events are useful in simulations of particle physics collisions and can be passed to other programs for further analysis. For example, it is often desirable to pass events through PYTHIA [54] which hadronizes colored final states and adds radiation to the event. Event generation is done in the **Monte-Carlo simulation** menu. It consists of two steps. In the first step, the generator is prepared and in the second, the events are generated. Details of the algorithms used and the format of the event files can be found in Appendices F, I.3 and I.4.

Generator preparation. The efficiency of the event generator depends on the number of phase space cubes and the estimation of the maximum differential cross section (or partial width) in each cube. Since the differential cross section (or partial width) can vary greatly from phase space point to

point (especially around a resonance), a larger number of phase space cubes allows for a more efficient generator. The menu item **Event cubes** allows to modify this parameter. On the other hand, for each phase space cube, it is very important to have a good estimate of the maximum for that cube. The maximum is searched for during *Vegas* integration sessions when the grid is frozen. The more phase space points *Vegas* generates, the more likely the estimate of the maximum is accurate. On the other hand, the greater the number of generator phase space cubes, the longer *Vegas* will have to run to find the maximum for each cube.

When a user would like to generate events, he/she will typically begin by improving the *Vegas* grid as described in Subsection 5.11. Once the grids are optimized, they should be frozen and *Vegas* should be run again. After each *Vegas* session, the current estimate of the event generation efficiency will be displayed along with the cross section or particle width. *Vegas* should be run until the event generation efficiency converges. If the final efficiency is too low, the number of phase space cubes should be increased via the **Event cubes** menu item and *Vegas* should be run again. This process can be continued until a satisfactory efficiency is achieved. The user must balance obtaining a high efficiency against the time it takes to estimate the maxima for a large number of phase space cubes.

Event generation. When the generator is prepared and the efficiency is acceptable, the user can enter the **Generate Events** menu under **Monte-Carlo Simulation**. The first item on this menu, **Number of events**, allows to specify the number of events to generate. Event generation is started by the menu item **Launch generator**. During event generation, if an event is ever produced with a differential cross section (or partial width) which is greater than the estimated maximum value in that cube then two things happen. The first is that the event is split into multiple unweighted events. The second is that the maximum for that phase space cube is increased in order to prevent this from occurring in the future.

When CalcHEP finishes generating the events, it displays an informational message which states the number of events generated, the actual efficiency, the number of multiple events generated and the number of events with negative weight generated. The user is asked whether he/she would like to accept these events. If there are many multiple events, it is a good idea to improve the generator further and try again. We recommend the user test the generator with a small number of events before generating the full desired set of events.

Once the user is satisfied with the events, he/she can answer y to accept

them. The events will be written in plain text to the file `events_N.txt` where `N` is the CalcHEP session number. A full description of the event format can be found in Appendix I.4. These events can be converted to the Les Houches Accord format by use of the `event_mixer` program found in the `bin` directory. `event_mixer` is further described in Section 6.2.

Event analysis. In addition to many external programs that can analyze events, CalcHEP contains a program which can histogram the events and generate a distribution. Its name is `events2tab` and it is stored in the `bin` directory. For example, it can be run as

```
$CALCHEP/bin/events2tab <var> <min> <max> <N> < <evnts> > <plt>
```

where `<var>` is the kinematical variable to be histogrammed (and must be in quotation marks), `<min>` and `<max>` are the minimum and maximum values of the distribution, and `<N>` is the number of bins for the histogram. The events are given to this program via the redirection operator `<` and `<evnts>` is the event file. The output is the distribution data and would typically be redirected by the `>` operator to the file `<plt>`. For example, `$CALCHEP/bin/events2tab "M(b,B)" 0 200 100 < events_1.txt > plot_1.txt` will read the events from the file `events_1.txt`, generate the $M(b,B)$ distribution with minimum value of 0, maximum value of 200 and 100 bins and write the plot data to the file `plot_1.txt`. The resulting plot can be viewed by use of the program `plot_view` which is stored in the `bin` directory. This will display the plot using the Plot Viewer (see Section 3.)

5.13 Simpson Integration

In the case of $2 \rightarrow 2$ processes, the integration is one-dimensional and standard one-dimensional integration techniques are used. This method is very fast and very accurate. It can be accessed via the `1D integration` menu item which brings the user to Menu 8 (see Fig. 9.)

We note, however, that certain user settings are ignored when using the `1D integration` method. The structure functions are not included, the center-of-mass rapidity is set to zero, the regularizations are ignored and all the cuts are ignored. In fact, the only cut allowed in the Simpson integration is on the cosine of the angle between the third particle (the first outgoing particle) and the first incoming particle. We call this `cos13`. The user can modify the minimum and maximum values of this kinematical variable via the `Cos13(min)` and the `Cos13(max)` menu items. This cut is often necessary to remove T-channel singularities from massless propagators (such as a T-channel photon in $e^+e^+ \rightarrow e^+e^+$). The default cut is

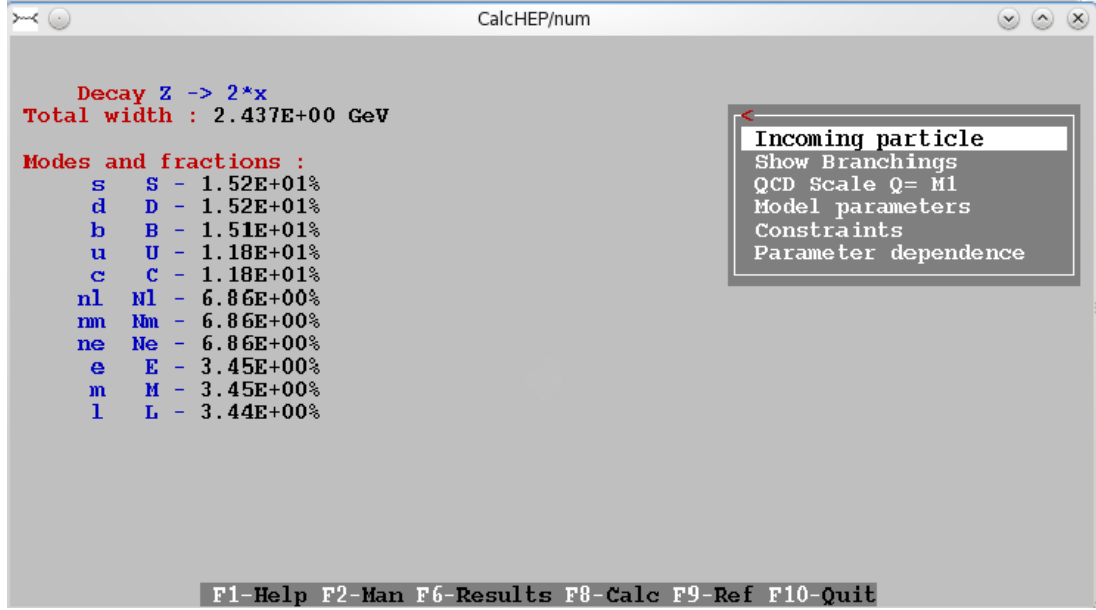


Figure 11: An example of the Easy 1->2 menu for the two particle decays of the SM Z boson.

$$-0.999 < \cos 13 < 0.999$$

Upon successful integration, the cross section is displayed on the screen. The precision of the calculation can be set via the **Set precision** menu item. The default value is 10^{-4} . The angular dependence of the differential cross section can be viewed via the **Angular dependence** menu item. Plots of the dependence of the cross section on the model parameters or the center-of-mass energy can be viewed by the **Parameter dependence** menu item. This menu also provides **sigma*v** plots for $v \times \sigma(v)$ at $v \rightarrow 0$. Here v is relative velocity of particles. This characteristic is useful for astroparticle applications, in particular for the estimations of the elastic scattering of Dark Matter candidates.

5.14 Two Particle Decays

$1 \rightarrow 2$ decays do not require numerical integration at all. It can be done symbolically. In this case, CalcHEP displays the **Easy 1->2** menu item which allows to view the widths and branching ratio. An example of this screen can be seen in Figure 11.

If multiple decay channels are generated during the symbolic session (for example, if the user specifies the process as $Z \rightarrow 2*x$), each nonzero decay

channel will be listed on this screen along with either its partial width or branching fraction. The default listing is with branching fractions, but the user can switch between the two via the **Show Branchings/Show Partial widths** menu item. The total width for the particle (the sum of the partial widths) is displayed at the top of the screen.

The dependence of the branching ratios and the width on the model parameters can be viewed by using the **Parameter dependence** menu item. This will open a menu allowing to choose either the total width or one of the decay branching ratios. It will then allow to choose the model parameter and display the dependence using the **Plot Viewer**.

The width of the Higgs particle depends sensitively on the effective quark masses which can accumulate higher order QCD corrections to their Yukawa couplings. In the built-in version of the Standard Model, we assume that these masses depend on the model parameter Q . The correct width is obtained if Q is set equal to the mass of the particle that is decaying (e.g. the Higgs). This is the default value of Q for a decay. But, the user can adjust this, or view the dependence of the width on this parameter as desired. For other models to enjoy this feature, the masses must be implemented in a similar fashion.

6 Collecting Subprocesses

The CalcHEP interactive sessions are designed to run one subprocess at a time. However, many typical collider processes contain many subprocesses that differ only by the initial state and/or final state particles. For example, at the LHC, the initial states are two colliding protons which, however, are composed of quarks, antiquarks and gluons. In this section, we describe routines that combine the results from different subprocesses.

6.1 Distribution Summation

The `bin` directory contains the program `sum_distr` which combines distributions from different CalcHEP sessions (typically from different subprocesses.) The way it works is that it sums the distributions that have exactly the same kinematical variable specification. In other words, $M(b,B)$ and $M(B,b)$ would not be combined although they are the same distribution. $M(u,d)$ would not be combined with $M(u,s)$ although they are both the invariant mass of two jet particles. $M(\text{Jet},\text{Jet})$ would be combined from different subprocesses, however, because they have exactly the same specification. The user must make sure each distribution specification is the same for each subprocess and that there is no ambiguity.

The distributions from each CalcHEP session are stored in the files `distr_N` where `N` is the CalcHEP session number. To sum the distributions from the subprocesses, the user would typically run

```
$CALCHEP/bin/sum_distr <distr_N1> <distr_N2> ... > <distr_out>
```

where `<distr_N1>`, `<distr_N2>` and `...` are the distribution files to be combined and `<distr_out>` is the file where the results should be written. The program `show_distr` can be used on the output file in the same way as for the distribution files written during the interactive session.

6.2 Event Mixing and LHEF

As described in Subsection 5.12, the interactive numerical session can write events for each subprocess and for each decay. However, it is often desirable to combine these events and connect production events with decay events so that the final events are fully decayed. The program `event_mixer` does this. The parameters of this program are the number of final events to produce and a list of the directories where the event files can be found. For example,

```
$CALCHEP/bin/event_mixer <N> <dir1> <dir2> ...
```

where `<N>` is the total number of events to produce, `<dir1>`, `<dir2>` and ... are the directories where the event files produced by the interactive numerical session are stored. `event_mixer` searches these directories for event files to mix.

Two other files are used by `event_mixer`. The first is the file `decaySLHA.txt` which contains a list of the particles masses, quantum numbers, widths and decay channels in SLHA format [37,55]. The user can generate this file during the interactive session by use of the `Constraints`, then the `Masses,Widths,Branching` and then the `All Particles` menu items. See Subsection 4.2 for further details. The `decaySLHA.txt` file is used to determine the total widths of the particles that are decaying in order to determine the branching ratios and the final cross section. If this particle is missing, `event_mixer` will use the current experimental values for the particles of the SM that have been discovered. For the Higgs particle and particles beyond the SM, `event_mixer` will assume that all the decay channels are present in the event files and estimate the total width from them.

The `run_details.txt` file contains information about the events and is placed by `event_mixer` in the header of the resulting event file. For example, this file could contain the parameter values, the center of mass energy, the parton distribution functions and so on.

`event_mixer` randomly mixes production events and their decays according to their cross sections and branching ratios. It does this until the requested number of events is generated or until it runs out of production events in any of the files. Before it mixes the events, it writes to screen the final cross section and the maximum number of events that can be generated. For example,

```
2.368E-01  -total cross section[pb]
10098      -maximum number of events
```

To get this number before mixing the events, simply request 0 events.

The results of `event_mixer` are stored in the file `event_mixer.lhe`. This file is written in the LHE file format with an XML header [41] and additional sections written in HepML [56] format. This format allows to automatically upload the LHE file to the CERN Monte-Carlo Database (MCDB) using the command

```
./upload2mcdb_hepml.pl -header hepml event_mixer.lhe
```

where the `upload2mcdb_hepml.pl` script can be downloaded from the MCDB website (<https://mcdb.cern.ch>.)

Some special features of LHE file generated by CalcHEP are:

- A history of each decay is presented for each event. The information about the parent particles and their mean life time is included. This information can be used for proper hadronisation and detector simulation.
- When connecting decays, `event_mixer` uses a Breit-Wigner virtual mass distribution, where we assume that the matrix elements of the subprocesses do not depend strongly on the off-shell momentum. Our procedure does not break momentum conservation.
- According to the LHE file format accord, the header (marked by `<header>` and `</header>`) section can be used for auxiliary information. `event_mixer` places the following in the header: a `<hepml>` section (see below), a `<slha>` section with information about quantum numbers, masses widths and decays of non-SM particles, and a `<calchep-batch>` section for the 'run_details.txt' file.
- Information about the process such as a list of the subprocesses, kinematical cuts, model name, number of generated events, cross sections and model parameters is stored in the `<hepml>` section. For instance,

```

<files>
  <file>
    <eventsNumber>          1000      </eventsNumber>
    <crossSection unit="pb"> 0.254087 </crossSection>
  </file>
</files>

```

This information is recognised when the LHE files are being uploaded in the MCDB data base and is used for automatically creating the 'article' in MCDB.

The routine `lhe2tab` histograms the events in an LHE file. It is called in a similar fashion to `events2tab` described in Subsection 5.12. For example:

```
$CALCHEP/bin/lhe2tab <var> <min> <max> <N> < <lhe> > <plot.txt>
```

where `<var>` is the kinematical variable, `<min>` and `<max>` are the minimum and maximum values for the distribution, `<N>` is the number of bins, `<lhe>` is the LHE event file, and `<plot.txt>` is the output file where the histogram data should be written. The only difference with respect to `events2tab` is that the PDG number of the particles should be used rather than the names of the particles. For example, `M(5,-5)` should be used in place of `M(b,B)` for

the invariant mass of the b-quark and the anti-b-quark. The output file also contains a line which records the largest deviation from energy momentum conservation. An example is

```
#lost_momenta_max/Etot 7.9E-11 1.3E-12 1.3E-12 8.0E-11
```

Typical value should be approximately 10^{-10} because the original event files recorded 11 digits for the particle momenta. This allows the user to test whether energy and momentum were conserved in the mixing.

The CalcHEP batch interface (see Section 7), `event_mixer` is automatically called at the end of the batch session to construct the resulting event file.

6.3 N-tuples.

CalcHEP contains a program that creates PAW NTUPLES from LHE files. An example of using this program is

```
$CALCHEP/bin/nt_maker <events.lhe>
```

where `<events.lhe>` is the file containing the LHE events.

7 Batch Mode

Initially CalcHEP was designed for interactive calculations with a graphical user interface. However, there are times when a batch system is ideal. For example, when a calculation takes a very long time, or the user is interested in doing scans over parameter space. Users of computers where parallelization is possible. In these situations, it is preferable to set up the program with a set of batch instructions and run it in the background.

CalcHEP provides the user with the ability to record their keystrokes and then use it as an instruction set in a batch mode. The recording step is done by using the `+blind` option. When this option is used, CalcHEP opens in interactive mode and records all the keystrokes the user performs. When the user exits, CalcHEP prints to the screen a string which represents the sequence of keystrokes. The user can then call the program with the `-blind` option along with the keystroke string to initiate the same calculation in batch mode. All the results of the calculations are stored in the same places as in interactive mode, allowing the user to use the results in the same way as in the interactive mode.

This batch option has proved to be very powerful, but also very difficult for users to take full advantage of. Some challenges are:

- If the user wishes to perform a complicated calculation that takes a very long time, it is not possible to quickly create the key sequence string using the `+blind` option. In this situation, the user must become adept at writing or modifying the key sequence string to achieve the desired results.
- The symbolic CalcHEP session does not always begin at the same point. The entry point depends on the previous session. Correspondingly, the key sequence required to achieve the desired result depends on the previous session and can be very difficult for the user to control.
- CalcHEP sometimes presents the user with a dialog that requires some form of input (such as “Press any key” or *Yes/No*). In batch mode, all such dialogs are skipped and *Yes/No* questions are automatically answered with *Yes*.
- If the user has a mistake in their keystroke sequence string or if there is a problem in the setup of the session, the batch mode can not interact with the user to fix the problem. CalcHEP simply quits with an error code meant to inform the user about the problem.

- Some interactive menus depend on the physical problem being analyzed. For example, the position of the t -quark mass could be in different locations in the parameter menu and different keystroke sequences would be required to change its value. In the present version, we have solved this by implementing a *Find* menu option which allows the user to type the name of the parameter and be taken directly to it.
- Scans over parameter space are not directly supported by this keystroke sequence batch mode.

In the present version of CalcHEP, we have attempted to streamline this process and provide the user more powerful and convenient ways to use CalcHEP in batch mode. In this section, after reviewing how to use the batch mode for the symbolic and numerical sessions, we give details of a variety of shell scripts which allow the user to control CalcHEP in blind mode without the need to write keystroke sequence strings. The final subsection details a new Perl interface to CalcHEP which automates the procedure of scanning over parameters and parallelizing the calculations.

Before continuing with the details of the improvements, we note that the presence and interrelated nature of the interactive and the batch regime facilitates gives the user the ability to set everything up in the interactive regime where the user can see the results and check that everything is working properly and then run the long calculation in batch mode. This combines the advantages of the interactive and batch modes.

7.1 Blind mode

As mentioned, CalcHEP has an option `-blind` which allows the user to run in batch mode. It can be used with both the symbolic and numerical code as in:

```
s_calchep -blind "STRING"
n_calchep -blind "STRING"
```

where **STRING** is a string which represents the sequence of keystrokes required which would be used in an interactive session to achieve the desired result. In this string, each printing character (e.g. alphabetical characters) is represented by itself while up, down, enter and escape are represented by `]`, `[`, `{`, and `}` respectively. The response of the numerical characters depends on where in the interactive session they appear. If alphanumeric input is required of the user, such as the input of a process, cut, distribution, etc., a numeric characters is treated as the number it represents. However, if alphanumeric input is not required, then the numeric characters signal function

]	Up
[Down
{	Enter
}	Escape
\NN	Special keys where NN is the hexadecimal value
0-9	Function keys or numeric input

Table 1: Characters used in blind mode for Up, Down, Enter and Escape. Other special keys are specified by their hexadecimal number 'NN'. The interpretation of numeric characters depends on when they are initiated as discussed in the text. All other characters which print to screen are used directly.

keys. For example, in this situation, 0 corresponds with the 'F10' key which signals the end of the session and instructs CalcHEP to quit. For this reason, keystroke sequence strings are usually terminated with a 0. All other special keystrokes are represented by \NN where NN is the hexadecimal value for the character. For example, the 'Tab' key is represented by \08. For reference, we also include these characters in Table 1.

With these definitions, it is possible to write keystroke sequence strings that perform any desired calculation that can be achieved in an interactive session. However, it can be very difficult, in practice, to create a keystroke sequence string from scratch. For this reason, another option **+blind** was created and is used as in:

```
s_calchep +blind
n_calchep +blind
```

This will open an interactive session where each keystroke the user makes is stored internally. When the user quits, the entire keystroke sequence string is printed to the screen. The user can then copy or modify this string and use it with the **-blind** option.

As an example of this process, suppose the user would like to run vegas with 6 iterations of 100,000 calls, clear the statistics and then run 10 iterations of 1,000,000 calls each. This could take a long time depending on the complexity of the process. A simple way of achieving this is to start **n_calchep** with the **+blind** option. When the interactive session starts, the user would use their keyboard, as usual, to move through the menus and change the values of **nSess_1**, **nCalls_1**, **nSess_2**, and **nCalls_2** but to smaller values which will finish in a reasonable time, for example, 1, 20000,

1, and 20000 respectively. After quitting, CalcHEP prints the following to the screen:

```
"[[[[[[[[{{1{{20000{{1{{20000{{{{0"
```

The user can then simply change the numerical values to those they desire and run with the `-blind` option as in:

```
n_calchep -blind "[[[[[[[{{6{{100000{{10{{1000000{{{{0"
```

CalcHEP will run vegas with the specified settings in the background. Other scenarios can easily be imagined.

We note that the blind mode is used for the automatic width calculation. When CalcHEP need the numerical value of an automatically calculated particle width, it begins by running the symbolic session in blind mode. It then compiles the code as a shared library which is then attached to the currently running program. The micrOMEGAs [42] package is based on this idea too. In this case, all the processes of Dark Matter annihilation are generated on the fly using the blind mode of the symbolic session.

7.2 Shell Scripts

It is possible to write universal shell scripts based on the blind mode which accept user input, create the keystroke sequence string and start calchep in blind mode. This relieves the user from the burden of determining the keystroke sequence string but allows the user to run CalcHEP in batch mode. We provide several such shell scripts for common tasks in the `$CALCHEP/bin` directory which is symbolically linked to the `WORK/bin` directory. In this section, we describe them. If any of these scripts require parameters, but are called without any parameters, the script first prints a message to screen informing the user of the required parameters and then quits.

We begin by describing `s_blind` which does a symbolic calculation. It is called from the users `WORK` directory.

- **s_blind nModel Process nOutput:** This command runs `s_calchep` in blind mode and generates the squared amplitude code for the process `Process` which must be enclosed in quotation marks, as in `"e,e->m,M"`. The model is specified by `nModel` which must be an integer and corresponds with the position of the model in the model list. The first model is specified by 1, the second by 2 and so on. `nOutput` determines the format of the output and is also required to be an integer. The supported outputs are C code, Reduce code,

Mathematica code, and Form code which are specified by 1, 4, 5, and 6 respectively.

There are some further things the user should keep in mind when using this shell script. Because the keystroke sequence required for this shell script depends on the previous session, this script first removes the previous results and starts a fresh session. Feynman gauge is always used by this shell script. There is no possibility to use alias definitions, remove particles or choose diagrams with this script.

This script is used by MicrOMEGAs [21] to generate numerical code at run-time.

We now describe scripts which are designed to work with the numerical session. These are called from the directory where the numerical code `n_calchep` is stored. Typically, this is the `WORK/results` directory, but this directory can be renamed or moved by the user. For each of these scripts, all session parameters are kept fixed except for the ones explicitly described as being changed. Typically, the user would start `n_calchep` in interactive mode and set all the session parameters as desired. The user would then quit the interactive session and run one of these scripts.

- `run_vegas it1 N1 it2 N2`: This script runs the Vegas Monte Carlo integration of the phase space. It runs Vegas `it1` times with `N1` calls each and then `it2` times with `N2` times each. The results are cleared between these two runs. If the user calls this script with all of `it1`, `N1`, `it2`, and `N2` nonzero, Vegas will run `it1` times, clear the results, and then run `it2` times more, the idea being that the first `it1` calls allow Vegas to adapt the grid but the second `it2` calls achieve the actual integration. If `it1` and `N1` are nonzero, but `it2` or `N2` are zero, it will run Vegas `it1` times and quit. This would typically be used if the user is satisfied with the adaptation of the Vegas grid and has already cleared the results but wants to add more statistics to their integration. If `it1` or `N1` are zero but `it2` and `N2` are nonzero, it will first clear the results and then run Vegas `it2` time. This would typically be used if the user has been adapting the grid, but is now satisfied and wants to perform the integration after clearing the statistics.
- `set_vegas it1 N1 it2 N2 nCubes` : This script sets parameters of two loops Vegas calculation which drive script `run_vegas`. Here `it1` is number of Vegas runs and number of integrand calls for the first

loop. `it2` and `nCall2` define the second loop calculations. The `nCubes` parameter defines number of sub-cubes which are used in the second loop for a proper fitting of integrand for efficient event generation. Note that parameters of second loop can not be defined in graphic interface mode. The parameters are stored in 'session.dat' file.

- **run_vegas:** This script launches subsequently two loops of Vegas Monte Carlo integration of the phase space using parameters defined by `set_vegas`. At first loop we allow Vegas to adopt the integration grid. After that all obtained results for cross section and histogram are cleaned and we launch second loop with fixed grid which generates final statistics and prepare integrand fitting for event generation. If `it1=0` or `N1=0` then the only the second loop is running. In the same manner only the first loop is running if `it2=0` or `N2=0`. If only one loop is active intermediate clearing of results is not applied.
`run_vegas` is used by `pcm_cycle`, `name_cycle`, `subproc_cycle`, and `par_scan` scripts presented below.
- **set_momenta p1 p2:** This script updates the momenta of the incoming particles to `p1` and `p2` and then quits.
- **set_param name1 value1 name2 value2 ...:** This script changes the numerical values of one or more of the independent model parameters `name1`, `name2`, etc. to `value1`, `value2`, etc. respectively and then quits.
- **set_param File:** In this case, this script changes the numerical values of the independent model parameters as specified in the file `File`. `File` must have each model parameter on a separate line with the name coming first followed by the new numerical value, separated by white space.
- **pcm_cycle pcm0 step:** This script scans the cross-section over the center of mass energy. For each point in the scan, it updates the momenta of the initial state particles and then runs the Vegas Monte Carlo integration. When it is finished, it writes the resulting cross-sections to the file `pcm_tab_j1_j2` where `j1` is the session number when the script began and `j2` is the session number when it finished where $N=j2-j1+1$. It begins its calculations with the momenta of the initial state particles equal to `pcm0` and `-pcm0` and increases in steps of size `step` for a total of `N` steps. If there are distributions specified then

they are stored in the files `distr_k` where `k` corresponds with the session number when it was generated. In general $j1 \leq k \leq j2$. These distributions can be viewed using the program `disp_dist` contained in the `WORK/bin` directory.

- **name_cycle name val0 step N:** This script scans the cross-section over a model parameter's value. For each point in the scan, it updates the parameter `name` and then calculates the cross-section. When it is finished, it writes the resulting cross-sections to the file `name_tab_j1_j2` where `name` is the name of the parameter, `j1` is the session number when the script began and `j2` is the session number when it finished. Again, $N = j2 - j1 + 1$. The scan begins with the parameter `name=val0` and then increases it by size `step` until `N` steps are completed. As in the previous case, distributions are stored in the files `distr_k` and can be viewed using the `disp_dist` program.
- **subproc_cycle L Nmax:** This script calculates the cross-section and generates events for each subprocess. When it is finished, it adds the cross-sections together and prints the total cross-section to the screen. If there are distributions specified then they are added together and the resulting distribution is stored in the file `distr_j1_j2` where `j1` is the first session number and `j2` is the final session number and where $j2 - j1 + 1$ is equal to the number of subprocesses. It also generates unweighted events for each subprocess. The number it generates is equal to the smaller of the cross-section times the luminosity which is specified by `L` and `Nmax`. It writes these events to the files `events_k.txt` where `k` is the session number when it was generated and, again, $j1 \leq k \leq j2$. These events can be combined using the program `event_mixer` which is stored in the `WORK/bin` directory. The cuts, regularization and histograms must apply to all subprocesses and the outgoing particles must be identical.
- **par_scan < data.txt:** calculates the cross-sections according to the grid for names and parameters given in `data.txt` file. The format of `data.txt` is supposed to be

```

name_1  name_2  ... name_N
val_11  val_12  ... val_1N
.....
val_N1  val_N2  ... val_NN

```

where `name_1 name_2 ... name_N` should be the set of names of independent model parameters, while `val_11 ... val_1N` are values for the respective parameters to be used for the first point and `val_N1 ... val_NN` are values for these parameters for the last grid point of the calculation. Note, that this script does not do summation over the subprocesses i.e. it will do the grid calculation only for chosen subprocess in the menu. The results of the calculation are printed in the terminal in the format

```
name_1 name_2 ... name_N
val_11 val_12 ... val_1N res_1
.....
val_N1 val_N2 ... val_NN res_N
```

or can be redirected into some file, e.g. `results.txt` with `par_scan < data.txt > results.txt` command.

- `par_scan_sum < data.txt`: calculates the cross-sections according to the grid for names and parameters given in `data.txt` file similarly to the `par_scan` one, but in addition it performs a summation over all available subprocesses.
- `gen_events Nevents`: This script can be launched after successful end of `run_vegas` script with active second Vegas loop. Parameter `Nevents` defines number of events to generate.

If any of these scripts ends with an error, a message is printed to `stderr` and the return value of the script can be seen by issuing `echo $?` on the shell. A description of the possible error codes can be found in the CalcHEP manual.

7.3 Batch interface

Although the shell scripts of the previous subsection greatly improve the users ability to run their desired processes in batch mode, there are still some limitations when doing large complex calculations involving scans over parameter space, many subprocesses and parallelization. To overcome these challenges, we have written a Perl script which we call the “batch interface”. The main features of this Perl interface are:

- The input is a pure text file we call the “batch file”. It consists of a series of keywords together with values for those keywords, with each keyword on a separate line. Most of the options available in the interactive session are supported by keywords in the batch file and thus most calculations can be done using the batch interface.
- A library of subprocess numerical codes is utilized. Each time the batch interface is run, it first checks whether the subprocess numerical code exists. If it does, it reuses it and skips the often long process of code generation. Any requested numerical codes not in the library are then generated and added to the library. If the model changed, the numerical codes are regenerated as appropriate.
- The numerical phase space integration is done and events are generated for each subprocess and the results are combined. Production and decay events are connected and the final event output is an LHE file with all the events fully decayed which can be used directly by Pythia or other software.
- Multiple parameters can be scanned over. For each parameter point, the results are combined and stored with names unique to that parameter point for easy retrieval.
- Both the symbolic calculations and the numerical calculations are parallelized. Each subprocess and each parameter point are run as separate jobs and run on all available cpu cores. The number of cores available is set by the user as is the type of cluster software used. Multicore machines, PBS cluters and LSF clusters are currently supported.
- The progress of the calculation is stored in a series of html files which can be viewed in a web browser. These html pages contain information about the progress of the calculation as well as the results of the calculations which are already finished. The final event files are linked as are the session.dat and prt files which give the full details of each individual calculation. Pure text versions of the progress pages are also created for situations where a web browser is not convenient.

Once the user creates the batch file and runs the batch interface, no user input is required until it finishes. It can be run in the background and checked periodically.

After the user has created their batch file, they would typically run the batch interface from their CalcHEP work directory as


```
./calchep_batch batch_file
```

where `batch_file` is the name of their batch file, which can be named anything the user likes. The batch interface will start by printing a message to the shell which will contain the location of the html progress reports which the user can simply copy and paste into their browser url window. The first time the user runs the batch interface, they can also run the following from the work directory

```
./calchep_batch -help
```

which will complain that no batch file was present, create a series of html help files and quit. The location of the html help files will be printed to screen. This html help file can be opened in a web browser and contains all the details that are presented here.

In the following subsection we describe each keyword available for the batch file and how to use it. An example batch file is stored in `CALCHEP/utile/batch_file`.

7.3.1 Structure and keywords of the batch file

Comments

Any line beginning with a `#` is ignored by `run_batch`. The `#` has to be at the very beginning of the line. Some examples are:

```
# This is ignored.  
#Model: Standard Model This is ignored.  
Model: # Standard Model(CKM=1) This is not ignored.
```

Model

The first section of the batch file should contain the specification of the model. This is done by model name and should match exactly the name in the CalcHEP model list. So, if you want to run the "Standard Model(CKM=1)", you would specify this with the batch file line:

```
Model : Standard Model(CKM=1)
```

There is no default for this line. It must be included.

The gauge of the calculation should also be specified in this section. Choices are Feynman and unitary gauge. CalcHEP is much better suited to calculation in Feynman gauge, but there may be times that unitary gauge is useful. This can be specified using the keyword **Gauge** as in:

```
Gauge : unitary
```

The default is Feynman.

Process

Processes are specified using the **Process** keyword and standard CalcHEP notation as in:

```
Process : p,p->j,l,l
```

Multiple processes can also be specified as in:

```
Process : p,p->E,ne
```

```
Process : p,p->M,nm
```

As many processes as desired can be specified. When more than one process is specified, the processes are numbered by the order in which they are specified in the batch file. So, in this example, **p,p->E,ne** is process 1 and **p,p->M,nm** is process 2. This numbering can be useful when specifying QCD scale, cuts, kinematics, regularization and distributions allowing these to be specified separately for each process. There is no default for this keyword. It must be specified.

Decays are specified using the **Decay** keyword and are also in standard CalcHEP notation as in:

```
Decay : W->l,nu
```

Again, multiple decays can be specified as in:

```
Decay : W->l,nu
```

```
Decay : Z->l,l
```

The default is to not have any decays. Cuts, kinematics, regularization and distributions do not apply to decays.

It is sometimes convenient to specify groups of particles as in the particles that compose the proton or all the leptons. This can be done with the keyword **Alias** as in:

```
Alias : p=u,d,U,D,G
```

```
Alias : l=e,E,m,M
```

```
Alias : nu=ne,Ne,nm,Nm
```

```
Alias : W=W+,W-
```

As many alias particles as necessary can be specified. These definitions can be used in cuts and distributions as well as in the processes and decays. The default is not to have any alias definitions.

It is sometimes necessary to remove particles from internal lines of the diagrams. This can be done with the keyword "Remove" as in:

Remove : $W \rightarrow 2, Z$

The notation is the same as in the graphical interface and is used in the symbolic session of productions processes directly without any modifications. These remove statements are not used in the decays. (All decay diagrams are included independent of any remove statements.) Aliases can not be used in Remove statements. We would like to stress that the user should use this keyword with great care as violation of gauge invariance could occur ruining important cancellations between diagrams.

PDF

The PDF of a proton or antiproton can be specified with the `pdf1` and `pdf2` keywords which correspond to the pdfs of the first and second incoming particles respectively. Choices for these keywords are:

- `cteq6l` (anti-proton)
- `cteq6l` (proton)
- `mrst2002lo` (anti-proton)
- `mrst2002lo` (proton)
- `cteq6m` (anti-proton)
- `cteq6m` (proton)
- `cteq5m` (anti-proton)
- `cteq5m` (proton)
- `mrst2002nlo` (anti-proton)
- `mrst2002nlo` (proton)
- `None`

An example for the LHC is:

```
pdf1 : cteq6l (proton)
pdf2 : cteq6l (proton)
```

The default is `None`. These keywords can also be used for electron positron colliders. For this process the available pdfs are:

- ISR
- ISR & Beamstrahlung
- Equiv. Photon
- Laser photons
- None

The following proton electron collider pdf is also available:

- Proton Photon

All of these pdfs must be typed exactly or copied into the batch file.

If ISR & Beam is chosen, then the following beam parameters may be specified:

Bunch x+y sizes (nm) : 550
 Bunch length (mm) : 0.45
 Number of particles : 2.1E+10

The default values are the default values in CalcHEP and correspond roughly with the ILC.

If Equiv. Photon is chosen for the pdf, then the following parameters may be specified:

Photon particle : e^-
 $|Q|_{\max}$: 150

Choices for the Photon particle keyphrase are μ^- , e^- , e^+ , μ^+ . The default is e^+ . The default for the keyword $|Q|_{\max}$ is the same as in the CalcHEP interactive session.

If Proton Photon is chosen then the following may be specified:

Incoming particle mass : 0.937
 Incoming particle charge : -1
 $|Q^2|_{\max}$: 2.1
 Pt cut of outgoing proton : 0.11

The defaults are the same as in the CalcHEP interactive session.

Momenta

The momentum of the incoming states can be specified with the keywords `p1` and `p2` and are in GeV as in:

```
p1 : 7000
p2 : 7000
```

These are the default values for the momenta.

Parameters

The default parameters of the model are taken from the `varsN.mdl` file in the `models` directory. Other parameter values can be used if specified using the `Parameter` keyword. Here is an example:

```
Parameter : EE=0.31
Parameter : SW=0.481
Parameter : MZ=91.1884
Parameter : wW=2.08895
```

This gives a convenient way of changing the default values of the parameters. Simply open CalcHEP in symbolic mode, choose to edit the model and change the values of the independent parameters. These new values will then become the default values used by this batch program. There is no need to redo the process library.

Scans

In some models it is useful to scan over a parameter such as the mass of one of the new particles. For example, if there is a new W' gauge boson, it may be desirable to generate events and/or distributions for a range of masses for the W' . This can be done with the `Run parameter`, `Run begin`, `Run step size` and `Run n steps` keyphrases. Here is an example:

```
Run parameter : MWP
Run begin : 400
Run step size : 50
Run n steps : 17
```

This will generate the events and/or distributions for the model with the mass of the W' set to 400GeV, 450GeV, 500GeV,...1200GeV. As many runs as desired can be specified (including zero). For each run, all four keyphrases have to be specified. Furthermore, if there is more than one run, all four keyphrases have to be specified together. Here is an example with two runs:

```

Run parameter : MWP
Run begin : 400
Run step size : 50
Run n steps : 17
Run parameter : MF
Run begin : 2000
Run step size : 200
Run n steps : 11

```

This example will run over both parameters MWP and MF.

QCD

The parameters of the QCD menu of the numerical session can be specified as in the following example:

```

parton dist. alpha : ON
alpha(MZ) : 0.118
alpha nf : 5
alpha order : NLO
mb(mb) : 4
Mtop(pole) : 174
alpha Q : M45

```

The default values are the ones in the interactive session. Not all the keywords have to be included in the batch file. It is sufficient to include the ones that need to be changed. For example, if only the QCD scale needs to be changed, it can be specified as:

```
alpha Q : Mt/2
```

The QCD scale can be specified in terms of the invariant mass of certain final state particles as in M_{ij} which means that the QCD scale is taken to be the invariant mass of particles i and j . Or, it can be specified as a formula in terms of the parameters of the model as in $M_t/2$ which means half of the top quark mass. When specifying the scale in terms of the invariant mass of final state particles, the numbers are taken from the way the processes are entered with the **Process** keyword. So, if the process is specified as $p, p \rightarrow j, l, n$, M45 means the invariant mass of the lepton and neutrino (l, n). The batch script will take care of renumbering if the subprocesses have the final state particles in a different order. It is also sometimes useful to use a different scale for different processes. For example, suppose the two processes $p, p \rightarrow j, l, n$ and $p, p \rightarrow j, j, l, n$ are specified in the batch file, the scales could be specified as in this example:

```
alpha Q :1: M45
alpha Q :2: M56
```

The number between the `::` specifies which process to apply this scale and corresponds to the order in which the user specified the processes. If more than one process is specified, but the same non default scale is desired for all of them, this can be specified as in:

```
alpha Q : Mt/2
```

This specification will apply the same scale `Mt/2` to all processes.

Cuts

Cuts are specified with the keywords `Cut parameter`, `Cut invert`, `Cut min` and `Cut max` and use standard CalcHEP notation, except for `Cut invert` which can be either `True` or `False`. These cuts are only applied to the production processes. They are not applied to the products of the decays. Here is an example:

```
Cut parameter : T(1e)
Cut invert : False
Cut min : 20
Cut max :
```

For each cut, all four keyphrases have to be present. As many cuts as desired can be included. Including `Cut min` or `Cut max` but leaving the value blank will leave the value blank in the CalcHEP table. If the cut should only be applied to a certain process, then the colon can be changed to `:n:` where `n` is the process number. So, for example, we could do:

```
Cut parameter : T(1)
Cut invert : True
Cut min :
Cut max : 20
Cut parameter : T(j)
Cut invert : False
Cut min : 20
Cut max :
Cut parameter :2: J(j,j)
Cut invert : False
Cut min :2: 0.4
Cut max :2:
```

This set of cuts will apply a p_T cut to leptons and jets in all processes but a jet cone angle cut only to process 2. The numbering of the processes corresponds to the order in which the processes are entered in the batch file. Alias particle names can be used as long as they are defined by the keyword **Alias** in the process section. Note that both of the transverse mass cuts apply a $p_T > 20\text{GeV}$ cut in this example.

Kinematics

As the number of final state particles increases, it can be very helpful to specify the **kinematics** which helps CalcHEP in the numerical integration stage. This is done in exactly the same notation as in CalcHEP. The numbering corresponds to the order the particles are entered in the process in the batch file. Here is an example:

```
Kinematics : 12 -> 34 , 56
Kinematics : 34 -> 3 , 4
Kinematics : 56 -> 5 , 6
```

If multiple processes are specified, using a single colon as in the previous example will apply the kinematics to all processes. If different kinematics are desired for each process, then the **:n:** notation can be used as in:

```
Kinematics :1: 12 -> 34 , 56
Kinematics :1: 34 -> 3 , 4
Kinematics :1: 56 -> 5 , 6
Kinematics :2: 12 -> 3 , 456
Kinematics :2: 456 -> 45 , 6
Kinematics :2: 45 -> 4 , 5
```

where **n** corresponds with the process number as entered in the batch file.

Regularization

When a narrow resonance is present in the signal, it is a good idea to specify the **Regularization**. This is done with the same notation as in CalcHEP. Here is an example:

```
Regularization momentum : 34
Regularization mass : MW
Regularization width : wW
Regularization power : 2
```


Regularization for as many resonances can be specified as desired. Furthermore, different resonances can be specified for each process using the `:n:` notation as in:

```
Regularization momentum :1: 34
Regularization mass :1: MW
Regularization width :1: wW
Regularization power :1: 2
Regularization momentum :2: 45
Regularization mass :2: MZ
Regularization width :2: wZ
Regularization power :2: 2
```

Distributions

Distributions are only applied to the production process. The decays are ignored. Standard CalcHEP notation is used for the distribution parameter. Here is an example:

```
Dist parameter : M(e,E)
Dist min : 0
Dist max : 200
Dist n bins : 100
Dist title : p,p->l,l
Dist x-title : M(l,l) (GeV)
```

The value for the keyphrase `Dist n bins` has to be one of 300, 150, 100, 75, 60, 50, 30, 25, 20, 15, 12, 10, 6, 5, 4, 3 or 2. These are the values allowed by the CalcHEP histogram routines. The values given for the titles have to be pure text. No special characters are currently allowed. Gnuplot must be installed for plots to be produced on the fly and included in the html progress reports. More than one distribution can be specified, however each distribution must be unambiguous and apply in exactly one way for each subprocess. Also, distributions will work even if no events are requested.

For this to work, the distributions have to be unambiguous and apply to all subprocesses the same way. For example, if a process is `p,p->l,l,l` and the distribution `M(l,l)` is given, then this routine will not know which two leptons to apply the distribution to and the results are unpredictable. If the process is `p,p->l,l,l` where `l=e,E,m,M` and the distribution `M(e,E)` is desired, this distribution will only apply to some of the subprocesses and give unpredictable results. Make sure your distribution is unambiguous and applies in exactly one way to each subprocess. If this is done, it should work.

Nevertheless, check each distribution carefully to make sure it is being done correctly.

Events

The number of events is specified with the keyphrase **Number of events**. This specifies the number of events to produce after all subprocesses are combined and decayed. If a run over a parameter is specified, this keyphrase determines the number of events to produce for each value of the run parameter. The number of events requested can be zero. In this case, the cross sections are determined and the distributions generated but no events are produced. Here is an example:

```
Number of events : 1000
```

The name of the file can be specified using the **Filename** keyword. If specified, all the files will begin with this name. Here is an example:

```
Filename : pp-11
```

If **nt_maker** has been installed in the bin directory, PAW ntuples can be made on the fly by setting **NTuple** to **True** as in:

```
NTuple : True
```

The default is **False**.

The keyword **Cleanup** determines whether the intermediate files of the calculation are removed. This can be useful if many large intermediate files are created and space is an issue. On the other hand, it can be useful to keep the files when debugging is necessary. If this keyword is set to **True**, the intermediate files are removed. If set to **False** then they are not removed. Here is an example:

```
Cleanup : False
```

Parallelization

The parallelization mode is set using the keyphrase **Parallelization method** and can be either **local**, **pbs** or **lsf**. In **local** mode, the jobs run on the local computer, in **pbs** mode, the jobs are run on a pbs cluster and in **lsf** mode, the jobs are run on an lsf cluster. If run from a pbs or lsf cluster, the terminal should be on the computer with the pbs or lsf queue. Here is an example of setting the batch to run in pbs mode:

Parallelization mode : pbs

Local mode is the default.

If run in **pbs** mode, there are several options that may be necessary for the pbs cluster. All of them can be left blank in which case they will not be given to the pbs cluster. Here is an example of the options available:

```
Que : brody
Walltime : 1.5
Memory : 1
email : name@address
```

The **que** keyword specifies which pbs queue to submit the jobs to. **Walltime** specifies the maximum time (in hours) the job can run for. If this time is exceeded, the jobs are killed by the pbs cluster. **Memory** specifies the maximum amount of memory (in G) that the jobs can use. If this memory is exceeded by a job, the pbs cluster will kill the job. **email** specifies which email to send a message to if the job terminates prematurely. The default for all of these is whatever is the default on the pbs cluster.

If run in **lsf** mode, there are several options that may be necessary for the lsf cluster. All of them can be left blank in which case they will not be given to the lsf cluster. Here is an example of the options available:

```
Que : brody
Walltime : 1.5
Memory : 1
email : name@address
Project : project_name
```

The **que** keyword specifies which lsf queue to submit the jobs to. **Walltime** specifies the maximum time (in hours) the job can run for. If this time is exceeded, the jobs are killed by the lsf cluster. **Memory** specifies the maximum amount of memory (in G) that the jobs can use. If this memory is exceeded by a job, the lsf cluster will kill the job. **email** specifies which email to send any messages to. The default for all of these is whatever is the default on the lsf cluster.

Sleep time specifies the amount of time (in seconds) the batch script waits before checking which jobs are done and updating the html progress reports. If a very short test run is being done, then this should be low (say a few seconds). However, if the job is very large and will take several hours or days, this should be set very high (say minutes or tens of minutes or hours). This will reduce the amount of cpu time the batch program uses. Here is an example setting the sleep time to 1 minute:

`sleep time : 60`

The default is 3 seconds.

When jobs are run on the local computer, the keyword `Nice level` specifies what nice level the jobs should be run at. If other users are using the same computer, this allows the job to be put into the background and run at lower priority so as not to disturb the other users. This should be between 0 and 19 where 19 is the lowest priority and the nicest. Typically, it should be run at level 19 unless the user is sure it will not disturb anyone. The nice level should be set both for a local computer and for a pbs or lsf batch run. The reason is that some jobs are run on the pbs or lsf queue computer even on the pbs or lsf cluster. Here is an example:

`Nice level : 19`

Level 19 is the default.

Vegas

The number of vegas calls can be controlled with the keywords `nSess_1`, `nCalls_1`, `nSess_2` and `nCalls_2`. The values are the same as in CalcHEP. Here is an example:

```
nSess_1 : 5
nCalls_1 : 100000
nSess_2 : 5
nCalls_2 : 100000
```

The defaults are the same as in CalcHEP.

Generator

The following parameters of the event generation can be modified:

```
sub-cubes : 1000
random search : 100
simplex search : 50
MAX*N : 2
find new MAX : 100
```

The defaults are the CalcHEP defaults.

7.3.2 Example of the batch file

This example generates 1000 events (for each Mh) of the process $p,p \rightarrow W,b,B$ for the model Standard Model(CKM=1).

```
#####
# Model Info
#####
Model:          Standard Model(CKM=1)
Model changed: False
Gauge:          Feynman
#####
# Process Info
#####
Process:   p,p->W,b,B
Decay:     W->le,n

Alias: p=u,U,d,D,s,S,c,C,b,B,G
Alias: W=W+,W-
Alias: le=e,E,m,M
Alias: n=ne,Ne,nm,Nm
Alias: jet=u,U,d,D,s,S,c,C,b,B,G
#####
# PDF Info
#####
pdf1 : cteq6l (proton)
pdf2 : cteq6l (proton)

#####
# Momentum Info
#####
p1 : 4000
p2 : 4000

#####
# Parameter Info
#####
Parameter : Mtp=172.5
#####
# Run Info
#####
Run parameter: Mh
```

Run begin: 120
Run step size: 5
Run n steps: 3

```
#####  
# QCD Running Info  
#####  
alpha Q : M45
```

```
#####  
# Cut Info  
#####  
Cut parameter: M(b,B)  
Cut invert: False  
Cut min: 100  
Cut max:
```

```
Cut parameter: J(jet,jet)  
Cut invert: False  
Cut min: 0.5  
Cut max:
```

```
Cut parameter: T(jet)  
Cut invert: False  
Cut min: 20  
Cut max:
```

```
Cut parameter: N(jet)  
Cut invert: False  
Cut min: -2.5  
Cut max: 2.5
```

```
#####  
# Kinematics Info  
#####  
Kinematics : 12 -> 3, 45  
Kinematics : 45 -> 4 , 5
```

```
#####  
# Regularization Info  
#####
```

```

Regularization momentum: 45
Regularization mass:      Mh
Regularization width:     wh
Regularization power:     2

```

```

#####
# Distribution Info
#####
Dist parameter:    M(W+,b)
Dist min:  100
Dist max:  200
Dist n bins: 100
Dist title:  p,p->W,b,B
Dist x-title: M(W+,b) (GeV)

```

```

#####
# Event Info
#####
Number of events: 1000
Filename : pp-Wbb-lnbb

```

```

#####
# Parallelization Info
#####
Parallelization method : local
Max number of cpus : 2
sleep time : 3
#####
# Vegas Info
#####
nSess_1 : 5
nCalls_1 : 100000
nSess_2 : 5
nCalls_2 : 100000

```

7.3.3 Monitoring of the calchep batch session

After the start of calchep batch session with
`./calchep_batch batch_file`
command, the following information appears on the screen:

Processing batch:

Progress information can be found in the html directory.

Simply open the following link in your browser:

file:///WORK/html/index.html

You can also view textual progress reports in WORK/html/index.txt and the other .txt files in the html directory.

where WORK denote the path to calcchep working directory. Using browser, user can monitor the progress of all stages of the calcchep batch session and check CalcHEP batch details (Fig.12(a)), details of symbolic session (Fig.12(b)), the progress in numerical session (Fig.12(c)) as well as the progress on event generation (Fig.12(d)).

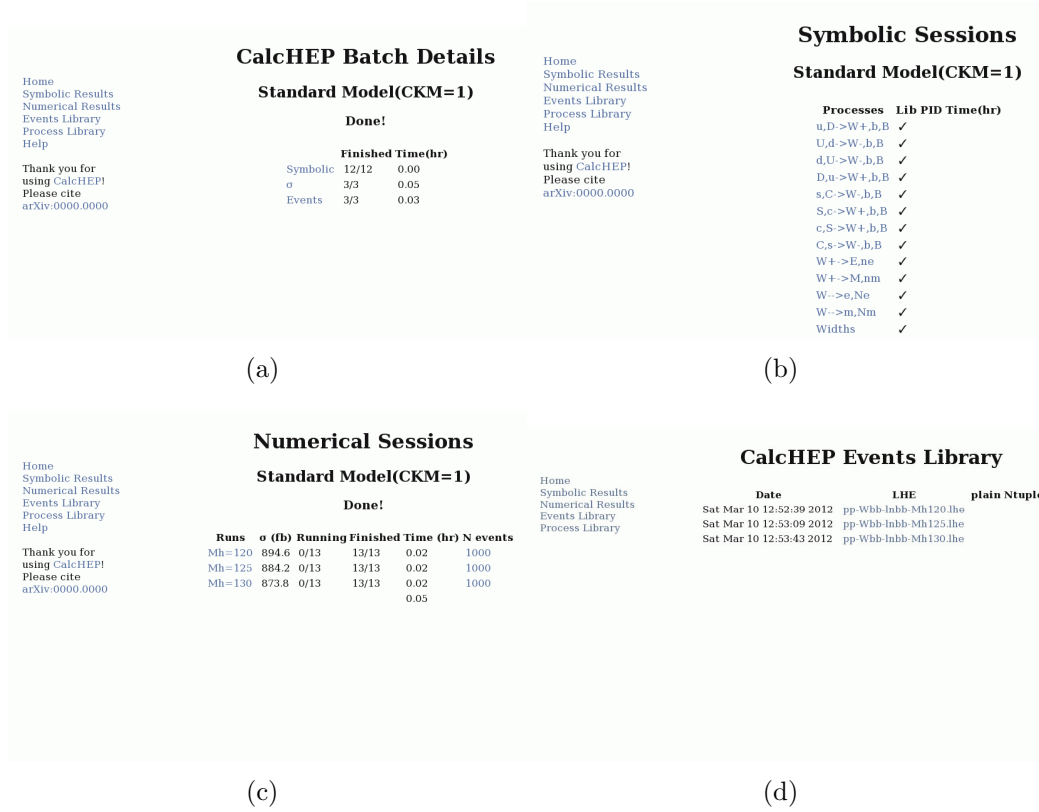


Figure 12: Monitoring of the CalcHEP batch session using through the web browser:

Further details on numerical session can be checked by clicking on particular value of the running parameter (M_h in our example), which opens

the window with detailed information shown in Fig.13. This page also present the requested distributions. Moreover, the results shown in html browser as also recorded in the ascii files located in the `WORK/html` directory. For example the results for numerical session are recorded in the file `WORK/html/numerical.txt` as well as in the files in the `WORK/html/runs` directory containing further details. For example after sucessful run of the `batch_file` given in the example above, the user should get the `WORK/html/numerical.txt` with the following info:

CalcHEP Numerical Details

Done!

Runs	sigma (fb)	Running	Finished	Time (hr)	N events
Mh120	8.9460e+02	0/13	13/13	0.02	1000
Mh125	8.8420e+02	0/13	13/13	0.02	1000
Mh130	8.7380e+02	0/13	13/13	0.02	1000

7.3.4 Results Storage

After the events and/or distributions are generated, they are stored in the Events directory. The prefix of the files is the name specified in the batch file plus either "-single" if no scans were specified or a string specifying the run parameter values if one or more scans are specified. We will assume this is filename in the following. If events are requested, they will be stored in the files

```
filename.lhe
filename.nt
```

where `filename.lhe` is the event file in Les Houches format and `filename.nt` is in PAW ntuple format. The ntuple file is only created if the keyword `NTuple` is set to `True` and `nt_maker` is present in the bin directory. If distributions are requested, they will be stored in the files

```
filename.distr
filename_1.png
filename_2.png
...
```

[Home](#)
[Symbolic Results](#)
[Numerical Results](#)
[Events Library](#)
[Process Library](#)
[Help](#)

Thank you for using
 CalcHEP!
 Please cite
 arXiv:0000.0000

Numerical Sessions

Standard Model(CKM=1)

Done!

Processes	σ (fb)	PID	Time (hr)	N events	Details
u,D->W+,b,B	1202.5	16747	0.00	383/383	prt_1 session.dat
U,d->W-,b,B	644.99	16750	0.00	220/220	prt_1 session.dat
d,U->W-,b,B	646.43	17063	0.00	220/220	prt_1 session.dat
D,u->W+,b,B	1203.5	17065	0.00	383/383	prt_1 session.dat
s,C->W-,b,B	81.777	17379	0.00	40/40	prt_1 session.dat
S,c->W+,b,B	81.762	17382	0.00	40/40	prt_1 session.dat
c,S->W+,b,B	82.08	17695	0.00	40/40	prt_1 session.dat
C,s->W-,b,B	82.073	17698	0.00	40/40	prt_1 session.dat
Total	4025.1			1366/1366	

Decays	Γ (GeV)	PID	Time (hr)	N events	Details
W+->E,ne	0.22339	18011	0.00	5101/5100	prt_1 session.dat
W+->M,nm	0.22339	18013	0.00	5101/5100	prt_1 session.dat
W-->e,Ne	0.22339	18555	0.00	5101/5100	prt_1 session.dat
W-->m,Nm	0.22339	18564	0.00	5101/5100	prt_1 session.dat

Widths	PID	Time (hr)	Details
Widths	19099	0.00	session.dat

Total 894.6 0.02

Distributions

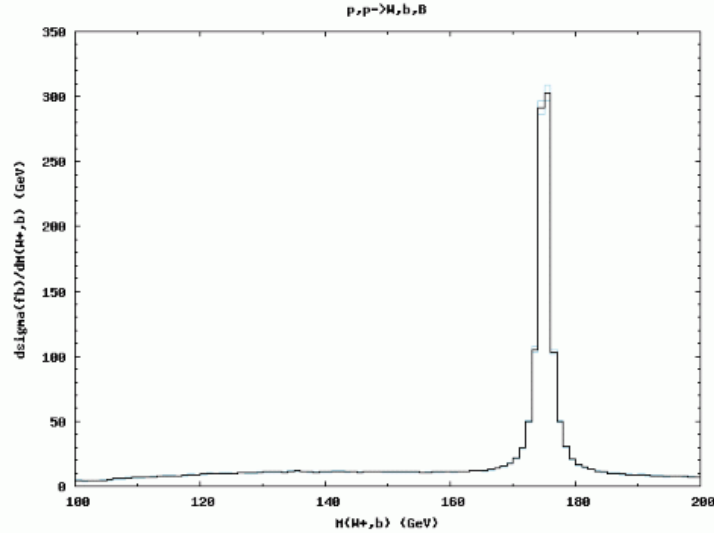


Figure 13: Monitoring of the CalcHEP batch session using through the web browser:

where `filename.distr` is the raw distribution data and can be read by `show_distr` in the `bin` directory. The distributions generated on the fly by the batch script are stored in the files ending in `.png`.

8 Particle Interaction Model Implementation

A model of particle interaction in CalcHEP is stored in five tables, named **Parameters**, **Constraints**, **Particles**, **Vertices** and **Libraries**. These tables are stored in the respective files `varsN.mdl`, `funcN.mdl`, `prtclsN.mdl`, `lgrngN.mdl`, `extlibN.mdl` which are located in the `models/` sub-directory of the users work directory. The `N` in these file names refers to the model number. Each model has a unique `N`. For all of these tables, a `%` at the beginning of any row means that that row is a comment and CalcHEP ignores it. We describe each of these tables in this section.

The SLHAplus [57] library contains external functions that allow CalcHEP to read parameters from a **SLHA** file. We include it in the CalcHEP package. A brief description can be found in Subsection 9.1.

Although it is possible to implement a new model of particle interactions directly using the table definitions described here, for complicated models with a large number of particles, parameters and Feynman rules, it is a good idea to use an external program to generate the model files. We briefly describe two programs that do this job at the end of this section: LanHEP [35] and FeynRules [36].

8.1 Independent Parameters

The table *Parameters* contains all the independent parameters of the model. It consists of the following three columns:

1. **Name** : This is where the name of the parameter belongs. It can contain up to 11 characters. The first character must be a letter. The others may be either letters or digits. The underscore symbol is also permitted and CalcHEP is sensitive to the case of the characters. There is a set of reserved names which cannot be used for parameter names:
 - `i` is reserved for the imaginary unit;
 - `Sqrt2` is reserved for $\sqrt{2}$;
 - `p1`, `p2`, `p3`, ... are reserved for particle momenta;
 - `m1`, ..., `M1`, ... are reserved for Lorentz indices;
 - `G5` is used for the γ^5 Dirac matrix;

There is another subtlety that should be considered when naming parameters. Although CalcHEP is sensitive to the case of the parameters, Reduce is not. Therefore, if the user would like to use the CalcHEP results in Reduce, he/she should distinguish all names by more than case.

Additionally, although CalcHEP allows underscores as part of parameter names, the underscore is treated differently by Mathematica. So, if the user would like to use the CalcHEP results in Mathematica, he/she should not use underscores in the parameter names. Furthermore, CalcHEP allows parentheses in parameter names but Reduce and Mathematica do not. The user should name their parameters accordingly.

2. **Value** : This is where the numerical value for the parameter is stored. Dimensionful parameters should be in powers of GeV.
3. **Comment** : This is where the user can enter a description of the parameter. It is ignored by CalcHEP and is purely for informational purposes.

8.2 Dependent Parameters

The table *Constraints* contains all the dependent parameters of the model. It consists of two columns:

1. **Name** : This is where the name of the parameter belongs. The restrictions on the names are the same as for the independent parameter names.
2. **Expression** : This is where the formula belongs which defines the value of this dependent parameter. The formula can contain the following:
 - integer and float point numbers,
 - independent parameter names contained in the **Parameters** table,
 - dependent parameter names defined above the current row,
 - parentheses () and arithmetic operators +, -, /, *, ^,
 - the symbols i and $Sqrt2$,
 - standard functions from the C mathematics library such as \sqrt{x} and $\sin(x)$. The full list of these functions is contained in the `$CALCHEP/include/extern.h` file,
 - functions from the SLHAplus package,
 - the function $if(x, y, z)$ which returns y if $x > 0$ and z otherwise,

- any user defined functions. The code containing these functions should be included in the **Libraries** table. Their prototypes can also be included in the **Libraries** table. If their prototypes are not included, CalcHEP assumes they return double type. A list of the resulting auto-prototyped functions appears in the **results/autopot.h** file after compilation of the numerical code.

Additionally, anything after the % symbol is considered a comment and ignored. This can be used to enter a comment about the dependent parameters. The user can follow the formula with a % and then a comment describing the parameter.

Public and local dependent parameters. Some models can contain thousands of dependent parameters. For a particular process, only a small subset of these is used. For this reason, CalcHEP attempts to reduce the file size by only including the dependent parameters that are used in the numerical code that it generates. The way it does this is that it divides the dependent parameters into two groups which we will call the *public* parameters and the *local* parameters. The *public* parameters are those parameters that are required to calculate all the particle masses and widths, all parameters that depend on external functions (except the standard *C* math functions) and all dependent parameters above any of these. In other words, all the parameters from the top of the **Constraints** table down to the last parameter required for the calculation are *public* and are included in the numerical code. All dependent parameters below this are defined as *local* and are not included in the numerical code.

If the user would like to force CalcHEP to include a larger subset of the dependent parameters in the numerical code, he/she can place the comment **%Local!** in the **Constraints** table in the first column. CalcHEP will always include the parameters up to, at least, this point. (All the parameters above the **%Local!** line will always be considered *public*.) An example of a **Constraints** table with this comment is:

Name	Expression
...	...
%Local!	
...	...

where the ... refers to other entries in the **Constraints** table.

All the *public* constraints are compiled and calculated together and separately from the squared amplitude code. Thus, passing of parameters via

global variables between functions involved in the calculation of the dependent parameters in user defined code is possible. The *public* parameters appear in the menus of the interactive sessions and can be used in the definition of the QCD scale and in the limits of the cuts and histograms.

The *local* constraints, on the other hand, are only calculated when needed by the squared amplitude. The code for these parameters is attached to the squared amplitude code.

8.3 Particles

The particles are defined in the **Particles** table which consists of 11 columns. Each particle anti-particle pair is described by one row of the table. The columns are:

1. **Full name:** The full name of the particle can be entered here. It is not used directly by CalcHEP . It is used to clarify what the short particle names mean.
- 2&3. **A and Ac** : These columns are where the particle name and antiparticle name belong. More precisely, these columns contain the quantum field and its C-conjugate. The field operator acting on the vacuum is understood to create the corresponding anti-particle. Self-conjugate fields (such as photons and Majorana neutrinos) should contain the same name in both columns. Any printing character can be used in the particle name except white space, parentheses and the percent symbol (%). The length of the particle name can not exceed 8 symbols. For long particle names, we should note that the graphical representation of the diagrams might contain overlapping symbols.
4. **PDG** : This is where the PDG code [58] belongs. This number is used mainly for interfacing with other packages. For example, these codes are included in event files [41] in order to communicate the particle flavor to other programs. The parton distribution functions are also applied according to this number. The conventional PDG codes should be used for SM particles. For other particles, the user should ensure that the code is not reserved for another particles such as a meson or baryon. Otherwise, conflicts could arise when passing events to other programs, such as Pythia.
5. **2*Spin** : This is where the spin of the particle is specified. It should be entered as the integer equal to twice the spin. In other words, 0 should be entered for a scalar field, 1 for a spin 1/2 fermion, 2 for a

vector boson, 3 for a spin 3/2 fermion and 4 for a spin-2 boson. Spin 3/2 and 2 particles should be massive.

6. **Mass** : This is where the mass of the particle is entered. If massless, 0 can be entered. Otherwise, it must be a parameter name which is defined in either the **Parameters** table or the **Constraints** table.
7. **Width** : This is where the width of the particle is entered. If the particle is stable, 0 can be entered. Otherwise, it must be a parameter name. In this case, however, this parameter can be defined in the **Parameters** table, the **Constraints** table or it can be preceded with the ! symbol. If it is preceded with the ! symbol, CalcHEP will automatically calculate it when needed. In this case, the parameter should not appear in either the **Parameters** table or the **Constraints** table. When automatically calculating the width, CalcHEP first attempts the $1 \rightarrow 2$ decays. If none are found, it attempts the $1 \rightarrow 3$ decays. If none are still found, it attempts the $1 \rightarrow 4$ decays. If none are found at this point, it takes the width as zero. If information about particle widths was downloaded via SLHA file (See section 9.1), then widths are not evaluated and CalcHEP substitutes downloaded values in particle propagators.
8. **Color** : This is where the color (SU(3)) representation is specified. Supported representations are the singlet (specified by a 1), the fundamental triplet (specified by a 3) and the octet (specified by a 8.) If the particle is specified as a triplet, the antiparticle is treated as an anti-triplet (the $\bar{3}$ representation.)
9. **Aux** : This field is used to modify the propagator of the field. For most fields, this column will be left blank. The other possibilities are:
 - ***** : specifies that the propagator should be point like (all momentum dependence is dropped.) This can be used to construct 4-fermion propagators, for example. These fields can not appear as external states of processes.
 - **l** and **r** : are used to specify that a fermion is purely left and right handed respectively. This can only be applied to massless fermions. The effect of this is that when CalcHEP averages over the spin of the incoming fermion, it takes into account that there is only one polarization for this particle. This is used, for example, for the SM neutrinos.

- `g` : declares that the vector particle is treated as a *gauge* boson. In this case t’Hooft-Feynman gauge is used for the vector boson propagator and the ghost fields `A.c` and `A.C` (where `A` is the name of the vector boson) as well as the Goldstone boson `A.f` can contribute to the Lagrangian. A massless vector particle must be treated as a *gauge* boson. In the absence of `g` in this column, the unitary gauge is always used for massive vector bosons and the ghosts and Goldstones associated with it are not used in Feynman diagrams.

The Formulaes for the particle’s propagators are presented in Section (8.6).

The `Aux` column can also be used to specify a particle’s electric charge. This charge is required by many external packages. CalcHEP already knows the charge of the SM particles and assigns it according to the particle’s PDG code. It can determine the charges of many BSM particles by analyzing the Feynman rules and assuming they conserve electric charge. However, for some particles, this will not be sufficient to determine their charge. For this reason, CalcHEP allows to specify the charge of any BSM particle. Specifically, three times the charge should be entered in the `Aux` field. For example, a particle with electric charge of -1 would be entered as `-3`, a particle with electric charge of $2/3$ would be entered as `2` and so on. This charge must be written before other symbols in this column (if there are any.) We reiterate that this charge is not used to define the Feynman rules of the photon in calculations done by CalcHEP. The interactions of the photon are entered in the `Vertices` table along with all other Feynman rules (see Subsection 8.4.) The electric charge defined in the `Aux` column of the `Particles` table is only used to communicate with other programs that require it.

- 10&11. `LaTeX(A)` and `LaTeX(A+)` : This is where the \LaTeX symbol for the particle and antiparticle are entered. These symbols are used when CalcHEP produces \LaTeX output for the Feynman diagrams that it constructs.

8.4 Interaction Vertices

The `Vertices` table contains the Feynman rules for the model. The first four columns (`A1`, `A2`, `A3` and `A4`) specify the particles and antiparticles involved in the interaction. These must be the particle and antiparticle names defined

in the **Particles** table. The last of these **A4** may be empty, which specifies a three-point vertex. The first three columns must be nonempty. (The propagators are not specified in this table. They are hard coded. Section 8.6 contains further details.)

The last two columns, **Factor** and **LorentzPart** define the vertex. If S is the action for a particular vertex, the vertex can be obtained by functionally differentiating with respect to the fields in the vertex as in

$$\frac{\delta S}{\delta A1_{[m_1]}(p_1) \delta A2_{[m_2]}(p_2) \delta A3_{[m_3]}(p_3) [\delta A4_{[m_4]}(p_4)]} = \quad (7)$$

$$(2\pi)^4 \delta^4(p_1 + p_2 + p_3 [+p_4]) [C^{-1T}] \textit{ColorStructure} \cdot \textit{Factor} \cdot \textit{LorentzPart} ,$$

where p_i and m_i refer to the 4-moment and Lorentz indices (if any.) The square brackets ([and]) denote parts of the expression which only appear in some vertices, but not others. The Fourier transform is defined as

$$A(x) = \int \frac{d^4k}{(2\pi)^4} e^{-ik \cdot x} A(k) . \quad (8)$$

The other pieces of Equation (8.4) will be discussed below.

The **Factor** column is where the *Factor* from Equation (8.4) belongs. This must be a rational monomial constructed from the model parameters, integer number and the imaginary unit (**i**). It is best to factor as much as possible from the *LorentzPart* since the *LorentzPart* of the Feynman diagrams is usually the most time consuming and memory intensive part of the calculation.

The **LorentzPart** column is where the *LorentzPart* from Equation (8.4) belongs. It must be a Lorentz tensor or a Dirac γ -matrix expression. The coefficients of the terms in this expression can be polynomials of the model parameters and scalar products of the momenta. The division operator (/) is forbidden from this column. It must be transferred to the **Factor** column or into a model parameter.

The notation for Lorentz indices, momenta, contractions, and the metric tensor are similar to those in the *Reduce* package. The Lorentz indices of the fields in the vertex are labeled by a **m** for the first index and a **M** for the second index followed by the particle number for that vertex. For example, a vector field in the third column would have Lorentz index **m3** while a tensor field in the second column would have Lorentz indices **m2** and **M2**. The momenta use the symbol **p** followed by the same number. For example, a scalar field in column 1 would have momentum **p1**. A dot (.) is placed between two momenta, a momentum and its Lorentz index, and between two Lorentz indices (for the metric tensor.) Here are some examples:

$$\begin{array}{lll}
\mathbf{p1.p2} & \text{means} & p_{1\mu} p_2^\mu \\
\mathbf{p1.M2} & \text{means} & p_1^{M_2} \\
\mathbf{m1.m2} & \text{means} & g_{m_1 m_2}.
\end{array}$$

Dirac γ -matrices are written with a **G** and the momentum or Lorentz index in parentheses, while the γ_5 matrix has a 5 without parentheses. For example, we have:

$$\begin{array}{lll}
\mathbf{G(m1)} & \text{means} & \gamma^{m_1} \\
\mathbf{G(p2)} & \text{means} & \gamma^\mu p_{2\mu} \\
\mathbf{G5} & \text{means} & \gamma_5
\end{array}$$

The γ_5 matrix is defined by

$$\gamma_5 = i \gamma_0 \gamma_1 \gamma_2 \gamma_3 .$$

The anti-commutation relation for the gamma matrices in CalcHEP notation is

$$\mathbf{G(v1) G(v2) + G(v2) G(v1) = 2 v1.v2 ,}$$

where **v1** and **v2** are either momenta or Lorentz indices.

In the case of anti-commuting fields the functional derivative in Equation (8.4) is assumed to act from the right. The number of fermion fields in a vertex must be either two or zero. If the user would like to implement a four-fermion interaction, he/she must use an unphysical auxiliary field with a point-like propagator (see Subsections 8.3 and 8.6 for further details.)

CalcHEP interprets the anti-particle spinor field as a C-conjugated particle field, rather than the Dirac conjugated field. These definitions are related to each other by

$$\frac{\delta}{\delta \psi^c} = C^{-1T} \frac{\delta}{\delta \psi} \quad (9)$$

which is the reason for the appearance of the C^{-1T} matrix in Eq. (8.4). The particle and anti-particle fields can appear in the vertices in any order. Vertices can also contain two particle fields or two antiparticle fields. In other words, vertices that violate fermion number are allowed.

Any fermion vertex can be written in two forms which depend on the order of the fermion fields. After permutation of the fermion fields, the **LorentzPart** is transformed according to

$$G(v_1) G(v_2) \dots [G5] \dots G(v_n) \rightarrow (-G(v_n)) \dots [G5] \dots (-G(v_2)) (-G(v_1)) , \quad (10)$$

where the order of the gamma matrices is reversed and each gamma matrix with a Lorentz index gets a sign change while the γ_5 matrix does not get a sign change.

We note that the definition in Eq. (8.4), the *LorentzPart* has the appropriate symmetry property when identical particles appear in the vertex. This symmetry is not checked by CalcHEP, and its absence will lead to the wrong results. Equation 10 can be used to check this symmetry in the case of two identical Majorana fields in one vertex. It should also be noted that in the case of n identical particles, the functional derivative (8.4) gets a corresponding factor of $n!$ which should be included in the vertex.

The totally antisymmetric Levi-Civita tensor can be used in vertices. It is given by `eps(v1,v2,v3,v4)`, where `v1`, `v2`, `v3`, and `v4` are either momenta or Lorentz indices.

The *ColorStructure* from Eq. (8.4) is not included in the **Vertices** table. CalcHEP substitutes it in automatically according to the following rules: If all the particles in the vertex are color singlets, CalcHEP inserts 1. If the vertex contains one fundamental and one antifundamental ($3 \times \bar{3}$), the identity matrix is inserted. If the vertex contains two color octet fields (8×8), the identity matrix is inserted. If the vertex contains three color octet fields ($8 \times 8 \times 8$), it inserts

$$-if(a1,a2,a3)$$

where $f_{a2,a3}^{a1}$ is the structure constant of SU(3) and the color adjoint indices $a1$, $a2$, and $a3$ are taken in the same order they appear in the **Vertices** table. If the vertex contains a fundamental, an antifundamental, and a color adjoint field ($3 \times \bar{3} \times 8$), CalcHEP inserts

$$\frac{1}{2}\lambda(\bar{i},j,a),$$

where $\lambda(\bar{i},j,a)$ are the Gell-Mann matrices. Other color structures are not implemented in CalcHEP, however, it is possible to construct them by means of an unphysical auxiliary field (see Subsections 8.3, 8.6 and 13 for further details.)

8.5 External functions and libraries.

The **Libraries** table is used to link external code and declare external functions. Lines beginning with a `%` are comments and are ignored by CalcHEP. Lines beginning with the keyword `extern` are considered to be prototypes of external functions defined in external code. These lines should include the

full function prototype (including the semicolon at the end) on one line in the syntax of the *C* programming language. These functions can be used in the definitions of the dependent parameters in the **Constraints** table (see Subsection 8.2.)

External code and libraries can be linked to the numerical code by using this table as well. The user should enter a list of the external code, libraries and any flags necessary for his/her model in this table. Some typical examples of external code are user defined kinematical variables which can be used in cuts and histograms (see Subsection 5.7) and the LHAPDF libraries (see Subsection 5.3.) All lines which do not start with % or **extern** are concatenated and passed to the linker which creates the executable for numerical calculations. These lines can make use of environment variables. CalcHEP defines two in its startup scripts (**calchep** and **calchep_batch**) that the user can make use of. They are **\$CALCHEP** which is the path to the CalcHEP root directory and **\$WORK** which is the path to the user's working directory. The user can also make use of his/her own environment variables. These environment variables can be used with or without parentheses (either **\$CALCHEP** or **\$(CALCHEP)** is acceptable.) CalcHEP will translate between the two depending on whether they are used in a **Makefile** or in a shell environment.

For functions presented in the SLHAplus package (section (9.1) prototyping and special link instructions are not needed.

8.6 Propagators

CalcHEP defines the propagators for particles of spin less than or equal to two. These propagators are hard coded and not modifiable in by the user unless specified below.

Spin 0: The spin-0 propagator is given by

$$\langle 0|T[A(p_1), A^+(p_2)]|0 \rangle = \Delta_c(p_1, p_2, M) = \frac{i\delta(p_1 + p_2)}{(2\pi)^4(p_1^2 - M^2)} .$$

Spin 1/2: The spin-1/2 propagator is given by

$$\langle 0|T[A(p_1), \bar{A}(p_2)]|0 \rangle = (\not{p}_1 + M) \Delta_c(p_1, p_2, M) ,$$

where $\not{p} = p^\mu \gamma_\mu$. If the fermion is defined to be purely left or right handed (see Subsection 8.3), the propagator is defined as

$$\frac{\not{p}_1(1 \pm \gamma_5)}{2} \Delta_c(p_1, p_2, M) .$$

Spin 1: In unitary gauge, the propagator is given by

$$\langle 0|T[A^{m_1}(p_1), (A^{m_2})^+(p_2)]|0 \rangle = -(g^{m_1 m_2} + \frac{p_1^{m_1} p_2^{m_2}}{M^2}) \Delta_c(p_1, p_2, M), \quad (11)$$

while in t'Hooft-Feynman gauge, it is given by

$$-g^{m_1 m_2} \Delta_c(p_1, p_2, M) .$$

We remind the user that a massless vector particle must be defined as a gauge boson (see Subsection 8.3.)

Spin 3/2: The spin-3/2 propagator is given by

$$\begin{aligned} \langle 0|T[A^{m_1}(p), \bar{A}^{m_2}(p')]|0 \rangle = & \left(-3(\not{p} + M)(g^{m_1 m_2} - \frac{p^{m_1} p^{m_2}}{M^2}) \right. \\ & \left. - (\gamma^{m_1} + \frac{p^{m_1}}{M})(\not{p} - M)(\gamma^{m_2} + \frac{p^{m_2}}{M}) \right) \Delta_c(p, p', M) \end{aligned} \quad (12)$$

Spin 2: The spin-2 propagator is given by

$$\begin{aligned} \langle 0|T[A^{m_1 \mu_1}(p), (A^{m_2 \mu_2})^+(p')]|0 \rangle = & \left((g^{m_1 m_2} + 2\frac{p^{m_1} p^{m_2}}{M^2})(g^{\mu_1 \mu_2} + 2\frac{p^{\mu_1} p^{\mu_2}}{M^2}) \right. \\ & - 3(g^{m_1 \mu_1} p^{m_2} p^{\mu_2} + g^{m_2 \mu_2} p^{m_1} p^{\mu_1} + g^{m_1 \mu_2} p^{\mu_2} p^{\mu_1} + g^{m_2 \mu_1} p^{m_1} p^{\mu_2}) M^{-2} \\ & \left. + 3(g^{m_1 \mu_1} g^{m_2 \mu_2} + g^{m_1 \mu_2} g^{m_2 \mu_1} - g^{m_1 m_2} g^{\mu_1 \mu_2}) \right) \Delta_c(p, p', M) \end{aligned}$$

Auxiliary propagators: When massive particles are marked as auxiliary fields (see Subsection 8.3) by putting a ***** in the **Aux** column, the momentum dependence of the propagator is removed. $\Delta_c(p_1, p_2, M)$ is replaced with

$$\frac{\delta(p_1 + p_2)}{(2\pi)^4 i M^2}$$

and all terms proportional to the particle momentum p in the numerator are dropped. Auxiliary particles cannot appear as incoming or outgoing states. They are only used to implement point-like interactions.

8.7 Ghost and Goldstone fields propagators

In addition to the fields enumerated in the *Particles* table, the Lagrangian can depend on a few other fields. In particular, gauge theories have Faddeev-Popov ghosts [48] and, if broken, Goldstone bosons. Furthermore, complex color structures require a special tensor auxiliary field. All of these fields are automatically generated by CalcHEP where appropriate by adding a final **.c**, **.C**, **.f**, **.t** or **.T** as described below.

Faddeev-Popov ghosts and anti-ghosts are generated for any gauge vector particle which is marked by a **g** in the '*Aux*' column of the *Particles* table (see Subsection 8.3.) The names of the Faddeev-Popov ghosts and anti-ghosts are constructed by adding a **.c** and **.C**, respectively, to the particle name. For example, if the gluon is named **G**, the gluonic ghost is named **G.c** and the gluonic anti-ghost is named **G.C**. The ghosts and anti-ghosts corresponding with the **W+** and **W-** gauge bosons are **W+.c**, **W+.C**, **W-.c** and **W-.C**. Hermitian conjugation transforms a Faddeev-Popov ghost into a ghost with the same sign whereas it changes the sign of the anti-ghost. For example,

$$\begin{aligned}(G.c)^+ &= G.c \\ (G.C)^+ &= -G.C \\ (W+.c)^+ &= W-.c \\ (W+.C)^+ &= -W-.C\end{aligned}$$

Faddeev-Popov (anti)ghosts are anti-commuting, scalar fields⁶. The nonzero propagators for these fields are:

$$\langle 0|T[A+.c(p_1), A.C(p_2)]|0 \rangle = \langle 0|T[A+.C(p_1), A.c(p_2)]|0 \rangle = \Delta_c(p_1, p_2, M),$$

where $A+$ is the conjugate of A and M is the mass of the parent particle (we are assuming Feynman gauge.)

The reason CalcHEP introduces the Faddeev-Popov ghosts at tree-level is that it sums over the unphysical polarizations of the gauge bosons in the external states as well as the physical polarizations (see Appendix C) in order to reduce precision loss due to large cancellations. The Faddeev-Popov ghosts (and the Goldstone bosons for a broken gauge theory) are required to cancel the unphysical polarizations. (See [48] for further details.)

Goldstone boson are related to broken symmetries. In the case of broken gauge symmetries, they become the longitudinal degrees of freedom of the gauge boson. CalcHEP automatically generates these fields for massive vector bosons by appending a **.f** to the end of the gauge boson name. For example, the **W+** and **W-** gauge bosons have the Goldstone bosons **W+.f** and **W-.f** associated with them. These Goldstone bosons are commuting, scalar fields that satisfy the same conjugation rules as the gauge boson they belong with. For example, $(W+.f)^+ = W-.f$. The nonzero propagators for these fields are:

$$T[A+.f(p_1), A.f(p_2)] = \Delta_c(p_1, p_2, M),$$

where $A+$ is the conjugate of A and M is a mass of the gauge boson (again we consider Feynman gauge.)

⁶The well-known spin-statistics relation is not valid for unphysical fields.

Auxiliary tensor field. Whereas the Faddeev-Popov ghosts and Goldstone bosons are standard elements of modern quantum field theory, this auxiliary tensor field was invented by the original CalcHEP authors in order to construct complicated color vertices such as the four-gluon vertex. These auxiliary fields are automatically generated whenever a particle is defined with a nontrivial $SU(3)$ color representation by adding `.t` and `.T` to the particle name. Two auxiliary tensor fields are generated automatically and are typically used for a constraint and a Lagrange multiplier.

These auxiliary fields are commutative and satisfy the same conjugation rule as the parent particle, while it is Lorentz-transformed like a tensor field. The propagator is point like

$$\langle 0|T[A+.t^{m_1M_1}(p_1), A.t^{m_2M_2}(p_2)]|0\rangle = \frac{1}{(2\pi)^4 i} \delta(p_1 + p_2) g^{m_1m_2} g^{M_1M_2} . \quad (13)$$

Further information about the use of the Faddeev-Popov ghosts, Goldstone bosons and auxiliary tensor fields can be found in Appendix (E).

9 Tools for model implementation.

9.1 The SLHAplus package

The *SLHAplus* [57] package included in CalcHEP allows to facilitate realization of constraints. Initially the package was designed for SLHA interface. In several models of elementary particles we have noticeable loop corrections to particle masses. There are several packages which allow to perform such calculations for MSSM-like models: Isajet, SoftSusy [59], Spheno [60], SuSpect [61], NMSSMTools [62]. There is an agreement to pass input parameters and the calculated particles spectra and mixing matrices via text files in a special format SLHA [37, 38]. Below is an example of a SLHA input file for MSSM spectrum calculation with input parameters at GUT scale.

```
Block MODSEL                                # Select model
  1      1                                # sugra
Block SMINPUTS                                # Standard Model inputs
  5      4.23000000E+00                      # mb(mb) SM MSbar
  6      1.73100000E+02                      # mtop(pole)
Block MINPAR                                  # Input parameters
  1      1.20000000E+02                      # m0
  2      5.00000000E+02                      # m1/2
  3      1.00000000E+01                      # tanb
  4      1      # sign(mu)
  5      -3.50000000E+02                     # A0
```

In order to create such file one can use the following records in Constraints table

```
open |openAppend("suspect2_lha.in")
input1|aPrintf("Block MODSEL          # Select model\n")
input2|aPrintf(" 1 1                  # SUGRA\n")
input3|aPrintf("Block SMINPUTS          Standard Model inputs\n")
input4|aPrintf(" 5      %E          #mb(mb) SM MSbar\n", MbMb)
input5|aPrintf(" 6      %E          #mt(pole)\n", Mtp)
input6|aPrintf("BLOCK MINPAR          # Input parameters\n")
input7|aPrintf(" 1      %E          # m0\n", Mzero, Mhalf)
input8|aPrintf(" 2      %E          # m1/2\n", Mhalf)
input9|aPrintf(" 3      %E          # tb\n", tb)
input10|aPrintf(" 4      %d          # sign(mu)\n", (int)sgn)
input11|aPrintf(" 5      %E          # A0\n", A0)
```

The `aPrintf` command is similar to C-`printf` family functions. Its return value is a number of printed symbols and not useful. Instructions above generate the file "suspect2_lha.in". Next instruction launches SuSpect which reads SLHA input file and writes in its turn SLHA output file "suspect2_lha.out".

```
sys |System("%s/suspect2.exe", ".")
```

Second argument of `System` command can be used to specify path to SuSpect. Generated output file in case of SuSpect is "suspect2_lha.out". It contains information stored in *Blocks*. For instance

```
BLOCK MASS # Mass Spectrum
# PDG code      mass      particle
      25      1.15987932E+02 # h
1000006      7.67852128E+02 # ~t_1
2000006      1.00613369E+03 # ~t_2
1000022      2.05916966E+02 # ~chi_10
1000023      3.89679950E+02 # ~chi_20
BLOCK STOPMIX # Stop Mixing Matrix
 1  1      4.29327465E-01 # cos(theta_t)
 1  2      9.03148896E-01 # sin(theta_t)
 2  1     -9.03148896E-01 # -sin(theta_t)
 2  2      4.29327465E-01 # cos(theta_t)
```

To read this file one can use record in Constraints table

```
rd |slhaRead("suspect2_lha.out",0)
```

The second argument of `slhaRead` specify kind of data to read. In general case second argument is $m_1 + 2m_2 + 4m_4 + 8m_8$ where

```
m1  0/1  overwrite all / keep old data
m2  0/1  ignore mistake / stop in case of mistake in input file
m4  0/1  read DECAY / don't read Decay
m8  0/1  read BLOCK / don't read Blocks
```

To get values of parameters after reading one can use the `slhaVal` function

```
Mst1 | slhaVal(MASS,MZ,1,1000006) % mass of stop1
Mst2 | slhaVal(MASS,MZ,1,2000006) % mass of stop2
Zt11 | slhaVal("STOPmix",MZ,2,1,1) % stop mixing matrix
Zt12 | slhaVal("STOPmix",MZ,2,1,2)
Zt21 | slhaVal("STOPmix",MZ,2,2,1)
Zt22 | slhaVal("STOPmix",MZ,2,2,2)
```

Here the first argument of `slhaVal` is the name of *Block*, the second one is the *scale* parameter (in this example the scale parameter is not specified in BLOCK), the third parameter fixes the number of *key* parameters, the key parameters themselves follow.

An SLHA file also can contain information about particle widths and decays channels. If such information is downloaded (m4=0), CalcHEP uses the downloaded values for particle widths instead of its automatic calculation.

The SLHApplus package in CalcHEP is considered as a collection of tools for realization of *Constraints*. Except of SLHA interface it contains routines for matrix diagonalizing adapted for direct implementation in CalcHEP tables. Using this part of SLHApplus one can diagonalize real symmetric matrix (for multiplet of real scalar fields), complex self-conjugated matrix(for complex boson fields), generic complex matrix (for spinor Dirac fields⁷), real matrix (for spinor fields). For example

```
id|  rDiagonal(d,M11,M12,..M1d,M22,M23...Mdd)
```

diagonalizes a real symmetric matrix of dimension *d*. The $d(d+1)/2$ matrix elements, M_{ij} ($i \leq j$), are given as arguments. The function returns an integer number *id* which serves as an identifier of eigenvalues vector and rotation matrix. Then `MassArray(id, i)` returns the eigenvalues m_i ordered according to their absolute values, and `MixMatrix(id,i,j)` returns the rotation matrix R_{ij} where

$$M_{ij} = \sum_k R_{ki} m_k R_{kj}$$

Another part of SLHApplus package supports implementation of QCD running couplings. It is initiated by the command

```
LamQCD |  initQCD(alphaSMZ, McMc, MbMb,Mtot)
```

Here input parameters are running QCD coupling at MZ, running masses for *b*- and *c*-quarks at their own scales: $Mc(Mc)$, $Mb(Mb)$, and *t*-quark pole mass. Return value is λ_{QCD} . After initiation one can use functions `alphaQCD(Q)` to calculate QCD coupling at any scale, `McRun(Q)`, `MbRun(Q)`, `MtRun(Q)` -to calculate running masses, and `McEff(Q)`, `MbEff(Q)`, `MtEff(Q)` which calculate quark masses for effective Yukawa couplings. These effective couplings lead to automatic account of loop corrections for Higgs width calculation. Functions presented in the next section also belong to SLHApplus package.

⁷For spinor field we can rotate independently left and right components. So, two unitary rotation matrices need for diagonalizing of generic fermion mass matrix.

9.2 Effective Higgs $\gamma\text{-}\gamma$ and $glu\text{-}glu$ interactions.

9.2.1 Construction of effective vertexes.

Higgs interaction with electric or color charged particle leads to loop induced $h\text{-}\gamma\text{-}\gamma$ and $h\text{-}glu\text{-}glu$ effective vertexes which are responsible for very important signal of Higgs decay and Higgs production in $\gamma\gamma$ and hadron colliders. At the lowest perturbative order these vertexes can be constructed by the rules [63, 64]:

$$h\bar{\psi}\psi \rightarrow \frac{\alpha}{8\pi} f_\psi^c q_\psi^2 h F^{\mu\nu} F_{\mu\nu} A_{1/2}(\frac{M_h^2}{4M_\psi^2})/M_\psi \quad (14)$$

$$M_v h \bar{v}_\mu v^\mu \rightarrow \frac{-\alpha}{16\pi} f_v^c q_v^2 h F^{\mu\nu} F_{\mu\nu} A_1(\frac{M_h^2}{4M_v^2})/M_v \quad (15)$$

$$M_s h \bar{s} s \rightarrow \frac{\alpha}{16\pi} f_s^c q_s^2 h F^{\mu\nu} F_{\mu\nu} A_0(\frac{M_h^2}{4M_s^2})/M_s \quad (16)$$

$$h\bar{\psi}i\gamma_5\psi \rightarrow \frac{\alpha}{16\pi} f_\psi^c q_\psi^2 h F^{\mu\nu} \tilde{F}_{\mu\nu} \tilde{A}_{1/2}(\frac{M_h^2}{4M_\psi^2})/M_\psi \quad (17)$$

Here α presents electromagnetic or strong coupling. f^c is a color factor which depends on color structure of virtual particle. In case of fundamental SU(3) representation f^c is 3 for photon vertexes and 1/2 for gluon ones. For adjoint representation f^c is 8 and (-2) correspondingly. q should be loop electric charge for $\gamma\gamma$ and 1 for gluon channel.

Functions $A_{1/2}$, $A_1 A_0$, $\tilde{A}_{1/2}$ are presented in [64]. These functions are realized in SLHApplus package and have names HggF, HggV, HggS, HggA correspondingly. In general these functions return complex value, the imaginary part appears if argument is larger than one.

In CalcHEP notation $\lambda h F_{\mu\nu}(A) F^{\mu\nu}(A)$ is realized as

```
P1 |P2 |P3 |P4 |> Factor <|>dLagrangian/dA(p1)dA(p2)dA(p3)
A |A |h | | -4*lambda |(p1.p2*m1.m2-p1.m2*p2.m1)
```

TP-odd interaction $\lambda h F_{\mu\nu}(A) \tilde{F}^{\mu\nu}(A)$ reads

```
A |A |h | | -4*lambda |eps(p1,m1,p2,m2)
```

CalcHEP assumes that all vertexes are self-conjugated. From other side effective hVV coupling can be complex. We propose to replace complex coupling on it absolute value after summation of all amplitudes. One expects exactly the same result after amplitude squaring.

9.2.2 QCD corrections to $h\gamma\gamma$.

There are important QCD correction for $h\gamma\gamma$ effective vertex if this vertex is originated by color particles loop. This correction can be presented as a overall factor for coupling

$$1 + \frac{\alpha_s}{\pi} C_l \left(\frac{M_h^2}{4M_l^2} \right) \quad (18)$$

where M_l is mass of loop particles. The C_l functions are known for vector, pseudo-vector, and scalar interactions if loop particle has color dimension 3. The formulas are rather cumbersome and have simple analytical forms only in asymptotic. SLHAplus contains $\text{HgamF}(\tau)$, $\text{HgamA}(\tau)$, $\text{HgamS}(\tau)$ complex functions which interpolate tabulated data and present C_l functions for fermion loop with scalar interaction, fermion loop with pseudo-scalar interaction and scalar particle loop correspondingly. The HDECAY package was used to generation these tables. The appropriate QCD scale for α_s is $M_h/2$. It allows effectively take into account large logarithmic corrections of next orders [63].

9.2.3 QCD corrections for $h\text{-}glu\text{-}glu$

The $h \rightarrow GG$ process at NLO level contains radiative corrections which are plagued by infrared divergence which in its turn cancel infrared divergence of loop diagrams caused by virtual gluons attached to external legs. This is a reason why QCD NLO corrections are presented for partial widths and cross sections, but not for effective vertex as in the $h\gamma\gamma$ case. CalcHEP user has to implement LNO factor to vertex in form

$$\sqrt{1 + \frac{\alpha_s}{\pi} F_{nlo}}$$

where F_{nlo} presents NLO contribution for hGG partial width.

QCD correction for hGG interaction induced by heavy quark loop are known in NNLO precision. In case of scalar (14) and pseudo-scalar (17) interaction they are correspondingly [65], [66]

$$F_{nlo}^{\bar{q}q} = \left(\frac{95}{4} - \frac{7n_f}{6} \right) + \frac{\alpha_s}{\pi} \left(370. - 47.n_f + 0.9n_f^2 - \left(\frac{19}{8} + \frac{2n_f}{3} \right) \log \frac{M_f^2}{M_h^2} \right) \quad (19)$$

$$F_{nlo}^{\bar{q}\gamma\gamma q} = \frac{221}{12} + \frac{\alpha_s}{\pi} \left(171.5 - 5 \log \frac{M_f^2}{M_h^2} \right) \quad (20)$$

The last expression is presented for $n_f = 5$. Here for quark masses one has to use the pole values [67] and α_s has to be calculated at M_h scale. The α_s^3 QCD corrections to hgg vertex in case of massive fermion loop $M_f \gg Mh/2$ was calculated in [68] and appears about $\approx 1\%$.

The NLO correction to scalar loop (16) (SUSY *squars*) is known with NLO precision and appears to be 17/6 larger than $F_{nlo}^{\bar{q}q}$ [69]

$$F_{nlo}^{ss} = \left(\frac{319}{12} - \frac{7n_f}{6} \right) \quad (21)$$

Formulas (19, 20, 21) were obtained in the limit $M_{f,s} \ll M_h/2$. But *a posteriori* it appears that they work well even out of this limit. See example of implementation of loop induced Higgs boson vertexes in the SM(CKM=1 with hGG/AA) model.

9.3 LanHEP: automatic generation of models.

The LanHEP [35] package allows automatic generation of CalcHEP model files. It starts from model definition in terms of particle multiplets and performs substitution of physical particle fields in multiplets. LanHEP also checks at the symbolic level the absence of linear terms and at the numerical level the absence of off-diagonal terms in the quadratic part of Lagrangian. Also LanHEP compares diagonal quadratic terms with declared particle masses. LanHEP allows to avoid a large number of mistakes which could appear in the derivation of Feynman rules by hand. The package is disposed at

<http://theory.sinp.msu.ru/~semenov/lanhep.html>

The downloaded `lhnpNNN.tgz` file contains the source code and examples. As an example we present here part of a LanHEP input file for the Inert Doublet Model model describing only new particles and new interactions beyond the Standard Model.

The IDM [70, 71] contains two $SU(2) \times U(1)$ scalar doublets. In the unitary gauge

$$H_1 = \begin{pmatrix} 0 \\ \langle v \rangle + h/\sqrt{2} \end{pmatrix}, \quad H_2 = \begin{pmatrix} \tilde{H}^+ \\ (\tilde{X} + i\tilde{H}_3)/\sqrt{2} \end{pmatrix} \quad (22)$$

where H_1 is the SM Higgs doublet and H_2 is a new *inert* doublet which does not couple to quarks and leptons. Unlike the SM scalar doublet it does not develop a vacuum expectation value. \tilde{H}^+ , \tilde{X} , and \tilde{H}_3 are the new fields of

the model. The IDM Lagrangian contains only even powers of the doublet H_2

$$\begin{aligned} \mathcal{L} = & (SM \text{ terms}) + D^\mu H_2^* D_\mu H_2 - \mu^2 |H_2|^2 \\ & - \lambda_2 H_2^2 - \lambda_3 H_1^2 H_2^2 - \lambda_4 |H_1^* H_2|^2 - \lambda_5 \text{Re}[(H_1^* H_2)^2] \end{aligned} \quad (23)$$

Because of the $H_2 \rightarrow -H_2$ symmetry, the lightest new particle is stable. In the following example we will use the masses of new particles as well as λ_2 and $\lambda_L = (\lambda_3 + \lambda_4 + \lambda_5)/2$ as independent parameters, The couplings μ , λ_3 , λ_4 , λ_5 can be expressed in terms of the independent parameters.

LanHEP source file for IDM should contain description of new free parameters

```
parameter MHX=111,MH3=222,MHC=333. % Declaration of new masses
parameter laL=0.01, la2=0.01.      % Declaration of new couplings
```

Declaration of constrained parameters.

```
%mu^2 as a function of masses
parameter mu2=MHX**2-laL*(2*MW/EE*SW)**2.
% constraints for couplings
parameter la3=2*(MHC**2-mu2)/(2*MW/EE*SW)**2.
parameter la5=(MHX**2-MH3**2)/(2*MW/EE*SW)**2.
parameter la4=2*laL-la3-la5.
```

New particles of the model

```
scalar '~H3'/'~H3':('odd Higgs',pdg 36, mass MH3, width wH3 = auto).
scalar '~H+'/'~H-':('Charged Higgs',pdg 37,mass MHC,width wHC=auto).
scalar '~X'/'~X':('second Higgs',pdg 35,mass MHX,width wHX=auto).
```

Now we use physical fields defined above to construct second doublet of (23)

```
let h2 = { -i* '~H+',    ('~X'+i* '~H3')/Sqrt2 },
          H2 = { i* '~H-',    ('~X'-i* '~H3')/Sqrt2 }.
```

At the next step we define covariant derivatives for new doublet. Below B1 is SM $U(1)$ gauge field and $WW=\{W-,W3,W+\}$ is SM $SU(2)$ triplet. $g1$ and g are corresponding couplings; taupm is generator of $SU(2)$ group in $\{W-,W3,W+\}$ basis.

```
let Dh2^mu^a = (deriv^mu+i*g1/2*B1^mu)*h2^a +
               i*g/2*taupm^a^b^c*WW^mu^c*h2^b.
let DH2^mu^a = (deriv^mu-i*g1/2*B1^mu)*H2^a
               -i*g/2*taupm^a^b^c*{'W-'^mu,W3^mu,'W+'^mu}^c*H2^b.
```

Terms of Lagrangian (23) can be written in LanHEP notation by

```
lterm DH2*Dh2.           % Kinematic and other terms.
lterm -mu2*h2*H2.
lterm -la2*(h2*H2)**2.
lterm -la3*(h1*H1)*(h2*H2).
lterm -la4*(h1*H2)*(H1*h2).
lterm -la5/2*(h1*H2)**2 + AddHermConj.
```

where `h1` and `H1` present the SM Higgs doublet and its conjugation.

Indeed all models used by CalcHEP were constructed with LanHEP. Compilation of LanHEP source file for CalcHEP can be done by the command

```
lhcp <source file> -ca -evl 2
```

Other packagers which facilitate the implementation of complicated models for CalcHEP are FeynRules [36] and SARAH [72].

9.4 FeynRules

10 CalcHEP as a generator of matrix elements for other packages.

Here we present tools which allow to compile Squared Matrix Elements for different processes and calculate them for needed values of input parameters and momenta. Actually we presents tools which allow to construct such packages as micrOMEGAs where matrix elements generated by CalcHEP are used for calculation of different observables related to Dark Matter. We assume that user writes a corresponding *main* program which can be compiled by

```
$CALCHEP/bin/make_main mainProgram.c
```

See example of such main program in `$CALCHEP/utile/main_22.c`. After compilation executable program *mainProgram* has to appear. Because CalcHEP uses run-time generated code for width calculation, a tool which works with CalcHEP matrix elements has to have an opportunity to generate new matrix element. If so, it is quite naturally to provide user an option to generate any matrix element in run-time. In the same time the user can link preliminary generated matrix elements as well.

10.1 Choosing of model

The first command of the main routine should be

```
int setModel(char*modelFilesDisposition,int modelNumber)
```

For example, if ones prefer to work in CalcHEP working directory, the command can be `setModel("models",1)`

The `setModel` command generates **aux** subdirectory which is organised as CalcHEP working directiry with subrirectories **models**, **results**, **tmp** and directory **so_generated** to store compiled code of matrix elements. **set-Model** has to generate **VandP.so** file in directory **so_generated** which contains compiled constrains of the model, list of variables, and list of particles. In case of problems not rezo value is returned.

The user can change a model from session to session or even during one session, but any time he changes the model, codes of matrix elements obtained before will be cleaned.

10.2 Setting of parameters and calculation of constraints.

Three functions can be used to set the value of independent parameters:

```
int assignVal(char*name,double val)
```

```
void assignValW(char*name,doube val)
```

assigns value *val* to parameter *name*. The function `assignVal` returns a non-zero value if it cannot recognize a parameter name while `assignValW` writes an error message.

```
int readVar(char*fileName)
```

reads parameters from a file. The file should contain two columns with the following format

```
name    value
```

`readVar` returns zero when the file has been read successfully, a negative value when the file cannot be opened for reading and a positive value corresponding to the line where a wrong file record was found.

After parameter assignment is completed, in has to call

```
int calcMainFunc(void)
```

routine with calculates *public* constraints. Zero return value of this routine means that all constraints were successfully calculated. It non-zero `err` value is returned then there is a problem in calculation of `varNames[err]` parameter.

The following routines are used to display the value of independent and constrained *public* parameters: `int findVal(char*name,double*val)` finds the value of variable *name* and assigns it to parameter *val*. It returns a non-zero value if it cannot recognize a parameter name.

`double findValW(char*name)` just returns the value of variable *name* and writes an error message if it cannot recognize a parameter name. The variables accessible by these commands are all free parameters and the constrained parameters of the model (in file `model/func1.mdl`) treated as *public*.

For treating of independent and constrained physical parameters the basic variables are:

```
int nModelVars;
int nModelFunc;
char**varNames; // contains nModelVars+nModelFunc+1 elements.
                // The zero one is not used
REAL *varValues; // contains nModelVars+nModelFunc+1 elements.
                // type REAL is defined in
```

Type `REAL` is defined in `$CALCHEP/../../include/nType.h`. By default `REAL` means `double`

10.3 Testing of particle contents.

```
char* pdg2name(int nPDG)
```

returns the name of the particle whose PDG code is *nPDG*. If this particle does not exist in the model the return value is `NULL`.

`int pNum(char * name)` returns PDG code of particle defined by `name`. If the input parameters does not corresponds to any particle, then return value is zero.

`int qNumbers(char*pName,int*spin2,int*charge3,int*cdim)` returns the quantum numbers for the particle `pName`. Here `spin2` is double spin of the particle; `charge3` is three times the electric charge; `cdim` is the dimension of the representation of $SU(3)_c$, it can be 1, 3, -3 or 8. The value returned is the PDG code. If `pName` does not correspond to any particle of the model then `qNumbers` returns zero.

`double * pMass(char*pName)` returns numerical value of the particle mass.

Basic variables which defines particles are

```
int nModelParticles;
ModelPrtclsStr*ModelPrtcls;
```

Structure `ModelPrtclsStr` is defined in `$CALCHEP/include/VandP.h`

10.4 Decay widths and branching fractions.

The calculation of particle widths, decay channels and branching fractions can be done by the function

`double pWidth(char*pName, txtList branchings)` returns directly the particle width. If the 1->2 decay channels are kinematically accessible then only these channels are included in the width. If not, `pWidth` compiles all open 1->3 channels and use these for computing the width. If 1->3 channels are closed too, then 1->4 channels are considered. An improved routine with a better matching between the 1->2, 1->3 and 1->4 calculations is kept for the future. The returned parameter `branchings` gives an address where information about the decay channels is stored. The `txtList` type is presented in

`$CALCHEP/c_sources/dynamic_me/include/dynamic_cs.h`

If SLHA file with decay information was used before `pWidth` call, then `pWidth` will not calculate widths using CalcHEP matrix elements, but will display information stored in the file.

`void printTxtList(txtList branchings,FILE*f)`

writes to open file branching fractions obtained by `pWidth`.

`double findBr(txtList branchings , char * pattern)`

finds the branching fraction for a specific decay channel specified in `pattern`,

a string containing the particle names in the CalcHEP notation. The names are separated by commas or spaces and can be specified in any order.

```
int slhaDecayPrint(char * pname, FILE*FD)
```

Calculates the width and branching ratios of particle *pname* and writes down the result in SLHA format. The return value is the PDG particles code. In case of problem, for instance wrong particle names, this function returns zero. This function just present another output format for `pWidth/printTxtList`.

10.5 Compilation of new processes.

Generic procedure for matrix element compilation reads

```
numout*getMEcode(
    int twidth,          // flag which forces Breit-Wigner form of t-channel
                        // propagators
    int UG,              // flag which forces Unitary gauge
    char*Process,        // process name
    char*excludeVirtual, // list of particles which forbidden in propagators
    char*excludeOut,     // list of particles which forbidden as outgoing ones
    char*libName)        // name of shared library (without terminating ".so")
```

In case of success result of the compilation is stored in the library

`aux/so-generated/libName.so.`

If the library `libName` already exists, it is not recompiled and the correspondence between the contents of the library and the *Process* parameter is not checked. *libName* is also inserted into the names of routines in the *libName.so* library. Thus *libName* can not contain symbols that cannot be used in identifiers, for example the symbols `{+ - * /}`.

The `getMEcode` return address of computer memory where compiled code and auxiliary programs are stored. If compilation was not successful then return value is `NULL`. It can be in case when input process is absent in the model. User can find definition of `struct numout` in file `dynamic_cs.h`. We will not explain this structure but give some examples of it usage.

Codes of matrix elements are not related directly with parameters of the model describe above. So, before we start to work with SQME codes we have to export numerical values of model variables to code of matrix elements. Let `cc` is a variable of `numout*` type obtained by `getMEcode` procedure. Then export of parameters can be done by the command

```
for(i=1; i<=cc->interface->nvar; i++)
    if(cc->link[i]) cc->interface->va[i]=*(cc->link[i]);
```

Number of compiled subprocesses and number of incoming and outgoing particles for these subprocesses can be detected by

```
int procInfo1(nuout*cc,int*nproc,int*nin,int*nout)
```

Particle contents for given subprocess ($1 \leq nsub \leq ntot$) can be obtained by

```
int procInfo2(numout*cc,nsub,char**pName,REAL*Masses)
```

Here `pName` and `Masses` are arrays of $nin + nout$ elements which present particle names and particle masses.

10.6 Calculation of matrix elements.

To calculate matrix element one has to fill particle momenta. Momenta are presented by one dimension array of type `REAL` which contains in turn 4-momenta of incoming and outgoing particles. 4-momenta are started from energy (zeroth) component.

Value of squared matrix element can be obtained by

```
cc->interface->sqme(nsun,GG,pvect,&err);
```

where `cc` is a pointer for compiled process; `nsub` - number of subprocess; `GG` is strong coupling ($\alpha_{qcd} = GG^2/(4\pi)$); `pvect` - array of momenta; `err` integer variable which contains error code after execution. Sumation over outgoing polarization and averaging over incoming ones is performed.

11 CalcHEP output for Reduce, Mathematica and Form

11.1 General structure

In addition to writing *C* code, which has already been described, CalcHEP can output the results of the built-in symbolic calculation in a format suitable for the *Reduce* package [45], the *Mathematica* package [73] and the *FORM* package [74]. We have attempted to present the results in a form which can easily be used for different purposes.

All the squared diagram contributions for one subprocess are stored in one file. The subprocess number is appended to the file name. For example, the `syml1.red` and `syml1.m` files contain the symbolic expressions for the squared diagrams in the first subprocess in the syntax of *Reduce* and *Mathematica*, respectively.

The output files have the following structure:

Initial declarations <code>initSum()</code>
First diagram code <code>addToSum()</code>
Second diagram code <code>addToSum()</code>
.
.
.
<code>finishSum()</code>

Initial declarations includes the declaration of variables for the momenta and conservation law relations for them, the declaration of the independent parameters involved in the calculation and their numerical values, the declaration of the dependent parameters and their substitution rules, and, finally, the declaration of the process. The momenta are named `p1`, `p2`, `p3`, ... where `p1` is the momentum of the first particle in the process, `p2` is the momentum of the second particle and so on. The signs of momenta are defined in such a way that the sum of incoming momenta is equal to the sum of the outgoing momenta. The list of substitutions of numerical values for the independent parameters is stored in the variable `parameters`. The list of substitutions for the dependent parameters is stored in the variable `substitutions`. The lists of incoming and outgoing particles is stored in the variables `inParticles` and `outParticles`, respectively.

After the initial declarations, the function `initSum()` is called. Then, the expression for each squared diagram is presented. After each squared diagram

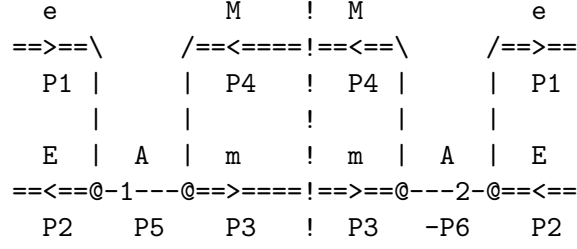


Figure 14: An example of a pseudo-graphic representation of a squared diagram found in the symbolic expression code.

code, the function `addToSum()` is called. After all the squared diagrams are finished, the function `finishSum()` is called. These three procedures (`initSum()`, `addToSum()` and `finishSum()`) must be written by the user and loaded before the the process code is loaded. In this way, the user can decide how the code from the different diagrams is combined.

Now we shall explain the structure of each squared diagram contribution. Each squared diagram begins with a pseudo-graphic image of the diagram such as in the $e^+e^- \rightarrow \mu^+\mu^-$ example found in Fig. 11.1. This is followed by the assignments:

- `totFactor` is a rational function depending on the model parameters;
- `numerator` is a polynomial of the model parameters and scalar products of the momenta;
- `denominator` is a product of the propagator denominators

`propDen(P,Mass,Width)`

where `P`, `Mass`, and `Width` are, respectively, the momentum, mass, and width of propagating particle. In the case that `Width`= 0, `propDen` should be defined as $(\text{Mass}^2 - P^2)$. The user can treat the `Width` argument as he/she likes.

The total contribution of a squared diagram is then: $\text{totFactor} \frac{\text{numerator}}{\text{denominator}}$.

We note that, as was mentioned in Section 4.4, the result obtained by the summation of the diagrams must be symmetrized in the case of identical outgoing particles. This can be done, for example, by the `finishSum()` procedure.

11.2 *Reduce* examples

We have prepared some example programs to work with the *Reduce* output. They are:

- `sum_cd.red` which combines the expressions from the squared diagrams and presents it in a form with a single common denominator;
- `sum_22.red` which combines the expressions from the squared diagrams and presents it as a sum of pole terms;
- `sum_int.red` which combines the expressions from the squared diagrams and integrates it over the phase space and presents it as a total cross section.

`sum_22.red` and `sum_int.red` only work with $2 \rightarrow 2$ processes. These files are stored in the `$CALCHEP/utile` directory.

Suppose the symbolic output (`symb1.red`) for $\gamma\mu$ scattering ($A,m \rightarrow A,m$) is prepared⁸. We copy the `sum_cd.red`, `sum_22.red` and `sum_int.red` files into our `WORK` directory and launch the *Reduce* program from within the `results` directory. We display the outcome of using *Reduce* with each of these example programs:

Example 1

If we use the `sum_cd.red` program, we will get:

```
in"../sum_cd.red"$           % to load the summation package
in"symb1.red"$               % to read the contributions of the diagrams
sum;                         % to write the answer

(32*ee**4*(2*p1.p2**4-4*p1.p2**3*p1.p3+3*p1.p2**2*p1.p3**2-2*p1.p2**2*p1.p3
*mm**2-p1.p2*p1.p3**3 + 2*p1.p2*p1.p3**2*mm**2 + p1.p3**2*mm**4))
/(propden(p1+p2,mm,0)**2*propden(p2-p3,mm,0)**2)$
```

Example 2

If we use the `sum_22.red` program, we will get:

```
in"../sum_22.red"$           % to load the summation package
in"symb1.red"$               % to read the contributions of the diagrams
sum;                         % to write the answer
```

⁸If the electron is massless, it leads to a divergence in the total cross section. For the purposes of this illustration, we therefore use a muon instead of an electron.


```
2*ee**4*(4*sp(mm)**2*mm**4 + 8*sp(mm)*up(mm)*mm**4 + 4*sp(mm)*mm**2 +
sp(mm)*t+ 4*up(mm)**2*mm**4 + 5*up(mm)*mm**2 - up(mm)*s + 1)$
```

where $s = (p_1 + p_2)^2$, $t = (p_1 - p_3)^2$ and $u = (p_1 - p_4)^2$ are the Mandelstam variables and the functions `sp`, `tp`, `up` are defined as

$$\begin{aligned} sp(x) &= 1/(s - x^2) \\ tp(x) &= 1/(t - x^2) \\ up(x) &= 1/(u - x^2) \end{aligned}$$

Example 3

If we use the `sum_int.red` program, we will get:

```
in"../sum_int.red"$           % to load the summation package
in"symb1.red"$               % to read the contributions of the diagrams
sum;                          % to write the answer for total cross section

(ee**4*(2*s**4*log(s/mm**2) + s**4 - 12*s**3*log(s/mm**2)*mm**2 + 14*s**3*mm**2
- 6*s**2*log(s/mm**2)*mm**4 - 16*s**2*mm**4 + 2*s*mm**6 - mm**8))/(16*s
**2*pi*(s**3 - 3*s**2*mm**2 + 3*s*mm**4 - mm**6))$
```

Sometimes the result for the total cross-section includes a complicated expression of kinematic variables under the square root which appears as a result of the evaluation of the integrand limits:

$$be_ = (t_{max} - t_{min})/s$$

where s and t are Mandelstam variables. Substitution for `be_` is not done. Instead variable `beSquared = be_**2` is defined.

11.3 *Mathematica* examples

There following packages for *Mathematica*: `sum_cd.m`, `sum_22.m` and `sum_int.m` perform the same task as the *Reduce* packages did (see the previous subsection.) The following output was performed in *Mathematica* 3.0. `SC[pi,pj]` is the scalar product of momentum `pi` and `pj`.

Example 1

If we use the `sum_cd.m` program, we will get:

```
In[1] := <<"../sum_cd.m"
In[2] := <<"symb1.m"
```

```
In[3]:= sum
```

```
Out[3]= (32 EE (2 SC[p1, p2]4 - 4 SC[p1, p2]4 SC[p1, p3]3 + Mm4 SC[p1, p3]2 +
> SC[p1, p2]2 (2 Mm2 - SC[p1, p3]) SC[p1, p3]2 +
> SC[p1, p2]2 SC[p1, p3]2 (-2 Mm2 + 3 SC[p1, p3])) /
> (propDen[-p1 - p2, Mm, 0]2 propDen[p2 - p3, Mm, 0]2)
```

Example 2

If we use the `sum_22.m` program, we will get:

```
In[1]:= <<"../sum_22.m"
```

```
In[2]:= <<"symb1.m"
```

```
In[3]:= sum
```

```
Out[3]= 
$$\frac{2 EE (Mm^2 + s)^2}{(Mm^2 - s)^2} - \frac{2 EE t^4}{Mm^2 - s^2} + \frac{8 EE Mm^4}{(-Mm^2 + s + t)^2} +$$

> 
$$\frac{2 (3 EE Mm^4 + 6 EE Mm^2 s - EE s^4)}{(Mm^2 - s)^2 (-Mm^2 + s + t)^2}$$

```

Example 3

If we use the `sum_int.m` program, we will get:

```
In[2]:= <<"../sum_int.m"
```

```
In[3]:= <<"symb1.m"
```

```
In[4]:= sum
```

$$\text{Out}[4] = \frac{(E E^4 (-M_m^8 + 2 M_m^6 s - 16 M_m^4 s^2 + 14 M_m^2 s^3 + s^4 - 2 s^2 (-3 M_m^4 - 6 M_m^2 s + s^2) \text{Log}[\frac{M_m^2}{s}]))}{(16 \pi s^2 (-M_m^2 + s)^3)}$$

11.4 Form

11.5 *Reduce* program

The original output of CalcHEP was for the *Reduce* program. CalcHEP would generate the squared diagram expressions and have *Reduce* expand, contract indices, trace gamma matrix chains and simplify the expressions. Later on, when the CalcHEP build-in symbolic calculator was created, the *Reduce* output became unnecessary. However, we still keep this output for testing purposes.

The user does not see the internal details of the CalcHEP symbolic calculator and, so, it appears as a black box. On the other hand, the *Reduce* code that CalcHEP produces is written in a form that can be understood by humans and followed step-by-step in *Reduce*. The results of the *Reduce* calculation can then be compared with the results of the CalcHEP symbolic calculator as a test of its correctness (see Appendix B for details about the self tests.)

In this subsection, we describe the structure of the *Reduce* output. The Feynman rules used in this output can be found in Appendix D.

CalcHEP generates a separate file for each squared diagram. The files are named pNNN-MMM.red where NNN is the subprocess number and MMM is the diagram number.

Each file begins with a declaration of the momenta and Lorentz indices. For example,

```
% ----- VARIABLES -----
vector  A,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,ZERO_;
vector  m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16;
%
%----- Mass shell declarations -----
MASS  P1 = 0$  MSHELL P1$
MASS  P2 = 0$  MSHELL P2$
MASS  P3 = Mm$ MSHELL P3$
```

```
MASS P4 = Mm$ MSHELL P4$
```

```
%----- Momentum substitutions -----
Let p4 = +p1+p2-p3$
Let p5 = +p1+p2$
Let p6 = -p1-p2$
```

The vector **A** is used by the *Reduce* package to construct the γ^5 matrix as in $\gamma^5 = G(1n, A)$. The vectors whose names begin with **p** are used for the momenta. The vectors whose names begin with **m** are used for the Lorentz indices. After the vector declarations comes the mass-shell declarations for the incoming and outgoing particles. This is followed by substitution rules that use momentum conservation to replace the internal momenta with the external momenta.

The next thing in the file is a multiplicative factor for the diagram which includes a symmetry factor, a factor for averaging over the initial state helicities, a factor for the number of incoming fermions, a QCD factor and some model parameters that were factored out of the diagram expression. For example, in the case of $e^+e^- \rightarrow \mu^+\mu^-$ we have

```
%----- Factors -----
SymmFact:=1/1$ % Diagram symmetry factor
AverFact:=1/4$ % Normalization factor of polarization average
FermFact:=1$ % (-1)**(number of in-fermion particles)
ColorFact:=1/1$ % QCD color weight of diagram
%
totFactor_:=EE**4$
totFactor_:=totFactor_*SymmFact*AverFact*FermFact*ColorFact$
```

For the most part, these declarations are self-explanatory except, perhaps, for **SymmFact**. Generally, **SymmFact** = N/D where N is equal to 1 when the two diagrams in the squared-diagram are identical. Otherwise, this factor equals 2. D is a factorial connected with the presence of identical outgoing particles and partially reduced by a number of various possibilities to assign the momenta of the outgoing particles to the corresponding diagram lines.

Each file includes both the main squared diagram and any that are related to it by replacement of a gauge boson with a Faddeev-Popov ghost or a Goldstone boson according to the Feynman rules for the model. In unitary gauge, these are removed. We note that all these diagrams have the same denominator. The evaluation of the expression for the diagram is started with an initialization of a variable for the sum of the numerators for these diagrams:

```
numerator_:=0
```

This is followed by each diagram in this set. The code for each diagram is preceded by a pseudo-graphical representation of the squared diagram (for example, see Fig.11.1.) In these pseudo-graphical representations, the name of each particle and its corresponding momentum are written down near its line. Its Lorentz index is written in the line.

The code for each diagram is started with the fermion loop evaluation. CalcHEP moves along each fermion line, multiplying the vertex and propagator terms. The `nospur` instruction is declared before each loop evaluation to prevent *Reduce* from prematurely tracing the gamma matrix chains.

If the result of this multiplication contains Lorentz indices which can be contracted, the program declares the corresponding vectors as indices by means of the `index` instruction.

CalcHEP begins moving along the fermion lines in the direction opposite to the direction of the fermion flow arrows. When it encounters a vertex, it looks it up in the `Vertex` table. If it is moving against the fermion flow direction and the incoming fermion comes after the antifermion in the `Vertex` table, then CalcHEP enters the Lorentz part directly in the expression. If, on the other hand, the fermion comes before the antifermion in the `Vertex` table, then CalcHEP first transforms the vertex by means of the rule given in Eq. (10) and writes a comment stating that it has done this. For example, consider the vertex of a u , \bar{d} and a W^- . Suppose it is encoded in the `Vertex` table as `|D |u |W- |` (and its hermitian conjugate is encoded as `|U |d |W+ |`). If CalcHEP is moving against the fermion flow direction on a u quark line and because the `u` comes after the `D` in the `Vertex` table, the Lorentz piece is inserted without change. If, however, the vertex was encoded as `|u |D |W- |`, then the Lorentz piece would first be reversed, using Eq. (10), before being inserted into the expression.

If, on the other hand, CalcHEP is moving with the fermion flow direction (perhaps after a Majorana or fermion number violating vertex) and the incoming antifermion comes after the fermion in the vertex, CalcHEP, again, enters the Lorentz part directly. If, however, the antifermion comes before the fermion, CalcHEP first transforms the vertex by means of Eq. (10) and writes a comment about the transformation. Using the same example vertex above, if CalcHEP is moving with fermion flow on a u quark line and the `U` comes before the `d` (as in `|U |d |W+ |`), the Lorentz piece is first reversed before insertion.

Because a Majorana fermion is its own antiparticle, the order does not matter in the case of a Majorana fermion vertex and the Lorentz piece is inserted directly. The same is true for a *C-conjugate* operator.

Once the fermion chain is finished, the `spur` switch is restored and *Reduce* performs the γ -matrix trace. CalcHEP multiplies this trace by -4 . The 4 is required because *Reduce* leaves this out of the gamma matrix trace. The -1 is due to the Feynman rules.

The next step is the multiplication of the other vertices and the contraction of the Lorentz indices. The Lorentz indices that need to be contracted are declared by the `Index` instruction before the multiplication. An example of this fragment of code is:

```
Index m2$
Vrt_3:=Vrt_3*Vrt_5$
RemInd m2$
```

This code instructs *Reduce* that vertex 3 is multiplied by vertex 5 and that Lorentz index `m2` is contracted. The result of this multiplication is stored in the variable `Vrt_3`.

In order to make the *Reduce* code produced by CalcHEP manage memory better, we declare the symbols `m` as vectors indices step by step as we perform the calculation.

If two vertices are connected by the propagator of a massive vector particle treated in unitary gauge (see Eq. 11), the code is organized in the following way. CalcHEP first writes the code for the product with the indices contracted (the $g_{\mu\nu}$ piece of the propagator). It then writes the same product but with the $p_\mu p_\nu / M^2$ piece of the propagator. It then adds the pieces together. Below, we present an example of such a piece of code:

```
Index m1$
Vrt_0:=Vrt_1*Vrt_2$
RemInd m1$
Vrt_L:=Vrt_1$   Vrt_R:=Vrt_2$
Vrt_L:=(Vrt_L where m1=>(+P2+P3)/MZ)$
Vrt_R:=(Vrt_R where m1=>+(-P2)+(-P3))/MZ)$
Vrt_0:=Vrt_0 + Vrt_L*Vrt_R$
```

The end of the code for any diagram contains the instruction

```
numerator_:=numerator_ +DiagramFactor*GhostFact*Vrt_1
```

which adds it's contribution to the total. It also multiplies by the ghost factor, if any, where $\text{GhostFact} = (-1)^{l+v}$ where l is the number of loops of Faddeev-Popov ghosts and v is the number of vector field lines in the diagram. The $(-1)^v$ factor appears because the evaluations described above

correspond to substituting $(g_{\mu\nu} - k_\mu k_\nu / M^2)$ for the propagator of the vector field whereas the correct expression has the opposite sign.

The last step of the program is the assignment of the variable `denominator_`. It is expressed as a product of the terms `propDen(P,Mass,Width)` as explained in Subsection 11.

Finally, the total result for the evaluated squared diagram is given by:

$$totFactor \times \frac{numerator}{denominator}$$

Appendix

A \LaTeX output

CalcHEP uses the *Axodraw* package by J.A.M. Vermaseren [44] to create diagrams and plots in \LaTeX format. To use this package the *Axodraw* style should be included in the *documentstyle* statement. For example,

```
\documentstyle[axodraw]{article}
```

With kind permission of the author, we include a copy of the *axodraw.sty* file in the $\$CALCHEP$ directory.

The *Axodraw* syntax is straight forward and the user can modify the CalcHEP output to fine-tune the picture. Other changes the user can make are to the line width, the scale of the picture, and the size of the characters. The \LaTeX and *Axodraw* instructions are located at the beginning of the output. For example,

```
{ \small           % letter size control
  \SetWidth{0.7}    % line width control
  \SetScale{1.0}     % line scale control
  \unitlength=1.0 pt % text position control
  .....
}
```

Note that the `\SetScale` instruction influences the position of the lines, whereas the `\unitlength` variable is responsible for the position of text. Consequently, if the user would like to change the scale of the picture, he/she has to modify both of these settings in a consistent manner. For instance, to increase the size of the picture by a factor of two, use the following:

```
\SetScale{2.0}      % picture size control
\unitlength=2.0 pt  % picture size control
```

In the case of Feynman diagram output, CalcHEP substitutes the \LaTeX names of the particles as they are defined in the `Particle` table (see Section 8.3).

B Self-check of the CalcHEP package

We have included a suite of tools for testing the CalcHEP package. Positive results from these tests signal that the CalcHEP package is working properly. The tests described here involve the symbolic calculation and have been realized with the help of the *Reduce* [45] symbolic manipulation system.

These test routines are stored in the `$CALCHEP/utile` directory. To use these tests, the user should, first, copy the test files into his/her working directory. The commands to run the tests should be executed from the same directory as the location of the test files (in the users working directory.)

B.1 Check of the built-in symbolic calculator

The first check is a comparison of the results of the CalcHEP symbolic calculator (see Section 11) against the results of the *Reduce* calculator (see Section 11.5). We take agreement between these results to mean that the CalcHEP symbolic calculator is working correctly. We note that the results of the *Reduce* code created by CalcHEP for the test can be viewed step by step as the calculation is performed by *Reduce*. The result can then be compared with the result of the CalcHEP symbolic calculator.

This check is realized by means of the program `check.red` which must be started from within a *Reduce* session by the command

```
in "check.red";
```

When doing this, it is assumed that the *Reduce* code for the diagrams and the corresponding symbolic expressions generated by CalcHEP are stored in the `results` directory in advance. The results of this check are saved in the `message` file. It consists of a list where each line contains a diagram number accompanied by the label `OK` or `Error` depending on the result of the comparison.

B.2 Comparison of results produced in two different gauges

A comparison of results produced in unitary gauge and t'Hooft-Feynman gauge is a very important, nontrivial test. Not only does this test check the CalcHEP symbolic calculator, but also the model implementation. In fact, we suggest that a test of gauge invariance always be performed for any new model implementation.

To perform this test, we evaluate the symbolic sum of squared diagrams in the two supported gauges (unitary and t'Hooft-Feynman) for the SM and compare the results. If the difference between these two calculations is non-zero, there is a mistake in the symbolic calculator or the model. The symbolic summation of the squared diagrams is performed by *Reduce*. This summation is the most difficult step of comparison because the sum of diagrams can be an extremely large complicated expression which is not easily simplified.

To perform this test, a process should be calculated in unitary gauge and the results should be stored in `results_/syml1.red`. Then, the same process should be calculated in t'Hooft-Feynman gauge and the results should be stored in `results/syml1.red`. Finally, *Reduce* should be started from the work directory (`results_` and `results` should be subdirectories of the work directory) and the program `cmp.red` should be read in as in

```
in "cmp.red";
```

This program will read in the expressions from `results_/syml1.red` and sum the squared diagrams. It will then read the expressions from `results/syml1.red` and sum the squared diagrams. It will then take the difference of the two expressions. If, after simplification, the difference is zero, it will print OK in the `message` file. Otherwise, it will write `Error`.

C Ghost fields and the squared diagram technique for the t'Hooft-Feynman gauge

C.1 The problem

Whenever we implement a new model with higher spin, we encounter a problem with asymptotic behavior of its propagator at high energy. This can be solved in the case of a spin-1 gauge boson by use of the Goldstone bosons and Faddeev-Popov ghosts associated with it as we describe in this appendix. The unitary gauge propagator for a massive vector boson is given by

$$\frac{i}{(2\pi)^4} \frac{g_{\mu\nu} - k_\mu k_\nu / m^2}{m^2 - k^2}. \quad (24)$$

where the $(g_{\mu\nu} - k_\mu k_\nu / m^2)$ projects out the unphysical polarizations and leaves only the physical polarizations. This projection is necessary because we have used a 4-component vector field to describe a particle with 3 degrees of freedom. Unfortunately, the $k_\mu k_\nu / m^2$ term leads to a quadratic growth of the amplitude at high energies which, if left uncanceled, leads to a violation of unitarity and renormalizability.

This problem is solved for vector fields in the framework of gauge field theories where the gauge symmetry leads to a cancellation between diagrams of the bad high energy growth [48]. However, numerical calculations have finite precision. If the cancellation is very large, although the cancellation is perfect in theory, the finite precision arithmetic can lead to partial cancellation and, consequently, incorrect results. An accompanying problem is

that a symbolic calculation can lead to complicated expressions with the appearance of mutually cancelling terms.

On the other hand, there is a freedom in the formulation of the Feynman rules for gauge theories resulting from an ambiguity in the gauge fixing terms [48]. These terms modify the quadratic part of the Lagrangian and, consequently, may improve the vector particle propagator. Indeed, in the case of t'Hooft-Feynman gauge, the propagator for vector particles takes the form

$$\frac{i}{(2\pi)^4} \frac{g_{\mu\nu}}{m^2 - k^2} . \quad (25)$$

where the term with the bad high energy growth ($k^\mu k^\nu / m^2$) is absent.

The price for this solution is the appearance of three additional unphysical particles in the model. They consist of two Faddeev-Popov ghosts and one Goldstone boson. All three of them have scalar type propagators (however the ghosts have the opposite sign) with the same mass (m) as the gauge boson. In contrast to Eq. (24), the propagator given by Eq. (25) is not orthogonal to the temporal polarization state

$$e^0 = k/m , \quad (26)$$

which is another way to see the appearance of additional unphysical state. The main principle of gauge invariance guarantees [48] that an expression for the amplitude should be the same for any gauge if only physical incoming and outgoing states are considered.

C.2 Incoming and outgoing ghosts

Generally, the t'Hooft-Feynman gauge solves the problem of large cancellations for the internal states. But, when calculating processes with incoming or outgoing massive vector particles, we meet a similar problem in the external states. Each diagram is multiplied by the vector boson polarization vectors. These polarization vectors (e^1, e^2, e^3) constitute an orthonormal basis in the sub-space orthogonal to the vector boson momentum k . By considering the relation

$$e_\mu^1 e_\nu^1 + e_\mu^2 e_\nu^2 + e_\mu^3 e_\nu^3 = k_\mu k_\nu / m^2 - g_{\mu\nu} \quad (27)$$

we can see that at least one of the polarization vectors grows as k/m for any choice of polarization basis. Let vector k have the components

$$k = (\sqrt{m^2 + p^2}, 0, 0, p) . \quad (28)$$

Then, the polarization vectors can be chosen as

$$e^1 = (0, 1, 0, 0) ; \quad (29)$$

$$e^2 = (0, 0, 1, 0) ; \quad (30)$$

$$e^3 = (p/m, 0, 0, \sqrt{1 + p^2/m^2}) , \quad (31)$$

The first two vectors correspond to spatially transverse polarizations while the third corresponds to the longitudinal polarization. We see that the longitudinal polarization grows as k/m and becomes large for large momentum. It may imply a rapid increase in the cross-section in processes with longitudinal polarizations at high energies for effective theories or the appearance of large cancellations between various diagrams. Indeed, for a gauge theory, the second case is realized and, hence, we again have a problem with finite precision calculations. As in the previous case, we see that the problem is related to the projection operator in Eq. (27) and wonder whether we might use gauge invariance to solve this problem too.

One solution to this problem is similar to a chess sacrifice. The main idea is to include unphysical polarizations in the incoming and outgoing states and cancel them by use of the Faddeev-Popov ghosts and Goldstone bosons. We consider the Faddeev-Popov ghosts and Goldstone boson states as new polarizations, similar to the temporal polarization (see Eq. (26).) We note that the Faddeev-Popov ghost states have a negative norm as does the temporal polarization, whereas the Goldstone boson state has a positive norm [48]. As a result, the unphysical fields can give both a positive and a negative contribution to the polarization sum.

The main point is that the sum of the contributions of all unphysical states and unphysical polarizations to the squared matrix element vanishes [75]. As a result,

$$\sum_{i \in S_{phys}} A_i A_i^* = \sum_{i \in S_{all}} \sigma(i) A_i A_i^* , \quad (32)$$

where i is a multi-index for the polarization states, A_i is the amplitude of the process, S_{phys} is the set of physical polarization states, S_{all} is the full set of physical polarizations, unphysical polarizations and unphysical states, and $\sigma(i) = \pm 1$ depending on the signature of the Hilbert space norm for the polarization state i .

A drawback of enlarging the set of polarization states is that we have a much greater number of squared matrix element terms for evaluation and subsequent summation. However, to see the advantage of this trick, let us sum the contributions from the temporal polarization (Eq. (26)) and the longitudinal polarization (Eq. (31)). We note each of them, separately, grow

as k/m , but together, that k/m growth cancels

$$e_\mu^0 e_\nu^0 - e_\mu^3 e_\nu^3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \quad (33)$$

which can be achieved in a different basis

$$e_\mu^0 e_\nu^0 - e_\mu^3 e_\nu^3 = e_\mu'^0 e_\nu'^0 - e_\mu'^3 e_\nu'^3, \quad (34)$$

where

$$\begin{aligned} e'^0 &= (1, 0, 0, 0); \\ e'^3 &= (0, 0, 0, 1). \end{aligned}$$

In this new polarization basis, none of the polarization vectors grow with energy. As a result, by inclusion of the temporal polarization, the sum over polarization squares (see Eq. (27)) becomes

$$e_\mu^0 e_\nu^0 - e_\mu^1 e_\nu^1 - e_\mu^2 e_\nu^2 - e_\mu^3 e_\nu^3 = -g_{\mu\nu}. \quad (35)$$

which, again, does not lead to the bad high energy growth and loss of precision. In the case of an unpolarized calculation, we simply replace the sum over polarizations by $-g_{\mu\nu}$ as in Eq. (35). In a gauge invariant theory, the unphysical polarizations are canceled if we add Faddeev-Popov ghosts and Goldstone bosons to the external states and we get the same result as if we only included the physical polarizations. However, now we have better behavior at high energy and much less loss from finite numerical precision.

C.3 Massless vector-particle case

Now that we have discussed the massive vector boson case, we turn to the massless gauge boson. It, also, has a gauge symmetry which must be satisfied in the calculation of the squared matrix element. The Feynman gauge leads, in this case, to the propagator of Eq. (25), with $m = 0$. As in the case of the massive gauge bosons, the massless gauge bosons, also, have Faddeev-Popov ghosts [48]. However, in distinction to the massive case, there are no Goldstone bosons associated with the massless gauge boson and the longitudinal polarization becomes unphysical like the temporal polarization. We can still extend the summation over physical polarizations to an enlarged set of polarization as in Eq. (32) (see the proof in [76]).

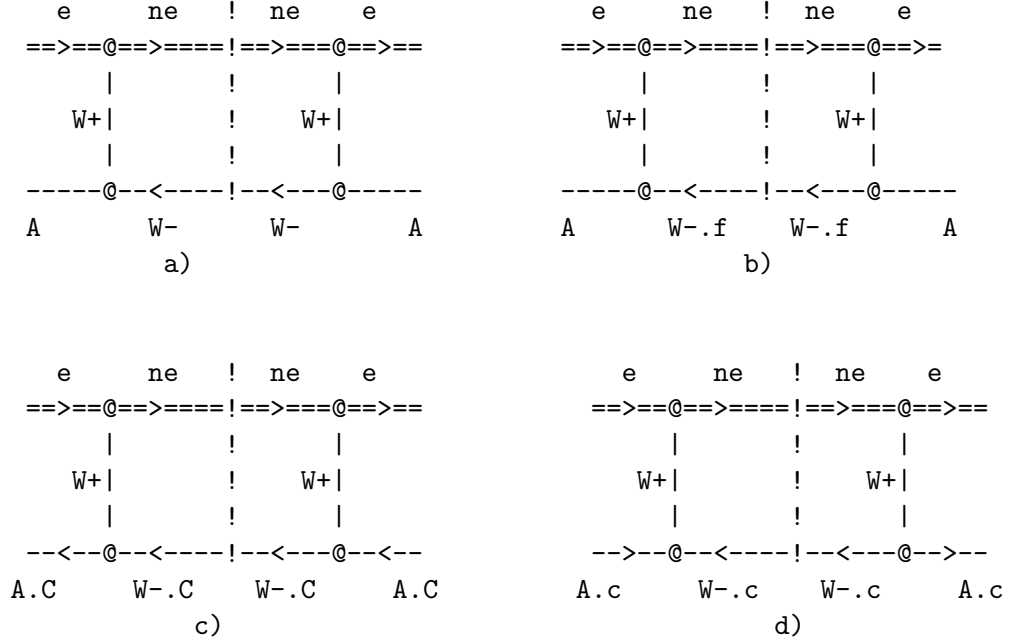


Figure 15: Ghost diagrams

C.4 Summation of ghost diagrams in CalcHEP

CalcHEP uses the squared diagram technique with a summation over the polarizations. According to our counting, one squared diagram corresponds to

$$\sum_{i \in S_{all}} A_i^k A_i^{*l} \quad (36)$$

in a squared matrix element, where A^k and A^l are the amplitudes of two Feynman diagrams in the set of all Feynman diagrams and i counts the polarizations and unphysical states as described in the previous subsections.

Strictly speaking, when gauge bosons are replaced by unphysical states, we have a much larger set of diagrams. For example, consider one squared diagram for the process $e^-, \gamma \rightarrow n_e, W^-$. The W^- is associated with two ghost and one Goldstone boson. Altogether, there are four diagrams (see Fig. 15) associated with this one parent diagram (see Fig. 15a). They all have the same topology, and in fact, they all have the same denominators. The numerators of these expanded diagrams are polynomials in scalar products of the momenta and the powers of these polynomials are the same for each expanded diagram. As a result, we might expect that after being summed together, the total expression is roughly the same size as each individual

expression. CalcHEP takes advantage of this by summing up the expanded diagram expressions and writes just one numerical code for this set.

C.5 Gauge symmetry and cancellations

Cancellation of diagram contributions is an essential point, both for symbolic and numerical calculations, because a relatively small variation of one diagram contribution may lead to a significant error. Such variations can be caused either by finite precision calculations of floating point operations or by modifications of Feynman rules, for instance, by including particle widths or by removal of some diagram subset from the calculation. We would like to stress the importance of these challenges to the user and encourage him/her to think carefully about his/her calculation.

There are two well known examples of gauge cancellations. The first is the ultraviolet cancellation of quickly growing terms originating from the propagators of massive vector particles. This problem can be resolved by the use of t'Hooft-Feynman gauge in the calculation of the squared matrix element, as described in Appendix C.

The second example is the cancellation of double pole terms of the t-channel photon propagator (of the form t^{-2}). For example, there is a wide class of processes where the incoming electron continues in the forward direction emitting a virtual photon, as in Fig.17. These diagrams have a $1/t$ pole, where t is the squared momentum of the virtual photon. For the described kinematics, the photon appears very close to its mass shell ($t \approx 0$), hence this configuration gives a respectively large contribution to the cross section.

However, after summing the diagrams in the squared matrix element, we expect the $1/t^2$ pole to be reduced to a $1/t$ pole [40] in the zero electron-mass limit. This cancellation is caused by electro-magnetic U(1) gauge invariance. If diagrams similar to that in Fig.17 contribute to your process, we strongly recommend to set the `Gauge invariance` switch to `ON` (see Section 5.5) to prevent the breaking of the gauge symmetry by width terms. Another way to solve this problem is by using the Weizsaecker-Williams approximation (see Appendix G.2).

D Feynman rules in CalcHEP

D.1 Lorentz part of diagram

Fermion loop calculation. Our algorithm for the evaluation of fermion loops takes into account the possible appearance of vertices with fermion

number violation. CalcHEP chooses an arbitrary direction for the multiplication of vertices on a fermion line. The vertices and propagators are multiplied according to this direction in order to evaluate the gamma-matrix trace. For those vertices which have a fermion line coming in, the second fermion is taken in the form it is presented in the **Vertices** table (see Subsection 8.4). Otherwise, we first rewrite them according to Eq. (10).

There are two kinds of fermion propagators: $\langle \psi(p1)\bar{\psi}(p2) \rangle$ and $\langle \psi^c(p1)\bar{\psi}^c(p2) \rangle$. In the case of Dirac or Majorana fermions, both of them are equal to

$$\langle \psi(p1)\bar{\psi}(p2) \rangle = \langle \psi^c(p1)\bar{\psi}^c(p2) \rangle = (\not{p}_1 + M)\Delta_c(p1, p2, M)$$

where Δ_c is the scalar part of the propagator. The propagators, thus, provide a factor of $(\not{p} + M)$ to a fermion line. The sign of p depends on the line direction.

There is an exception to this rule in the case of a pure left/right handed, massless fermion, which has the propagator:

$$\langle \psi_{l/r}(p1)\bar{\psi}_{l/r}(p2) \rangle = \frac{\not{p}_1(1 \mp \gamma_5)}{2}\Delta_c(p1, p2, 0)$$

and

$$\langle \psi_{l/r}^c(p1)\bar{\psi}_{l/r}^c(p2) \rangle = \frac{\not{p}_1(1 \pm \gamma_5)}{2}\Delta_c(p1, p2, 0).$$

Furthermore, the result of the trace evaluation is multiplied by (-1) .

We should also remark that in the case that a fermion vertex contains two identical fermions, the Wick contraction can be done in two ways. We remind the user (see Section (8.4)) that the expressions presented in the **Vertices** table of the model corresponds to the functional derivative of the Lagrangian. Therefore, the symmetry property and the factor of 2 should already be present there. Consequently, a special treatment of these vertices by CalcHEP is not needed. This also applies to other vertices. The combinatorial factors, in the case of identical particles, should already be present in the **Vertices** table.

We, also, note that a loop of Faddeev-Popov ghosts gives an extra factor (-1) . For convenience, CalcHEP uses a vector propagator with the wrong sign to achieve this. The total sign of a diagram is corrected at the end of the evaluation. Namely, the diagram is multiplied by $\mathbf{GhostFact} = (-1)^{n_g + n_v}$, where n_g is the number of Faddeev-Popov loops and n_v is the number of vector particles.

D.2 Color factor

We, here, explain the method [77] of color factor evaluation that is used by CalcHEP in the case of the QCD $SU(N_c)$ group. We shall use the designations *gluon* (g), *quark* (q), and *anti-quark* (\bar{q}) for colored particles which belong to the adjoint, fundamental and conjugate fundamental representations, respectively. Following section (8.4), the color part of the ggg vertex is given by the group structure constant $-i \cdot f_{abc}$, whereas, the color part of the $gq\bar{q}$ vertex is given by the fundamental representation matrix $\hat{\tau}_a$ of the group⁹.

By means of the basic relations,

$$\hat{\tau}_a \hat{\tau}_b - \hat{\tau}_b \hat{\tau}_a = i \cdot f_{abc} \hat{\tau}_c$$

and

$$tr(\hat{\tau}_a \hat{\tau}_b) = \frac{1}{2} \delta_{ab}$$

the ggg vertex can be expressed in terms of the $gq\bar{q}$ vertex:

$$-i \cdot f_{abc} = 2 \cdot tr(\hat{\tau}_b \hat{\tau}_a \hat{\tau}_c) - 2 \cdot tr(\hat{\tau}_a \hat{\tau}_b \hat{\tau}_c) \quad (37)$$

This vertex relation can also be achieved by a diagram substitution where the ggg vertex is replaced by two diagrams with a quark loop:

$$= 2 \qquad -2 \quad (37a)$$

The QCD group also has the following Fiertz identity

$$(\tau_a)_j^i (\tau_a)_l^k = \frac{1}{2} \delta_l^i \delta_j^k - \frac{1}{2N_c} \delta_j^i \delta_l^k \quad (38)$$

which, at the diagram level, can be presented with a substitution which removes gluon lines connecting quarks:

$$= \frac{1}{2} \qquad - \frac{1}{2N_c} \quad (38a)$$

⁹ $\hat{\tau}_a = \hat{\lambda}_a/2$, where λ_a are the Gell-Mann matrices.

The final result is that we have only closed, separated lines of quark color flows which are easily evaluated:

$$= \text{tr}(\hat{1}) = N_c \quad (39)$$

However, there is a complication in the separate determination of the color and the Lorentz parts of the ggg vertex. This vertex contains a symmetry corresponding to the permutation of identical legs. Altogether, it contains a product of an anti-symmetric Lorentz part and an anti-symmetric color part. Thus, in the case of separate calculations of the Lorentz and the color parts, one has to take care that the orientation of the legs in the color and the Lorentz diagrams are identical.

In the $N_c \rightarrow \infty$ limit, the tree-level squared matrix element has the asymptotic form

$$N_c^{(n_q+n_{\bar{q}}+n_g)/2}(C + O(1/N_c)) \quad (40)$$

where n_q , $n_{\bar{q}}$, n_g are the numbers of quarks, ant-quarks and gluons in the diagram. CalcHEP provides the user with the possibility to calculate matrix elements in the $N_c \rightarrow \infty$ limit where only the leading term of Eq. (40) is used. After the extraction of the leading term, we substitute $N_c = 3$. A significant subset of the interference diagrams are zero in this limit. Thus, this approach simplifies the calculation. Moreover, the precision of this approach is not too bad, especially in cases of QCD processes where loop corrections are at the same order. See Section (4.6) for more details about this option.

D.3 Common factors.

The final squared diagram is multiplied by a set of factors which are¹⁰:

- **SymmFact**= n/d , where $n = 1$ if the left part of the squared diagram is the same as the right part. Otherwise, $n = 2$. The denominator d equals the number of symmetric permutations of identical outgoing particles.
- **AverFact**= $1/N_a$, where N_a is the number of polarization and color states of the incoming particles.
- **FermFact**= $(-1)^{N_f}$, where N_f is a total number of incoming fermions and anti-fermions.

¹⁰In notations of section (11.5).

E Examples of model realization.

E.1 Implementation of QCD Lagrangian

3-gluon vertex. The QCD Lagrangian contains the following 3-gluon vertex:

$$S_{3G} = -\frac{g}{2} \int (\partial_\mu G_\nu^\alpha - \partial_\nu G_\mu^\alpha) f_{\beta\gamma}^\alpha G_\mu^\beta G_\nu^\gamma d^4x = -g \int \partial^{\mu_2} G_{\mu_1}^{\alpha_1} g^{\mu_1\mu_3} f_{\alpha_1\alpha_2\alpha_3} G_{\mu_2}^{\alpha_2} G_{\mu_3}^{\alpha_3} d^4x ,$$

where G_μ^α is the gluon field and g is the strong coupling constant. Applying the Fourier transformation (8) we get

$$S_{3G} = (2\pi)^4 g \int \delta(p_1+p_2+p_3) i f_{\alpha_1\alpha_2\alpha_3} p_1^{\mu_2} g^{\mu_1\mu_3} G_{\mu_1}^{\alpha_1}(p_1) G_{\mu_2}^{\alpha_2}(p_2) G_{\mu_3}^{\alpha_3}(p_3) d^4p_1 d^4p_2 d^4p_3 .$$

This vertex contains three identical fields, so the calculation of the functional derivatives gives us six terms:

$$\frac{\delta S_{3g}}{\delta G_{\mu_1}^{\alpha_1}(p_1) \delta G_{\mu_2}^{\alpha_2}(p_2) \delta G_{\mu_3}^{\alpha_3}(p_3)} = (2\pi)^4 \delta(p_1 + p_2 + p_3) i f_{\alpha_1\alpha_2\alpha_3} g(p_1^{\mu_2} g^{\mu_1\mu_3} - p_3^{\mu_2} g^{\mu_1\mu_3} + p_2^{\mu_3} g^{\mu_1\mu_2} - p_1^{\mu_3} g^{\mu_1\mu_2} + p_3^{\mu_1} g^{\mu_2\mu_3} - p_2^{\mu_1} g^{\mu_2\mu_3}) .$$

Comparing this expression with it's vertex representation in CalcHEP (8.4) where the *ColorFactor* is $(-i f_{\alpha_1\alpha_2\alpha_3})$, we get

$$Factor \cdot LorentzPart = -g \left((p_1^{\mu_2} - p_3^{\mu_2}) g^{\mu_1\mu_3} + (p_2^{\mu_3} - p_1^{\mu_3}) g^{\mu_1\mu_2} + (p_3^{\mu_1} - p_2^{\mu_1}) g^{\mu_2\mu_3} \right) .$$

CalcHEP uses the notation GG for the strong coupling constant g . So, for the 3-gluon vertex in CalcHEP format, we have

A1	A2	A3	A4	Factor	Lorentz part
G	G	G		GG	m1.m2*(p1-p2).m3+m2.m3*(p2-p3).m1+m3.m1*(p3-p1).m2

Quark-gluon interaction. The interaction of a gluon with a quark is described by the following action:

$$S_{QqG} = g \int G_\mu^\alpha(x) \bar{q}(x) \gamma^\mu \hat{t}_\alpha q(x) d^4x .$$

Applying the Fourier transformation and substituting $\bar{q} = q^+ \gamma^0$, we get

$$S_{QqG} = g(2\pi)^4 \int \delta(p_1 + p_2 + p_3) G_\mu^\alpha(p_3) \bar{q}(p_1) \gamma^\mu \hat{t}_\alpha q(p_2) d^4p_1 d^4p_2 d^4p_3 ;$$

and

$$\frac{\delta S_{QqG}}{\delta \bar{q}_i(p_1) \delta q^j(p_2) \delta G_\mu^\alpha(p_3)} = g(2\pi)^4 \delta(p_1 + p_2 + p_3) (\hat{t}_\alpha)_j^i \gamma^{\mu_3} .$$

The factor $(2\pi)^4 \delta(p_1 + p_2 + p_3) (\hat{t}_\alpha)_j^i$ is substituted by CalcHEP automatically. The factor C^{-1T} appears in (8.4) according to (9) and is, also, substituted by CalcHEP. Thus, the quark-gluon interaction is implemented in the CalcHEP **Vertex** table as:

A1	A2	A3	A4	Factor	Lorentz part
Q	q	G		GG	G(m3)

where q and Q are the designations for a quark and an antiquark.

Interaction of ghosts with gluon. This interaction is described by the term

$$S_{\bar{c}cG} = -g \int \bar{c}_\alpha(x) \partial^\mu (f_{\beta\gamma}^\alpha G_\mu^\beta(x) c^\gamma(x)) d^4x .$$

Fourier transformation and subsequent evaluation of the functional derivatives gives

$$S_{\bar{c}cG} = -g(2\pi)^4 \int \delta(p_1 + p_2 + p_3) \bar{c}_\alpha(p_1) (-i p_2 - i p_3)^{\mu_3} f_{\beta\gamma}^\alpha G_{\mu_3}^\beta(p_3) c^\gamma(p_2) d^4p_1 d^4p_2 d^4p_3$$

and

$$\frac{\delta S_{\bar{c}cG}}{\delta \bar{c}_{\alpha_1}(p_1) \delta c^{\alpha_2}(p_2) \delta G_{\mu_3}^{\alpha_3}(p_3)} = g(2\pi)^4 \delta(p_1 + p_2 + p_3) p_1^{\mu_3} i f_{\alpha_2\alpha_3}^{\alpha_1} .$$

Again, the factor $(2\pi)^4 \delta(p_1 + p_2 + p_3) (-i f_{\alpha_1\alpha_2\alpha_3})$ is substituted by CalcHEP. Thus, this interaction may be implemented in the **Vertex** table as:

A1	A2	A3	A4	Factor	Lorentz part
G.C	G.c	G		-GG	p1.m3

where $G.C$ and $G.c$ are the Faddeev-Popov anti-ghost (\bar{c}) and ghost (c), respectively.

4-gluon interaction. In addition to the 3-gluon interaction of QCD, the Lagrangian, also, contains the following 4-gluon interaction:

$$S_{4G} = -\frac{g^2}{4} g^{\mu\mu'} g^{\nu\nu'} \delta_{\alpha\alpha'} \int f_{\beta\gamma}^\alpha G_\mu^\beta(x) G_\nu^\gamma(x) f_{\beta'\gamma'}^{\alpha'} G_{\mu'}^{\beta'}(x) G_{\nu'}^{\gamma'}(x) d^4x .$$

Fourier transformation and functional differentiation lead us to an expression which contains three different $SU(3)$ color structures:

$$\begin{aligned} \frac{\delta S_{4G}}{\delta G_{\mu_1}^{\alpha_1}(p_1) \delta G_{\mu_2}^{\alpha_2}(p_2) \delta G_{\mu_3}^{\alpha_3}(p_3) \delta G_{\mu_4}^{\alpha_4}(p_4)} &= -g^2 (2\pi)^4 \delta(p_1 + p_2 + p_3 + p_4) \delta_{\epsilon\epsilon'} \times \\ &\left(f_{\alpha_1\alpha_2}^\epsilon f_{\alpha_3\alpha_4}^{\epsilon'} (g^{\mu_1\mu_3} g^{\mu_2\mu_4} - g^{\mu_1\mu_4} g^{\mu_2\mu_3}) + f_{\alpha_1\alpha_3}^\epsilon f_{\alpha_2\alpha_4}^{\epsilon'} (g^{\mu_1\mu_2} g^{\mu_3\mu_4} - g^{\mu_1\mu_4} g^{\mu_2\mu_3}) \right. \\ &\left. + f_{\alpha_1\alpha_4}^\epsilon f_{\alpha_2\alpha_3}^{\epsilon'} (g^{\mu_1\mu_2} g^{\mu_3\mu_4} - g^{\mu_1\mu_3} g^{\mu_2\mu_4}) \right). \end{aligned} \quad (41)$$

The complicated color structure of this vertex cannot be directly written down in the CalcHEP **Vertex** table. To implement this vertex, we must use the following trick. We introduce the auxiliary tensor field $t_{\mu\nu}^\alpha(x)$ and the following Lagrangian for its interaction with the gluon field:

$$S_{aux} = \int \left(\frac{i g}{\sqrt{2}} f_{\beta\gamma}^\alpha t_\alpha^{\mu\nu}(x) G_\mu^\beta(x) G_\nu^\gamma(x) - \frac{1}{2} t_{\mu\nu}^\alpha(x) t_\alpha^{\mu\nu}(x) \right) d^4x.$$

It can be seen that functional integration over the auxiliary field $t_{\mu\nu}^\alpha(x)$ reproduces the 4-gluon interaction in the partition function:

$$e^{i S_{4G}(G)} = \int e^{i S_{aux}(G,t)} \prod_{x,\alpha,\mu,\nu} dt_{\mu\nu}^\alpha(x).$$

For each colored vector particle, CalcHEP automatically adds a tensor field with the same color to the internal list of quantum fields. The propagator for this field (13) corresponds to the Lagrangian $(-\frac{1}{2} t_{\mu\nu}^\alpha(x) t_\alpha^{\mu\nu}(x))$. Consequently, in order to realize the 4-gluon interaction we must introduce the following vertex for the interaction of the gluon with this tensor field:

$$\begin{aligned} S_{tGG} &= \frac{i g}{\sqrt{2}} \int f_{\beta\gamma}^\alpha t_\alpha^{\mu\nu}(x) G_\mu^\beta(x) G_\nu^\gamma(x) d^4x; \\ \frac{\delta S_{tGG}}{G_{\mu_1}^{\alpha_1}(p_1) G_{\mu_2}^{\alpha_2}(p_2) \delta t_{\mu_3\mu_3'}^{\alpha_3}(p_3)} &= (2\pi)^4 \delta(p_1 + p_2 + p_3) (-i f_{\alpha_1\alpha_2\alpha_3}) \\ &\times \frac{g}{\sqrt{2}} (g^{\mu_2\mu_3} g^{\mu_1\mu_3'} - g^{\mu_1\mu_3} g^{\mu_2\mu_3'}). \end{aligned}$$

In the CalcHEP **Vertex** table, this vertex looks like:

A1	A2	A3	A4	Factor	Lorentz part
G	G	G.t		GG/Sqrt2	m2.m3*m1.M3 -m1.m3*m2.M3

where $G.t$ is the CalcHEP notation for the auxiliary tensor field $t_{\mu\nu}^\alpha$ associated with the vector field G . Capital M denotes the second Lorentz index of the tensor field.

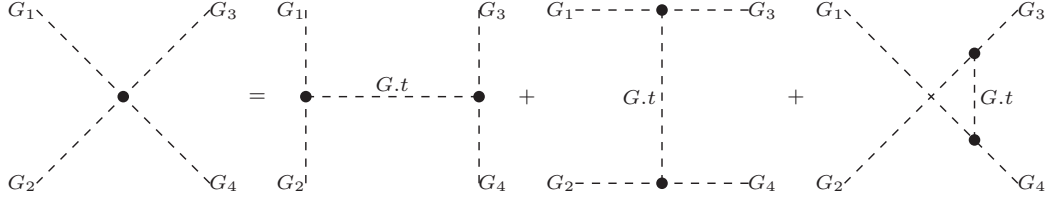


Figure 16: Splitting of four-gluon vertex

From the viewpoint of Feynman diagrams, this realization of the 4-gluon interaction means that instead of one 4-gluon vertex we substitute three sub-diagrams as shown in Fig.16. The contribution of each of these diagrams corresponds to one of the terms of expression (41).

E.2 Neutrino as a Majorana fermion

In the Standard Model, only the left handed component of the neutrino field takes part in the gauge interactions. In principle, the right handed component can be omitted. However, such a model can not describe neutrino oscillations, which have been detected at experiments. In the framework of the SM with right-handed neutrinos, the Yukawa interaction can give mass to the neutrino and the left- and right-handed components of the neutrino can be written as one Dirac 4-component spinor like the other SM fermions. However, in distinction to the other SM fermions, the right handed neutrino has zero $U(1)$ hypercharge, and is, in fact, a singlet under the SM gauge group. As a result, a mass term involving only the right-handed neutrinos is also allowed:

$$-\frac{1}{2}M\bar{\Psi}_\nu(1-\gamma_5)\Psi_\nu^c$$

The diagonalization of this combination of a Dirac mass term and a Majorana mass term splits the neutrino into two Majorana eigenstates with different masses. To see this explicitly, we express the 4-component Dirac neutrino field in terms of two Majorana fields ψ_l and ψ_r ,

$$\Psi_\nu = \frac{1}{2}(1-\gamma^5)\psi_l + \frac{1}{2}(1+\gamma^5)\psi_r,$$

In terms of the chiral fields, the mass terms are

$$-m_Y\bar{\Psi}_\nu\Psi_\nu - \frac{1}{2}M\bar{\Psi}_\nu(1-\gamma_5)\Psi_\nu^c = -\frac{1}{2}(m_Y\bar{\psi}_l\psi_r + m_Y\bar{\psi}_r\psi_l + 2M\bar{\psi}_r\psi_r)$$

were m_Y is the mass generated by the Yukawa interaction ¹¹. Choosing the

¹¹Note that $\bar{\psi}_r\gamma_5\psi_r = 0$

basis which diagonalize the mass matrix

$$\psi_l = -\cos(\alpha)\psi_1 + \sin(\alpha)\psi_2$$

$$\psi_r = \sin(\alpha)\psi_1 + \cos(\alpha)\psi_2$$

gives us the mixing angle α and the masses

$$\tan 2\alpha = 2m_Y/M$$

$$m_1 = -m_Y \tan \alpha = \frac{M}{2} - \sqrt{\frac{M^2}{4} + m_Y^2}$$

$$m_2 = m_Y / \tan \alpha = \frac{M}{2} + \sqrt{\frac{M^2}{4} + m_Y^2}$$

Negative masses are allowed and can be removed by the transformation

$$\psi_1 \rightarrow i\gamma_5\psi_1$$

This is the so-called *see-saw* mechanism which can potentially explain why the neutrino mass is so small. Using this mechanism, the Yukawa constant responsible for the value of m_Y can be of the same order as for the other fermions. If M is very large and the mixing angle α is very small, the light neutrino mass m_1 will be tiny and the heavy neutrino mass m_2 will be on the order of M . Roughly, we will have $\sqrt{m_1 m_2} \sim m_Y$.

To get the Feynman rules in terms of the Majorana fermions, we simply substitute the new fermion basis into the interaction Lagrangian. CalcHEP only supports 4-component fermion notation. In this notation, a Majorana fermion is given by:

Full name	A	A+	2*spin	mass	width	color	aux
neutrino	MN	MN	1	0	0	1	

In terms of the Dirac 4-component spinor field Ψ_ν a neutrino appears in the Standard Model Lagrangian in the following way¹²:

$$L_\nu = \frac{i}{2}(\bar{\Psi}_\nu \gamma_\mu \partial^\mu \Psi_\nu - (\partial^\mu \bar{\Psi}_\nu) \gamma_\mu \Psi_\nu) + \frac{e}{4 \sin \Theta_w \cos \Theta_w} Z_\mu \bar{\Psi}_\nu \gamma^\mu (1 - \gamma^5) \Psi_\nu + \frac{e}{2\sqrt{2} \sin \Theta_w} (W_\mu^- \bar{\Psi}_e \gamma^\mu (1 - \gamma^5) \Psi_\nu + W_\mu^+ \bar{\Psi}_\nu \gamma^\mu (1 - \gamma^5) \Psi_e) , \quad (42)$$

¹²where $Y = -1$, $\Psi_1 = \Psi_\nu$, $\Psi_2 = \Psi_e$, $g_2 = e/\sin \Theta_w$, $g_1 = e/\cos \Theta_w$.

where Ψ_e is the electron field. To rewrite this Lagrangian in terms of a Majorana neutrino let us perform the substitution

$$\Psi_\nu = \frac{1}{2}(1 - \gamma^5)\psi_l + \frac{1}{2}(1 + \gamma^5)\psi_r ,$$

where ψ_l and ψ_r are the Majorana fermions. Omitting the Lagrangian for ψ_r (there are no interactions for ψ_r) and applying the following identities for Majorana fermions

$$\frac{i}{4}(\bar{\psi}_l \gamma_\mu \gamma^5 \partial^\mu \psi_l - (\partial^\mu \bar{\psi}_l) \gamma_\mu \gamma^5 \psi_l) = \bar{\psi}_l \gamma^\mu \psi_l = 0$$

which can be obtained by means of Eq. (10), we get

$$\begin{aligned} L_\nu = & \frac{i}{4}(\bar{\psi}_l \gamma_\mu \partial^\mu \psi_l - (\partial^\mu \bar{\psi}_l) \gamma_\mu \psi_l) - \frac{e}{4 \sin \Theta_w \cos \Theta_w} Z_\mu \bar{\psi}_l \gamma^\mu \gamma^5 \psi_l \\ & + \frac{e}{2\sqrt{2} \sin \Theta_w} (W_\mu^- \bar{\Psi}_e \gamma^\mu (1 - \gamma^5) \psi_l + W_\mu^+ \bar{\psi}_l \gamma^\mu (1 - \gamma^5) \Psi_e) . \end{aligned}$$

The first term is the free Lagrangian for a massless Majorana fermion while the following terms define the interaction. Using the definition (8.4) we can write them in the CalcHEP **Vertex** :

A1	A2	A3	A4	Factor	Lorentz part
MN	MN	Z		-EE/(2*SW*CW)	G(m3)*G5
E	MN	W-		EE/(2*Sqrt2*SW)	G(m3)*(1-G5)
MN	e	W+		EE/(2*Sqrt2*SW)	G(m3)*(1-G5)

We note that there are two identical neutrino fields in the Lagrangian term which describes the interaction of neutrinos with a Z -boson. It leads to an additional factor of 2 and to the symmetry property of the corresponding vertex. One of the typical mistakes users make in the realization of this vertex is the erroneous introduction of a $G(m3)*(1-G5)$ term which breaks the symmetry property. Correct evaluation of the functional derivative (8.4) with the help of the identity (10) does not produce such a term.

E.3 Leptoquarks

In this section, we present an example of a Lagrangian which contains C-conjugated fermions. These terms appear in interactions which violate fermion number conservation. Let Ψ_e and Ψ_u be the fermion fields of the electron

and the u-quark, respectively and let these fields interact with a complex scalar leptoquark field F with Lagrangian

$$L = \lambda \bar{\Psi}_u^c (1 + \gamma^5) \Psi_e F + \lambda \bar{\Psi}_e (1 + \gamma^5) \Psi_u^c F^+$$

where $\bar{\Psi}_u^c = \Psi_u^T C$ and $\Psi_e^c = C \bar{\Psi}_e^T = C(\gamma^0)^T \Psi_e^*$.

In the basis used by CalcHEP, the charge conjugation operator is given by $C = -\gamma^0$. The Lagrangian can then be written in the form

$$\begin{aligned} L &= -\lambda \Psi_u^T \gamma^0 (1 + \gamma^5) \Psi_e F - \lambda \Psi_e^+ \gamma_0 (1 + \gamma^5) \gamma^0 (\gamma^0)^T \Psi_u^+ F^+ \\ &= -\lambda \Psi_u^T \gamma^0 (1 + \gamma^5) \Psi_e F + \lambda \Psi_u^+ \gamma^0 (1 + \gamma^5) \Psi_e^+ F^+ . \end{aligned}$$

Direct implementation of the definition (8.4) gives us the *Vertex* table

A1	A2	A3	A4	Factor	Lorentz part
u	e	F		-lambda	(1+G5)
E	U	F+		lambda	(1+G5)

By means of equation (10) we can rewrite this table in the equivalent form:

A1	A2	A3	A4	Factor	Lorentz part
e	u	F		-lambda	(1+G5)
U	E	F+		lambda	(1+G5)

F Color string basis.

CalcHEP performs averaging/summation over color states of incoming/outgoing quarks and gluons. But in order to describe the hadronization of outgoing particles one needs to specify the color states in more detail.

CalcHEP passes the problem of hadronization to other programs such as PYTHIA [54]. PYTHIA performs the hadronization in the framework of the color string model. According to this model, pairs of outgoing partons with opposite color are jointed into a colorless object, called strings. Partons are attached at the ends of the string and, usually, move in different directions. When the distance between the partons becomes large, the color string breaks creating two strings, each with smaller energy than the parent string. This process continues until the energy of each string is low enough to form a stable composite particle and it is then treated as a meson.

If we consider a QCD amplitude diagram without external gluons, we can use the rules from (37a) -(38a) to transform the amplitude's color diagram

to one where separated quark lines connect $q\bar{q}$ pairs. This corresponds to the color string picture described above and used by PYTHIA.

It should be noted that these color states are orthogonal only in the $N_c \rightarrow \infty$ limit. The orthogonality is required to treat the squared basis coefficients as mutually independent probabilities of producing the corresponding color strings. Thus, the color string model should be considered in the framework of the $1/N_c$ approximation. In the same approximation, one can consider the gluon color state as a $q\bar{q}$ state (38a). Therefore, the gluon is a particle where one color string is finished and another one begins.

During event sampling, CalcHEP generates a phase-space point according to the exact $N_c = 3$ matrix element. At the same time, it also calculates the leading coefficients of the $1/N_c$ expansion for the matrix element over the color flow basis. The generated phase space point is accompanied with a color flow with a probability proportional to the squared basis coefficient.

G Distribution functions and beam spectra

G.1 Backscattered photon spectrum

This function describes the spectrum of photons scattered backward from the interaction of laser light with the high energy electron beam:

$$f(x) = \begin{cases} 0, & \text{for } x > x_{max} \\ N(1 - x + 1/(1 - x)(1 - 4x/x_0(1 - x/(x_0(1 - x))))), & \text{for } 0 < x < x_{max} \end{cases}$$

where $x_0 = 4.82$, $x_{max} = x_0/(1 + x_0)$, N is a normalization factor.

The above spectrum corresponds to the special initial condition when unpolarized photons are created. See [78] for more details.

G.2 Weizsaecker-Williams approximation

Weizsaecker-Williams approximation is used to describe processes of electro-production in the case of small angle of charged particle scattering. In this case the virtual photon emitted by the scattering particle appears near to the mass shell (see Fig.17). It gives a possibility to reduce the process of electro-production to the photo-production one with an appropriate photon spectrum:

$$f(x) = (q^2 \alpha / (2\pi)) (\log((1 - x)/(x^2 \delta)) (1 + (1 - x)^2/x - 2(1 - x - \delta x^2)/x),$$

where α is the fine structure constant, q is a charge of incoming particle, m is its mass, $\delta = (m/Q_{max})^2$. Q_{max} sets out the region of photon virtuality

($P^2 > -Q_{max}^2$) which contributes to the process. It is assumed that region of large virtuality can be taken into account by direct calculation of electroproduction. As a rule this contribution is small enough.

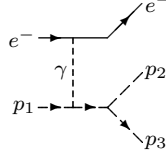


Figure 17: Example of process with the $1/t^2$ pole cancellation.

Parameters q , m , and Q_{max} are defined by the user. See [40] for the further explanations. In the case of CompHEP the Weizsaecker-Williams photon spectrum is available for charged leptons only.

G.3 ISR and Beamstrahlung

ISR (Initial State Radiation) is a process of photon radiation by the incoming electron due to its interaction with other collision particle. The resulting spectrum of electron has been calculated by Kuraev and Fadin [79]. In CompHEP we realize the similar expression by Jadach, Skrzypek, and Ward [80]:

$$F(x) = \frac{\exp(\beta(3/4 - Euler))\beta(1-x)^{\beta-1}}{((1+x^2) - \beta((1+3x^2)\ln(x)/2 + (1-x)^2)/2)/(2\Gamma(1+\beta))},$$

where

$$\begin{aligned} \alpha &= 1/137.0359895 \text{ is the fine structure constant;} \\ \beta &= \alpha(2\ln(SCALE/m) - 1)/\pi \\ m &= 0.00051099906 \text{ is the electron mass;} \\ Euler &= 0.5772156649 \text{ is the Euler constant;} \\ \Gamma() &\text{ is the gamma function;} \\ SCALE &\text{ is the energy scale of reaction.} \end{aligned}$$

In the Kuraev and Fadin article the parameter SCALE equals to the total energy of the process because they considered the process of direct e^+e^- annihilation. In order to apply this structure function to another processes we provide the user with a possibility to define this parameter.

Beamstrahlung is a process of energy loss by the incoming electron due to its interaction with the electron (positron) bunch moving in the opposite

direction. The key parameter of beamstrahlung is

$$\Upsilon = \frac{5 \alpha N E}{6 m_e^3 \sigma_z (\sigma_x + \sigma_y)},$$

where

N	is number particles in the bunch,
$\sigma_x, \sigma_y, \sigma_z$	are sizes of bunch,
E	is a center-of-mass momentum.

The effective energy spectrum of electrons we use approximated formulas of P. Chen [81]. Namely for $\Upsilon < 1$ we use formulas (17,18), and (22,23) otherwise. Picture (18) presents comparison of implemented approach with Monte Carlo simulation by Guinea Pig program for CLIC3000 design. Because singularity $(1 - x)^{\frac{2}{3}}$ is expected for beamstrahlung, we have defived spectra on this factor for proper comparison in $x \approx 1$ region. Convolution

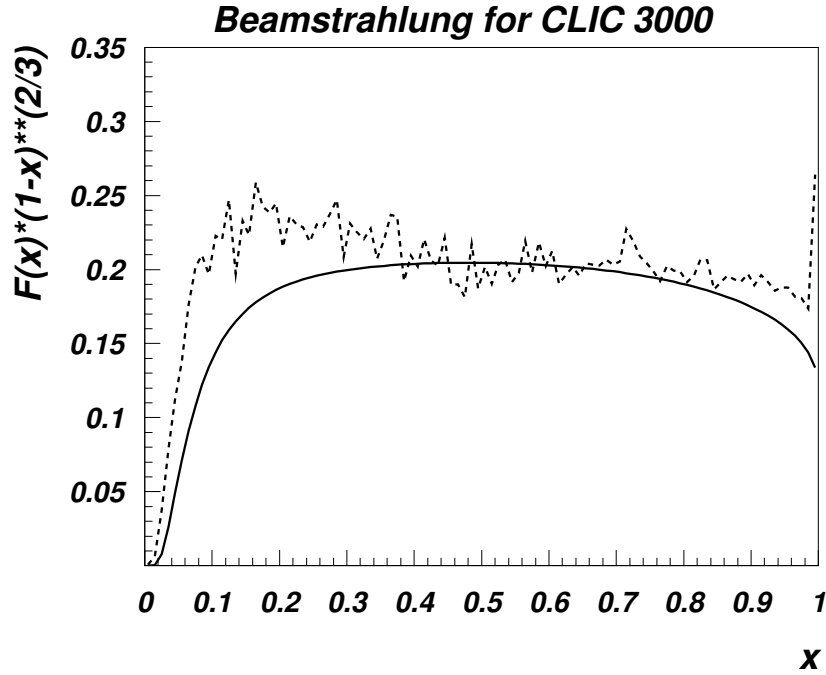


Figure 18: Comparison of P.Chen approximated formula (implemented in the CalcHEP, solid line) with Guinea Pig Monte Carlo for CLIC3000 (dashed line).

of beamstrahlung and ISR spectra is done automatically. ISR affect smooths

spectra and improve agreement of P.Chen formulas with Monte Carlo simulation. The Beamstrahlung spectrum cannot be integrated by the current CompHEP version because it contains a δ -function.

H PDT - Particle Distribution Tables in CalcHEP.

H.1 CTEQ and MRST parton distributions

Both CTEQ and MRST groups store information about parton distributions in two-dimensional tables and interpolate these tables. CalcHEP has its own file format for parton tables but uses interpolation procedures of CTEQ and MRST. Thus CalcHEP produces exactly the same results as original CTEQ/MRST functions. The information about interpolation procedure is stored in CalcHEP tables and is detected automatically.

Besides of parton distributions CalcHEP tables contain data for $\alpha_s(q)$ which correspond to the given parton set and this function is available to CalcHEP user. See section (5.4).

The files containing parton distributions must have the *"pdt"* extension. `n_calchep` searches such files in the directories "`$CALCHEP/pdTables`", `"../"`, and `"./"`. Usually the last two directories are the user's working directory and its sub-directory `results`.

`"$CALCHEP/pdTables"` contains the following parton sets: CTEQ6L, CTEQ6M [82] and `lo2002`, `mrst2002nlo`, `mrst2002nnlo` [83]

We pass to the user the routines which transform CTEQ and MRST data files to the CalcHEP format. By means of them the user can add other distribution to the list. The primary c-files are stored in the `$CALCHEP/pdTables` directory. In case of CTEQ the compilation instruction is

```
cc -o cteq2pdt cteq2pdt.c alpha.c -lm
```

The usage is

```
./cteq2pdt < cteq_file.tbl >calchep_file.pdt
```

for example

```
./cteq2pdt < cteq6m.tbl >cteq6m.pdt
```

The name of *pdt* file doesn't play a role. The `cteq2pdt` routine can be applied to any CTEQ4, CTEQ5, CTEQ6 file. It automatically detects version and α_s formula stored in the CTEQ file.

In the case of MRST file the corresponding compilation instruction is

```
cc -o mrst2pdt mrst2pdt.c alpha.c -lm
```

The usage is

```
./mrst2pdt name < mrst_file.dat >calchep_file.pdt
```

or

```
./mrst2pdt name nf order  $\alpha(MZ)$  < mrst_file.dat > calchep_file.pdt
```

For example

```
./mrst2pdt mrst2002nlo 5 nlo 0.1197 <mrst2002nlo.dat>mrstnlo.pdt
```

The number of parameters is increased in comparing with CTEQ case, because MRST tables don't contain the corresponding information. Note that *name* is the identifier of distribution that you will see in CalcHEP menu. If the last four parameters are not specified then α_s will not be included in the table. See MRST documentation to find the proper parameters¹³

For simple checks of *pdt* files one can use the *checkpdt* program. The source of this program is stored in \$CALCHEP/utile. Compilation instruction is

```
cc -o checkpdt checkpdt.c pdf.c -lm
```

The usage

```
./checkpdt file.pdt parton x q
```

where *parton* is a parton symbol: 'G' - for gluon, 'u', 'd', 's', 'c', 'b' - for quarks; 'U', 'D', 'S', 'C', 'B' - for anti-quarks. This program writes on the screen the corresponding parton density and $\alpha_s(q)$. See code checkpdt.c to create more extended test.

H.2 Format of parton distribution tables.

The structure of *pdt* file in CalcHEP is closed to the CTEQ one, but is more flexible and complete.

Generally distribution function depends on two arguments. They are the Feynman parameter X and the energy scale Q . X is unitless and runs in $[0,1]$ interval. $Q > 1\text{GeV}$ and traditionally is presented in GeV units. Thus we describe grids for X and Q variables and tables of parton distributions corresponding to the grids.

CalcHEP *pdf* file contains several items of information, Each item is started by a keyword. A keyword is started from the '#' symbol. Numerical data following a keyword must be separated by white-space characters. The appearance of new keywords designates the end of the previous item of information. The keywords are

- **#distribution**

After this word the title of distribution function is disposed. The title has been surrounded by the quotation marks (") symbols. It can contain space symbols inside. After that the incoming particle number and numbers for partons are disposed. All particles and partons are numerated according to the Monte Carlo numeration scheme [58]. Incoming

¹³nf=5 always, the order is included in the file name.

particle and its parton are separated by the "=>" string. Partons can be combined into groups by the the brackets (). For example¹⁴

```
#distribution "cteq6m(proton)" 2212=>(5 -5) (4 -4) (3 -3) -1 -2 21 2 1
```

All partons in one group have the same distribution. Position of a parton or of a group in the list correspond to number of table presented below. Say b and \bar{b} quark distributions will be disposed after the **#1-parton** keywords.

One file can contain several, usually two, **distribution** items. It allows to use one file both for the particle and for the anti-particle. For example, the "cteq6m.pdt" file contains also

```
#distribution "cteq6m(a-proton)" -2212=>(5 -5) (4 -4) (3 -3) 1 2 21 -2 -1
```

So **#7-parton** item describes u -quark in proton and \bar{u} in anti-proton.

- **#x-grid**
After this keyword the file has to contain a sequence of numbers which specify the X grid. The number of points has exceed 2. Numbers must increase from point to point and belong to interval $[0,1]$.
- **#q-grid**
Specifies the beginning of Q grid. The data must be positive and increase. This item is optional. Generally we can consider parton distributions which do not depend on Q .
- **#alpha**
designates beginning of data for $\alpha(Q)$ corresponding to Q -grid. The item is optional and can not precede the **q-grid** item.
- **#1-parton, #2-parton**
These keywords mark the beginnings of data for distribution functions. They can not precede the **x-grid** and **q-grid** items because the power of data for these items must be the production of the **x-grid** and **q-grid** powers. Date corresponding to i^{th} position of **x-grid** and j^{th} position of **q-grid** is disposed on the $(i + (j - 1)n_x)^{th}$ place, where n_x is the power of **x-grid**.

The reader of *pdt* file finishes its work when it has read the needed **parton** item. Thus all other items described below should preceed the **parton** one. These items are auxiliaries and if it needs can be specified separately before each **n-parton** item.

¹⁴1,2,3,4,5 are numbers of d,u,c,s,b quaks, 21 is the gluon number.

- **#x-min, #q-min**
specifies the minimum boundaries of 'x' and 'q' intervals where result of interpolation should be correct. The corresponding number must follow to the keyword. These keywords are optional. If one of them is absent the first point of the corresponding grid is user to define the limit. These limits do not influent on the work of the program. They are used to collect statistics of points out of limits. These statistics also count the number of points where Q exceeds the last point of q-grid.
- **#mass**
This keyword allows to introduce the mass of the composite particle. The corresponding numerical value must follow the keyword. Default value for mass parameter is 1 GeV.
- **#Interpolation**
It defines interpolation procedure. For current version the following interpolations are available: CTEQ4, CTEQ6, MRST2001¹⁵. CTEQ6 interpolation procedure depends on λ_{QCD} parameter which must be written after "CTEQ6".
- **#q-threshold**
Defines thresholds for c and b quarks. Used by MRST.

I Monte Carlo phase space integration

I.1 Parameterization of multi-particle phase space

I.1.1 Parameterization via decay scheme

The element of phase space volume for a n -particle state is equal to [58]

$$d\Gamma_n(q) = (2\pi)^4 \delta^4(q - p_1 - p_2 - p_3 - \dots - p_n) \prod_{i=1}^n \frac{\delta(p_i^2 - m_i^2)}{(2\pi)^3} d^4 p_i. \quad (43)$$

The same expression is valid for both the decay of unstable particle with momentum q and the interaction of two particles with momenta q_1 and q_2 such that $q_1 + q_2 = q$. For further discussion we need a designation for a phase space volume of some subset S of the full n -particle set. According to (43)

$$d\Gamma(q, S) = (2\pi)^4 \delta^4(q - \sum_{i \in S} p_i) \prod_{i \in S} \frac{\delta(p_i^2 - m_i^2)}{(2\pi)^3} d^4 p_i. \quad (44)$$

¹⁵CTEQ5 interpolation procedure is identical to CTEQ4

Let S_1 and S_2 be two disjoint particle subsets, then

$$\begin{aligned}
d\Gamma(q, S_1 \cup S_2) = & \int ds_1 ds_2 \left((2\pi)^4 \delta^4(q - q_1 - q_2) \frac{\delta(q_1^2 - s_1)}{(2\pi)^3} d^4 q_1 \frac{\delta(q_2^2 - s_2)}{(2\pi)^3} d^4 q_2 \right) \\
& \times \frac{d\Gamma(q, S_1)}{2\pi} \times \frac{d\Gamma(q, S_2)}{2\pi} .
\end{aligned} \tag{45}$$

The above formula expresses a multi-particle volume in terms of two-particle one, the volumes $d\Gamma(q_1, S_1)$ and $d\Gamma(q_2, S_2)$ with a reduced number of particles, and the virtual squared masses s_1, s_2 of clusters S_1, S_2 .

Recursive application of this formula allows one to express the multi-particle phase space in terms of two-particle phase space. In its turn the two-particle phase space is explicitly described by spherical angle Ω of motion of the first decaying particle in the rest frame of initial state [58].

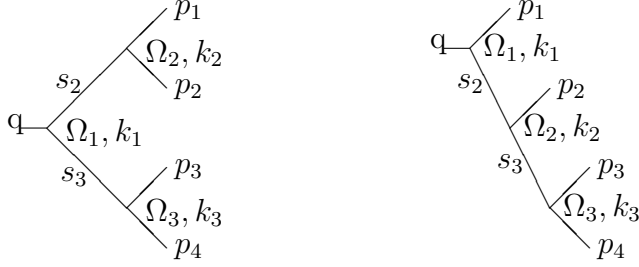
$$\frac{d\Gamma(q, [1, 2])}{2\pi} = \frac{k d\Omega}{4(2\pi)^3 \sqrt{q^2}} , \tag{46}$$

where k is the absolute value of three-dimensional momentum of outgoing particles in the rest frame. Thus, applying recursively (45) and (46) to (43) we obtain an explicit expression for the phase space volume in terms of the squared masses s_j of virtual clusters and the two-dimensional spherical angles Ω_j , where j is an ordinal number of decay:

$$d\Gamma_n(q) = \frac{k_1 d^2 \Omega_1}{4(2\pi)^2 \sqrt{q^2}} \prod_{j=2}^{n-1} \frac{k_j d^2 \Omega_j}{4(2\pi)^3 \sqrt{s_j}} \cdot \prod_{j=2}^{n-1} ds_j \tag{47}$$

Here k_j is a momentum of outgoing clusters produced by decay of the j^{th} cluster in its center-of-mass.

The expression (47) means some sequential 1->2 decay scheme which starts from incoming state and finishes with outgoing particles of the process. For example, the integration domain for s_j parameters depends on this scheme. Below we present two such schemes for a process with four outgoing particles:



In the case of CompHEP project such decay scheme is defined by the user via the ‘Kinematics’ menu (see Section 5.9).

I.1.2 Polar vectors

To complete phase space parameterization we must fix a polar coordinate system choosing the polar and the azimuthal angles for each of decays

$$d^2\Omega_j = d \cos \Theta_j d\Phi_j \quad (48)$$

We have an ambiguity in the choice of polar coordinate. Let us remind that our goal is not only parameterization of phase space but also regularization of the squared matrix element in the phase space manifold. The main idea of such regularization is a cancellation of integrand sharp peaks by the phase space measure. Originally the phase space measure (47) has no cancellation factors, but we can create them by means of a Jacobian of transformed variables. To get an appropriate Jacobian we need to have the initial phase space variables related to poles of the squared matrix element.

In their turn the poles of squared matrix element are caused by virtual particle propagators and generally have one of the forms (4), (5) or (6) (Section 5.10) depending on a squared sum of momenta. Variables s_j in (47) are also equal to squared sums of momenta. So, the parameterization (47) allows us to smooth some peaks of the matrix element.

It appears to be that the polar coordinates can be chosen in such a way that all $\cos\Theta_j$ have simple linear relations to the squared sums of momenta [84, 85]. The polar angle Θ_j can be unambiguously fixed by the *polar vector* $Pole_j$ whose space components in the rest frame of decay correspond to the $\Theta_j = 0$ direction. Let q_{j1} and q_{j2} be the momenta of the first and the second clusters produced by the j^{th} decay. Then

$$\begin{aligned} (Pole_j + q_{j1})^2 &= (Pole_j^0 + q_{j1}^0)^2 - |\overline{Pole_j}|^2 - |\bar{q}_{j1}|^2 - 2\cos\Theta_j |\overline{Pole_j}| |\bar{q}_{j1}| \\ (Pole_j + q_{j2})^2 &= (Pole_j^0 + q_{j2}^0)^2 - |\overline{Pole_j}|^2 - |\bar{q}_{j2}|^2 + 2\cos\Theta_j |\overline{Pole_j}| |\bar{q}_{j2}| \end{aligned}$$

Thus, in order to get $\cos\Theta_j$ related to a squared sum of some particle momenta we may construct the polar vector as a sum of particle momenta [84, 85].

For the non-contradictory construction we need to set the decays in some order with a natural requirement that the sub-decays of clusters produced by the j^{th} decay have the ordinal numbers larger than j . In giving such ordering we can construct a polar vector for each decay based on the incoming momenta and on those of particles produced by decays possessing smaller ordinal numbers.

The following statements can be proved. *In the framework of any ordered scheme of decays and for any sum P of particle momenta one can find the decay number j such that either $P^2 = s_j$ or P might be represented as $\text{Pole}_j + q_j$, where q_j is the momentum of one of the clusters in the j^{th} decay and Pole_j is a polar vector constructed according to the above rule.* In other words, any of poles (4), (5), (6) can be expressed either in terms of s_j parameters or in terms some of $\cos\Theta_j$ for an appropriate choice of the polar vector [84, 85].

In CompHEP the ordering is arranged automatically, so that all sub-decays of the first cluster have smaller numbers than those of the second cluster. Polar vectors are also constructed automatically according to the list of peaks prepared by the user.

I.1.3 Smoothing

The general idea of the integrand smoothing is trivial. Let us need to evaluate

$$\int_a^b F(x) dx \quad , \quad (49)$$

and let $F(x)$ have a peak like $f(x)$, where $f(x)$ is a simple symbolically integrable function in contrast to $F(x)$:

$$g(x) = \int_a^x f(x') dx' \quad . \quad (50)$$

Now we may represent the integral (49) as

$$\int_a^b F(x) dx = \int_0^{g(b)} dy \frac{F(g^{-1}(y))}{f(g^{-1}(y))}, \quad (51)$$

where $g^{-1}(y)$ is the inverse function for $g(x)$. The integrand is a smooth function now.

We face very often squared matrix elements which have several poles in one of variables. For example, the $\gamma \rightarrow b, \bar{b}$, $Z \rightarrow b, \bar{b}$ and $H \rightarrow b, \bar{b}$ virtual subprocesses may contribute just to the same amplitude. Although in

this case we can evaluate the integral function $g(x)$ symbolically, the inverse function $g^{-1}(y)$ can be computed only as a numerical solution of the corresponding equation. To bypass the calculation of inverse function CompHEP uses the multi-channel Monte Carlo (branching) method to smooth a sum of peaks.

The idea of the branching method is the following. Let $F(x)$ have two peaks, one is similar to $f_1(x)$ and another to $f_2(x)$. $f_1(x)$ and $f_2(x)$ are singular but elementary functions. Then, instead of one integration (49), we could perform two ones:

$$\int F(x)dx = \int \frac{F(x)f_1(x)}{f_1(x) + f_2(x)}dx + \int \frac{F(x)f_2(x)}{f_1(x) + f_2(x)}dx, \quad (52)$$

but now each integration has only a single peak! It is easy to extend this method for an arbitrary number of peaks.

The branching method was used in [86] to separate peaks which came from various diagrams. In that paper there was also proposed to use the expression (52) where $f_i(x)$ is replaced by $\alpha_i f_i(x)$ with a subsequent search for optimal coefficients α_i . CompHEP passes on this weight optimization to *Vegas*, combining two integrals in one *Vegas* hypercube.

As was mentioned above, CompHEP automatically searches for a polar vector for each angle integration in order to reach a linear relation between $\cos\Theta$ and one of the squared sum of momenta which is responsible for the peak. It could happen that various peaks need different polar vectors for the same decay. In this case CompHEP uses the branching method again, but now for the whole two-dimension sphere integration. In other words, we use the branching equation (52) where x is the two dimensional sphere angle [84,85].

I.2 Adaptive Monte Carlo integration package *Vegas*

This section contains a short description of the adaptive Monte Carlo program VEGAS. See for details [46,47].

The Monte Carlo method reduces a task of integral evaluation to the task of mean value calculation. Let $g(x)$ is a density function satisfying

$$\int g(x) dx = 1,$$

then

$$\int f(x) dx = \int f(x)/g(x) g(x) dx = \langle f/g \rangle = \lim_{N \rightarrow \infty} \sum (f(x_i)/g(x_i))/N,$$

where points x_i are sampled with the probability density $g(x) dx$.

The uncertainty σ_N of $\langle f/g \rangle$ estimation by N sample points is proportional to square root of function's variance divided over N :

$$\sigma_N = \sqrt{(\langle (f/g)^2 \rangle - \langle f/g \rangle^2)/N}.$$

VEGAS uses two techniques which allow to decrease the uncertainty of Monte Carlo calculation, namely the *importance sampling* and the *stratified sampling*.

I.2.1 Importance sampling

The idea of importance sampling technique is based on diminution of variance by a proper choice of the density function $g(x)$. The general solution of this problem could be in choosing

$$g(x) = |f(x)| / \int |f(x)| dx.$$

However this solution is useless because it returns us to the problem of evaluation of $f(x)$ integral and requires a generation of sampling points for complicated density function.

To bypass these problems VEGAS seeks this function in the factored form

$$g(x_1, x_2, \dots, x_n) = g_1(x_1) g_2(x_2) \dots g_n(x_n).$$

The optimal functions $g_i(x)$ could be easily evaluated in terms of $f(x)$ [46,47]. VEGAS is an adaptive program. For the first iteration it puts $g_i(x) = 1$. The information about $f(x)$ which VEGAS gets during the iteration is used to refine the density function. Generally VEGAS performs several iterations improving the density function after each of them.

The following parameters manage VEGAS work:

1. *Itmx* is a number of iterations;
2. *Ncall* is a number of integrand calls for one iteration.

I.2.2 Stratified sampling

The idea of stratified sampling method is to divide a volume of integration into a large number of sub-volumes and calculate integrals separately in each sub-volume. This method produces a smaller uncertainty comparing with the direct Monte Carlo method because here the uncertainty is caused only

by a function variance in the sub-volumes, while the integrand variation from one sub-volume to another does not contribute to the uncertainty.

The stratified sampling method is used to estimate the integral for any VEGAS iteration. The larger number Ncall is chosen, the smaller size of sub-volume becomes available and, consequently, the more successfully the stratified sampling works.

I.3 Generation of events

CalcHEP generates events according to the Von Neumann algorithm. See [58], p.202. Let the probability density $f(x)$ is smaller than an easily generated density $F(x)$ ¹⁶. Then one can generate x according to distribution $F(x)$ and accept this event with probability $f(x)/F(x)$. This procedure is repeated in cycle until the needed number of events is generated.

To build $F(x)$ CalcHEP divides the space volume on large number of sub-cubes and in each sub-cube sets $F(x)$ a constant which equals to $\max f(x)$. CalcHEP has two strategies of detecting the corresponding maxima. First one is a random search. The program generates random points in each sub-cube and tests $f(x)$ in these points. The second one is a search by the simplex method [47]. Here the program analyzes function in vertices of some simplex and tries to shift one vertex of this simplex to increase the function. This method leads to fast converges to local maximum, but one has to take into account that the distribution function can have several local maxima on the cub-cube boundary. Thus, preliminary random search needs to define a good start point for the search by the simplex method. The number of calls for random search and the number of steps for simplex search are defined by the user.

In general, the detected maxima are lower than the true ones. To satisfy the inequality

$$f(x) \leq F(x)$$

the function $F(x)$ based on the detected maxima may be multiplied by some factor, say 2. Of course, it decreases the efficiency of the generator just on the same factor. Nevertheless, in some sub-cubes where the variance of the function is large this factor may be not enough. If CalcHEP finds a point x where $f(x) > F(x)$ it accompany point with an integer weight w . This weight is the integral part of $f(x)/F(x)$ plus one with the probability equal to the fraction part of $f(x)/F(x)$. From view point of calculation of various distributions one event with integer weight w should be treated as

¹⁶We assume $f(x)$ and $F(x)$ are not normalized.

w independent events with identical parameters. But for the evaluation of statistical uncertainties a more careful treatment is needed.

I.4 Format of event file.

In general all needed comments are attached to the file. See below an example of header of such file. The numbers which accompany particle symbols are codes of Monte Carlo particle numbering scheme.

```
#CalcHEP version 2.3
#Type 2 -> 2
#Initial_state
  P1_3=1.000000E+02  P2_3=-1.000000E+02
  StrFun1="ISR(1.00S^.5 Beamstr.: OFF)" 11
  StrFun2="ISR(1.00S^.5 Beamstr.: OFF)" -11
#PROCESS 11(e) -11(E) -> 5(b) -5(B)
#MASSES 0.000000E+00 0.000000E+00 4.62000E+00 4.62000E+00
#Cross_section(Width) 1.700425E+01
#Number_of_events 1000
```

After that the table of events is written. The first column presents weights of events. Normal value for *weight* is 1. The reason of appearance $weight \neq 1$ is explained in the section (I.3). After that the columns which specify the momenta of particles are presented. The first line of the table contains titles for the columns. Say P3_2 means the second component of momentum of the third particle. The zero, energy, components are not presented because they can be calculated using the information about the particle masses. For incoming particles only the third momentum component is presented because other ones are zero.

After description of momentum the event record contains information about color flows. Each color flow is presented by a couple of particle numbers enclosed into brackets. It corresponds to propagation of color 3 from the first particle to the second one. See section (F) for algorithm of color flows generation.

Information about color flows is need to PYTHIA [54] to generate the correct fragmentation of colored quarks and gluons.

J Table of exit codes

For both s_ and n_ calchep	
0	normal termination
2,..,9..,16	the process was killed by the corresponding signal.
59?	error in edittab.c
65	error in writing on the disk
80	can not open X11 display
81	can not find "fixed" X11 font
100	LOCK file was not removed
101*	end of command sequence before end of program.
For s.calchep only.	
20	exit code for restart caused by user break or problem with memory.
22	exit code for restart to realize " Make n.calchep "
55	runtime error in colorf.c
60	error in lagrangian detected in the time of symbolic evaluation
62	error in model detected in the time of symbolic evaluation
70	not enough memory.
90?	debug exit for read_func.c
99	A needed directory, say "results", is absent and can't be created
102*	LOCK file in 'results' forbids to continue symbolical session
110*	error in input of process
111*	process of the type specified is absent
For n.calchep only.	
50	error in evaluation of QCD scale
51?	can not recognize position of singularity
52?	error in kinematics
53	runtime error caused by regularization(a pole in the phase space)
54	usrfun was not defined, but is used
121*	wrong name of variable.
122*	dependences can not be evaluated (NaN is produced)
123*	energy is too small
124*	can not evaluate cuts limlts
125*	wrong format of table of regularizations

The "*" symbol marks the exit codes which can be produced only in the *blind* mode. The "?" symbol marks exits included in the time of debugging. They are not expected and presented only for completeness.

References

- [1] A. Pukhov, “CalcHEP 2.3: MSSM, structure functions, event generation, batchs, and generation of matrix elements for other packages.” 2004.
- [2] H. Tanaka, T. Kaneko, and Y. Shimizu, “Numerical calculation of Feynman amplitudes for electroweak theories and an application to $e^+e^- \rightarrow W^+W^-$ gamma,” *Comput. Phys. Commun.* **64** (1991) 149–166.
- [3] F. Yuasa *et al.*, “Automatic computation of cross sections in HEP: Status of GRACE system,” *Prog. Theor. Phys. Suppl.* **138** (2000) 18–23, [arXiv:hep-ph/0007053](#).
- [4] G. Belanger *et al.*, “Automatic calculations in high energy physics and Grace at one-loop,” *Phys. Rept.* **430** (2006) 117–209, [arXiv:hep-ph/0308080](#).
- [5] H. Murayama, I. Watanabe, and K. Hagiwara, “HELAS: HELicity amplitude subroutines for Feynman diagram evaluations,”. KEK-91-11.
- [6] A. Pukhov *et al.*, “CompHEP: A package for evaluation of Feynman diagrams and integration over multi-particle phase space. User’s manual for version 33,” 1999.
- [7] **CompHEP** Collaboration, E. Boos *et al.*, “CompHEP 4.4: Automatic computations from Lagrangians to events,” *Nucl. Instrum. Meth.* **A534** (2004) 250–259, [arXiv:hep-ph/0403113](#).
- [8] J. Kublbeck, M. Bohm, and A. Denner, “FEYN ARTS: COMPUTER ALGEBRAIC GENERATION OF FEYNMAN GRAPHS AND AMPLITUDES,” *Comput. Phys. Commun.* **60** (1990) 165–180.
- [9] T. Hahn, “Generating Feynman diagrams and amplitudes with FeynArts 3,” *Comput. Phys. Commun.* **140** (2001) 418–431, [arXiv:hep-ph/0012260](#).
- [10] T. Hahn, “Automatic loop calculations with FeynArts, FormCalc, and LoopTools,” *Nucl. Phys. Proc. Suppl.* **89** (2000) 231–236, [arXiv:hep-ph/0005029](#).
- [11] F. Maltoni and T. Stelzer, “MadEvent: Automatic event generation with MadGraph,” *JHEP* **02** (2003) 027, [arXiv:hep-ph/0208156](#).

- [12] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer, and T. Stelzer, “MadGraph 5 : Going Beyond,” *JHEP* **06** (2011) 128, [arXiv:1106.0522 \[hep-ph\]](#).
- [13] A. Kanaki and C. G. Papadopoulos, “HELAC: A package to compute electroweak helicity amplitudes,” *Comput. Phys. Commun.* **132** (2000) 306–315, [arXiv:hep-ph/0002082](#).
- [14] C. G. Papadopoulos, “PHEGAS: A phase space generator for automatic cross- section computation,” *Comput. Phys. Commun.* **137** (2001) 247–254, [arXiv:hep-ph/0007335](#).
- [15] A. Cafarella, C. G. Papadopoulos, and M. Worek, “Helac-Phegas: a generator for all parton level processes,” *Comput. Phys. Commun.* **180** (2009) 1941–1955, [arXiv:0710.2427 \[hep-ph\]](#).
- [16] M. Moretti, T. Ohl, and J. Reuter, “O’Mega: An optimizing matrix element generator,” [arXiv:hep-ph/0102195](#).
- [17] W. Kilian, T. Ohl, and J. Reuter, “WHIZARD: Simulating Multi-Particle Processes at LHC and ILC,” 2007.
- [18] T. Gleisberg *et al.*, “SHERPA 1.alpha, a proof-of-concept version,” *JHEP* **02** (2004) 056, [arXiv:hep-ph/0311263](#).
- [19] T. Gleisberg *et al.*, “Event generation with SHERPA 1.1” *JHEP* **02** (2009) 007, [arXiv:0811.4622 \[hep-ph\]](#).
- [20] A. S. Belyaev, A. V. Gladyshev, and A. V. Semenov, “Minimal supersymmetric standard model within CompHEP software package,” [arXiv:hep-ph/9712303](#).
- [21] G. Belanger, F. Boudjema, A. Pukhov, and A. Semenov, “micromegas: A program for calculating the relic density in the mssm,” *Comput. Phys. Commun.* **149** (2002) 103–120, [hep-ph/0112278](#).
- [22] G. Belanger, F. Boudjema, C. Hugonie, A. Pukhov, and A. Semenov, “Relic density of dark matter in the NMSSM,” *JCAP* **0509** (2005) 001, [arXiv:hep-ph/0505142](#).
- [23] G. Belanger, F. Boudjema, S. Kraml, A. Pukhov, and A. Semenov, “Relic density of neutralino dark matter in the MSSM with CP violation,” *Phys. Rev.* **D73** (2006) 115007, [arXiv:hep-ph/0604150](#).

- [24] A. Belyaev, C.-R. Chen, K. Tobe, and C. P. Yuan, “Phenomenology of littlest Higgs model with T-parity: including effects of T-odd fermions,” *Phys. Rev.* **D74** (2006) 115020, [arXiv:hep-ph/0609179](#).
- [25] A. Belyaev, C. Leroy, R. Mehdiyev, and A. Pukhov, “Leptoquark single and pair production at LHC with CalcHEP/CompHEP in the complete model,” *JHEP* **09** (2005) 005, [arXiv:hep-ph/0502067](#).
- [26] A. Belyaev *et al.*, “Technicolor Walks at the LHC,” *Phys. Rev.* **D79** (2009) 035006, [arXiv:0809.0793 \[hep-ph\]](#).
- [27] H.-J. He *et al.*, “LHC Signatures of New Gauge Bosons in Minimal Higgsless Model,” *Phys. Rev.* **D78** (2008) 031701, [arXiv:0708.2588 \[hep-ph\]](#).
- [28] N. Christensen, P. de Aquino, C. Degrande, C. Duhr, B. Fuks, M. Herquet, F. Maltoni, and S. Schumann, “A Comprehensive approach to new physics simulations,” *Eur. Phys. J.* **C71** (2011) 1541, [arXiv:0906.2474 \[hep-ph\]](#).
- [29] A. Datta, K. Kong, and K. T. Matchev, “Minimal Universal Extra Dimensions in CalcHEP/CompHEP,” *New J. Phys.* **12** (2010) 075017, [arXiv:1002.4624 \[hep-ph\]](#).
- [30] B. A. Dobrescu, D. Hooper, K. Kong, and R. Mahbubani, “Spinless photon dark matter from two universal extra dimensions,” *JCAP* **0710** (2007) 012, [arXiv:0706.3409 \[hep-ph\]](#).
- [31] B. A. Dobrescu, K. Kong, and R. Mahbubani, “Leptons and photons at the LHC: Cascades through spinless adjoints,” *JHEP* **07** (2007) 006, [arXiv:hep-ph/0703231](#).
- [32] G. Burdman, B. A. Dobrescu, and E. Ponton, “Six-dimensional gauge theory on the chiral square,” *JHEP* **02** (2006) 033, [arXiv:hep-ph/0506334](#).
- [33] G. Belanger, M. Kakizaki, and A. Pukhov, “Dark matter in UED : the role of the second KK level,” [arXiv:1012.2577 \[hep-ph\]](#).
- [34] G. Belanger, A. Pukhov, and G. Servant, “Dirac Neutrino Dark Matter,” *JCAP* **0801** (2008) 009, [arXiv:0706.0526 \[hep-ph\]](#).
- [35] A. Semenov, “LanHEP - a package for the automatic generation of Feynman rules in field theory. Version 3.0” *Comput. Phys. Commun.* **180** (2009) 431–454, [arXiv:0805.0555 \[hep-ph\]](#).

- [36] N. D. Christensen and C. Duhr, “FeynRules - Feynman rules made easy,” *Comput. Phys. Commun.* **180** (2009) 1614–1641, [arXiv:0806.4194 \[hep-ph\]](#).
- [37] P. Z. Skands *et al.*, “SUSY Les Houches Accord: Interfacing SUSY Spectrum Calculators, Decay Packages, and Event Generators,” *JHEP* **07** (2004) 036, [arXiv:hep-ph/0311123](#).
- [38] B. C. Allanach *et al.*, “SUSY Les Houches Accord 2,” *Comp. Phys. Commun.* **180** (2009) 8–25, [arXiv:0801.0045 \[hep-ph\]](#).
- [39] M. R. Whalley, D. Bourilkov, and R. C. Group, “The Les Houches Accord PDFs (LHAPDF) and Lhaglu,” [arXiv:hep-ph/0508110](#).
- [40] V. M. Budnev, I. F. Ginzburg, G. V. Meledin, and V. G. Serbo, “The Two photon particle production mechanism. Physical problems. Applications. Equivalent photon approximation,” *Phys. Rept.* **15** (1975) 181–281.
- [41] J. Alwall *et al.*, “A standard format for Les Houches event files,” *Comput. Phys. Commun.* **176** (2007) 300–304, [arXiv:hep-ph/0609017](#).
- [42] G. Belanger, F. Boudjema, A. Pukhov, and A. Semenov, “micrOMEGAs2.0: A program to calculate the relic density of dark matter in a generic model,” *Comput. Phys. Commun.* **176** (2007) 367–382, [arXiv:hep-ph/0607059](#).
- [43] G. Belanger *et al.*, “Indirect search for dark matter with micrOMEGAs2.4” *Comput. Phys. Commun.* **182** (2011) 842–856, [arXiv:1004.1092 \[hep-ph\]](#).
- [44] J. A. M. Vermaseren, “Axodraw,” *Comput. Phys. Commun.* **83** (1994) 45–58.
- [45] A. C. Hearn, “REDUCE 2 USERS MANUAL,”. STAN-CS-70-181.
- [46] G. P. Lepage, “A New Algorithm for Adaptive Multidimensional Integration,” *J. Comput. Phys.* **27** (1978) 192.
- [47] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical recipes in C: the art of scientific computing*. Cambridge University Press, 1999.

- [48] J. Bjorken and S. Drell, *Relativistic quantum mechanics*. International series in pure and applied physics. McGraw-Hill, 1964.
- [49] U. Baur, J. A. M. Vermaseren, and D. Zeppenfeld, “Electroweak vector boson production in high-energy e^+e^- collisions,” *Nucl. Phys.* **B375** (1992) 3–44.
- [50] Y. Kurihara, D. Perret-Gallix, and Y. Shimizu, “ $e^+e^- \rightarrow e^-$ anti-electron-neutrino $\bar{\nu}_\mu$ anti- d from LEP to linear collider energies,” *Phys. Lett.* **B349** (1995) 367–374, [arXiv:hep-ph/9412215](#).
- [51] E. Boos, M. Dubinin, and L. Dudko, “Higgs boson production under the resonance threshold at LEP II,” *Int. J. Mod. Phys.* **A11** (1996) 5015–5026, [arXiv:hep-ph/9602220](#).
- [52] E. Byckling and K. Kajantie, *Particle kinematics*. Wiley, 1973.
- [53] V. A. Ilyin, D. N. Kovalenko, and A. E. Pukhov, “Recursive algorithm for the generation of relativistic kinematics for collisions and decays with regularizations of sharp peaks,” *Int. J. Mod. Phys.* **C7** (1996) 761, [arXiv:hep-ph/9612479](#).
- [54] T. Sjostrand, “High-energy physics event generation with PYTHIA 5.7 and JETSET 7.4” *Comput. Phys. Commun.* **82** (1994) 74–90.
- [55] J. Alwall *et al.*, “A Les Houches Interface for BSM Generators,” [arXiv:0712.3311 \[hep-ph\]](#).
- [56] S. Belov, L. Dudko, D. Kekelidze, and A. Sherstnev, “HepML, an XML-based format for describing simulated data in high energy physics,” *Comput. Phys. Commun.* **181** (2010) 1758–1768, [arXiv:1001.2576 \[hep-ph\]](#).
- [57] G. Belanger, N. D. Christensen, A. Pukhov, and A. Semenov, 2010.
- [58] “Particle data group.” <http://pdg.lbl.gov>.
- [59] B. C. Allanach, “Softsusy: A c++ program for calculating supersymmetric spectra,” *Comput. Phys. Commun.* **143** (2002) 305–331, [hep-ph/0104145](#).
- [60] W. Porod, “Sphenox, a program for calculating supersymmetric spectra, susy particle decays and susy particle production at e^+e^- colliders,” *Comput. Phys. Commun.* **153** (2003) 275–315, [hep-ph/0301101](#).

- [61] A. Djouadi, J.-L. Kneur, and G. Moultaka, “Suspect: A fortran code for the supersymmetric and higgs particle spectrum in the mssm.” hep-ph/0211331, 2002.
- [62] U. Ellwanger and C. Hugonie, “NMSPEC: A Fortran code for the sparticle and Higgs masses in the NMSSM with GUT scale boundary conditions,” *Comput. Phys. Commun.* **177** (2007) 399–407, [arXiv:hep-ph/0612134](#).
- [63] A. Djouadi, “The Anatomy of electro-weak symmetry breaking. I: The Higgs boson in the standard model,” *Phys.Rept.* **457** (2008) 1–216, [arXiv:hep-ph/0503172 \[hep-ph\]](#).
- [64] A. Djouadi, “The Anatomy of electro-weak symmetry breaking. II. The Higgs bosons in the minimal supersymmetric model,” *Phys.Rept.* **459** (2008) 1–241, [arXiv:hep-ph/0503173 \[hep-ph\]](#).
- [65] K. Chetyrkin, B. A. Kniehl, and M. Steinhauser, “Decoupling relations to $O(\alpha_s^3)$ and their connection to low-energy theorems,” *Nucl.Phys.* **B510** (1998) 61–87, [arXiv:hep-ph/9708255 \[hep-ph\]](#).
- [66] K. Chetyrkin, B. A. Kniehl, M. Steinhauser, and W. A. Bardeen, “Effective QCD interactions of CP odd Higgs bosons at three loops,” *Nucl.Phys.* **B535** (1998) 3–18, [arXiv:hep-ph/9807241 \[hep-ph\]](#).
- [67] M. Spira, A. Djouadi, D. Graudenz, and P. Zerwas, “Higgs boson production at the LHC,” *Nucl.Phys.* **B453** (1995) 17–82, [arXiv:hep-ph/9504378 \[hep-ph\]](#).
- [68] P. Baikov and K. Chetyrkin, “Higgs Decay into Hadrons to Order $\alpha_s^5(s)$,” *Phys.Rev.Lett.* **97** (2006) 061803, [arXiv:hep-ph/0604194 \[hep-ph\]](#).
- [69] S. Dawson, A. Djouadi, and M. Spira, “Qcd corrections to susy higgs production: The role of squark loops,” *Phys. Rev. Lett.* **77** (1996) 16–19, [hep-ph/9603423](#).
- [70] R. Barbieri, L. J. Hall, and V. S. Rychkov, “Improved naturalness with a heavy Higgs: An Alternative road to LHC physics,” *Phys.Rev.* **D74** (2006) 015007, [arXiv:hep-ph/0603188 \[hep-ph\]](#).
- [71] L. Lopez Honorez and C. E. Yaguna, “The inert doublet model of dark matter revisited,” *JHEP* **09** (2010) 046, [arXiv:1003.3125 \[hep-ph\]](#).

- [72] F. Staub, “SARAH,” *arXiv:0806.0538* [**hep-ph**] (2008) ,
[arXiv:0806.0538](#) [**hep-ph**].
- [73] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*. 1988.
- [74] B. Ruijl, T. Ueda, and J. Vermaseren, “FORM version 4.2”
[arXiv:1707.06453](#) [**hep-ph**].
- [75] L. Baulieu, “Perturbative Gauge Theories,” *Phys. Rept.* **129** (1985) 1.
- [76] T. P. Cheng and L. F. Li, “GAUGE THEORY OF ELEMENTARY PARTICLE PHYSICS,”. Oxford, Uk: Clarendon (1984) 536 P. (Oxford Science Publications).
- [77] A. P. Kryukov and A. Y. Rodionov, “COLOR: PROGRAM FOR CALCULATION OF GROUP WEIGHTS OF FEYNMAN DIAGRAMS IN NONABELIAN GAUGE THEORIES,” *Comput. Phys. Commun.* **48** (1988) 327–334.
- [78] I. F. Ginzburg, G. L. Kotkin, V. G. Serbo, and V. I. Telnov, “Colliding gamma e and gamma gamma Beams Based on the Single Pass Accelerators (of Vlepp Type),” *Nucl. Instr. Meth.* **205** (1983) 47–68.
- [79] E. A. Kuraev and V. S. Fadin, “On Radiative Corrections to e^+e^- Single Photon Annihilation at High-Energy,” *Sov. J. Nucl. Phys.* **41** (1985) 466–472. [*Yad.Fiz.*41:733-742,1985].
- [80] M. Skrzypek and S. Jadach, “Exact and approximate solutions for the electron nonsinglet structure function in QED,” *Z. Phys.* **C49** (1991) 577–584.
- [81] P. Chen, “Differential luminosity under multi - photon beamstrahlung,” *Phys. Rev.* **D46** (1992) 1186–1191.
- [82] J. Pumplin *et al.*, “New generation of parton distributions with uncertainties from global qcd analysis,” *JHEP* **07** (2002) 012, [hep-ph/0201195](#).
- [83] A. D. Martin, R. G. Roberts, W. J. Stirling, and R. S. Thorne, “Uncertainties of predictions from parton distributions. 1: Experimental errors,” *Eur. Phys. J.* **C28** (2003) 455–473, [arXiv:hep-ph/0211080](#).

- [84] V. A. Ilyin, D. N. Kovalenko, and A. E. Pukhov, “Recursive algorithm for the generation of relativistic kinematics for collisions and decays with regularizations of sharp peaks,” *Int. J. Mod. Phys. C* **7** (1996) 761, [arXiv:hep-ph/9612479](#).
- [85] D. N. Kovalenko and A. E. Pukhov, “Multiparticle phase space integration with arbitrary set of singularities in CompHEP,” *Nucl. Instrum. Meth. A* **389** (1997) 299–300.
- [86] F. A. Berends, R. Pittau, and R. Kleiss, “Excalibur: A Monte Carlo program to evaluate all four fermion processes at LEP-200 and beyond,” *Comput. Phys. Commun.* **85** (1995) 437–452, [arXiv:hep-ph/9409326](#).