R E C O L A 2

REcursive Computation of One-Loop Amplitudes 2 ☆

VERSION 2.0

Ansgar Denner[a], Jean-Nicolas Lang[a], Sandro Uccirati[b]

[a] *Universität Würzburg, Institut für Theoretische Physik und Astrophysik, D-97074 Würzburg, Germany*
[b] *Università di Torino e INFN, 10125 Torino, Italy*

Abstract

We present the Fortran95 program RECOLA2 for the perturbative computation of next-to-leading-order transition amplitudes in the Standard Model of particle physics and extended Higgs sectors. New theories are implemented via model files in the 't Hooft–Feynman gauge in the conventional formulation of quantum field theory and in the Background-Field method. The present version includes model files for Two-Higgs-Doublet Model and the Higgs-Singlet Extension of the Standard Model. We support standard renormalization schemes for the Standard Model as well as many commonly used renormalization schemes in extended Higgs sectors. Within these models the computation of next-to-leading-order polarized amplitudes and squared amplitudes, optionally summed over spin and colour, is fully automated for any process. RECOLA2 allows the computation of colour- and spin-correlated leading-order squared amplitudes that are needed in the dipole subtraction formalism. RECOLA2 is publicly available for download at http://recola.hepforge.org.

Keywords: NLO computations; one-loop amplitudes; Beyond Standard Model; higher orders; theories beyond the Standard Model

---

☆ The program is available from http://recola.hepforge.org.

Email addresses: ansgar.denner@physik.uni-wuerzburg.de (Ansgar Denner), jlang@physik.uni-wuerzburg.de (Jean-Nicolas Lang), uccirati@to.infn.it (Sandro Uccirati)

September 28, 2017

Contents

1. Introduction

In the era after the Higgs-boson discovery the focus in elementary particle physics is on the precise validation of the Standard Model (SM) and the search for possible extensions thereof a.k.a. Beyond Standard Model (BSM) theories. Nowadays we are able to perform precision predictions in the SM for a vast number of observables as a result of the automation of one-loop QCD [1–9] and electroweak (EW) [1, 10–14] corrections. In the future, amplitude providers need to be able to calculate one-loop QCD and EW corrections in general weakly interacting theories. However, the automation of one-loop EW corrections for BSM theories is more involved as several intermediate steps are required. The first step consists in the definition of new models in terms of a Lagrangian and a subsequent determination of the Feynman rules. This can be done by means of FeynRules [15, 16] and Sarah [17]. Then, the renormalization needs to be addressed which raises many questions for BSM theories as parameters cannot be necessarily linked to physical observables. Thus, many different renormalization schemes need to be investigated which in turn requires a systematic and flexible approach. Steps towards the automation of the renormalization for BSM theories have been undertaken in the FeynRules/FeynArts approach in Ref. [18]. Finally, the renormalized model file needs to be interfaced to a generic one-loop matrix-element generator. In Ref. [19] we proposed a complementary strategy to Ref. [18] combining the second and third step which requires a small set of external tools (FORM [20, 21] and REPT1L [19]). Our approach makes use of tree-level Universal FeynRules Output (UFO) model files [22] and results in renormalized one-loop model files for Recola2, a generalized version of Recola, allowing anyone to compute any process in the underlying theory at the one-loop level, with the only restriction being available memory and CPU power. Much effort has been spent on the validation of our system, and for this purpose the alternative formulation of quantum field theory (QFT) in the Background-Field method (BFM) has been implemented. In this report we describe the Recola2 library and the Recola2 model files for the computation of tree and one-loop amplitudes in the SM, the Two-Higgs-Doublet Model (2HDM) and the Higgs-Singlet Extension of the SM (HSESM).

This article is organized as follows: In Section 2 we summarize the new features of Recola2 compared to the prior version Recola [14]. In Section 3 the installation instructions for the Recola2 library and the model files are given. In Section 4 we describe the usage of Recola2 and comment on all

new subroutines related to models with extended Higgs sectors that can be called by the user. Finally, we conclude in Section 5 and list validation efforts in Appendix A.

2. New features in RECOLA2

RECOLA2 is an upgraded version of the FORTRAN95 code RECOLA [14] for the computation of tree-level and one-loop scattering amplitudes for general QFT, based on recursion relations [10]. At tree-level the algorithm computes amplitudes using Dyson–Schwinger equations [23–25]. At one-loop order the recursion relies on the decomposition of one-loop amplitudes in terms of tensor integrals, computed by means of the COLLIER library [26], and tensor coefficients computed by RECOLA2, within the framework of dimensional regularization. The extension to BSM concerns, in particular, the computation of tensor coefficients as they are process- and theory-dependent. The RECOLA2 library can generate arbitrary processes in BSM theories and allows for the computation of tensor coefficients involving new structures compared to the usual formulation of the SM.

In Ref. [19] we have presented our algorithm for a fully automated renormalization and computation of one-loop amplitudes. The intermediate results of this approach are RECOLA2-specific renormalized model files which are derived from nothing but tree-level UFO format [22] by means of the tool REPT1L. For the 2HDM and HSESM we specify in Section 2.1.1 and Section 2.1.2 the modified Higgs potentials and Yukawa sector, respectively. The tree-level UFO model files have been derived using FEYNRULES [15, 16]. With extended Higgs sectors the parameter space grows and requires renormalization of additional parameters. In Section 2.1.3 we list all implemented renormalization schemes available in the model files. In Section 2.2 we give details on the implementation of dimensional regularization and specify conventions used in $\overline{\text{MS}}$ renormalization schemes. In Section 2.3 we comment on the formulation of model files in the BFM and, finally, in Section 2.4 we list our conventions for the new fields used to define processes.

2.1. Extended Higgs sectors and their renormalization

2.1.1. Scalar potentials

Currently we support the 2HDM and HSESM as presented in Ref. [19] and summarized in the following. The models with extended Higgs sector are distinguished from the SM by modified scalar potentials and Yukawa

couplings. We derived the models for CP-conserving Z_2-symmetric renormalizable potentials. Under these constraints the most general scalar potential in the 2HDM reads [27]

$$\begin{aligned} V_{\text{THDM}} = {} & m_1^2 \Phi_1^\dagger \Phi_1 + m_2^2 \Phi_2^\dagger \Phi_2 - m_{12}^2 \left(\Phi_1^\dagger \Phi_2 + \Phi_2^\dagger \Phi_1 \right) \\ & + \frac{\lambda_1}{2} \left(\Phi_1^\dagger \Phi_1 \right)^2 + \frac{\lambda_2}{2} \left(\Phi_2^\dagger \Phi_2 \right)^2 + \lambda_3 \left(\Phi_1^\dagger \Phi_1 \right) \left(\Phi_2^\dagger \Phi_2 \right) \\ & + \lambda_4 \left(\Phi_1^\dagger \Phi_2 \right) \left(\Phi_2^\dagger \Phi_1 \right) + \frac{\lambda_5}{2} \left[\left(\Phi_1^\dagger \Phi_2 \right)^2 + \left(\Phi_2^\dagger \Phi_1 \right)^2 \right], \end{aligned} \tag{1}$$

with Φ_1, Φ_2 being Higgs doublets. The five couplings $\lambda_1 \dots \lambda_5$ and the two mass parameters m_1^2 and m_2^2 are chosen to be real. Further, we allow for soft-breaking of the Z_2 symmetry which is parametrized by the real parameter m_{12}^2.

Before spontaneous symmetry breaking (SSB) the parameters are expressed in the symmetric, defining basis and by the EW gauge couplings g and g' in the gauge eigenbasis. RECOLA2 operates in the physical basis expressed by masses, mixing angles and electromagnetic coupling. For the 2HDM this results in the following identification of physical parameters:

basis	V_{2HDM}	$\mathcal{L}_{\text{Gauge}}$
before SSB	$m_1, m_2, m_{12}, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$	g, g'
after SSB	$M_{H_\text{l}}, M_{H_\text{h}}, M_{H_\text{a}}, M_{H^\pm}, c_{\alpha\beta}, t_\beta, M_{\text{sb}}, M_\text{W}$	e, M_Z

The masses are uniquely defined as the eigenvalues of the canonically normalized mass matrices. Here, M_{H_l} and M_{H_h} denote the light and heavy neutral Higgs-boson masses, respectively, M_{H_a} denotes the pseudo-scalar Higgs-boson mass and $M_{H^\pm}$ the charged Higgs-boson mass. Besides being positive definite the neutral ones are constraint to $M_{H_\text{l}} < M_{H_\text{h}}$. The angles α and β are introduced to identify mass eigenstates and Goldstone-boson degrees of freedom. We follow the conventions in Ref. [27] where the dependence on the angles is parametrized as

$$\alpha,\ \beta \quad \to \quad c_{\alpha\beta} := \cos(\alpha - \beta),\ t_\beta := \tan\beta, \tag{2}$$

which is a natural choice for studying aligned scenarios. Here, t_β is defined as $t_\beta = \text{vev}_2/\text{vev}_1$, i.e. the ratio of the vevs associated to Φ_2 and Φ_1 [19, 28]. Below we give a selection of Feynman rules which can be used to fix the conventions. The angle α is defined in the window $[-\pi/2, \pi/2]$, whereas

the angle β is defined in the window $[0, \pi/2]$. This implies $c_\alpha := \cos\alpha = \sqrt{1-\sin^2\alpha}$, $c_\beta := \cos\beta = \sqrt{1-\sin^2\beta}$ and $t_\beta > 0$. Finally, we define the soft-breaking scale $M_{\rm sb}$ as

$$M_{\rm sb}^2 = \frac{m_{12}^2}{c_\beta s_\beta}. \tag{3}$$

We support two alternative sets of input parameters for mixing angles, namely:

parameter choice	domain
$c_{\alpha\beta}$	$[-1,1]$
t_β	$(0,\infty)$
s_α	$[-1,1]$
c_β	$(0,1)$

For the HSESM the most general CP-conserving Z_2-symmetric renormalizable scalar potential reads

$$V_{\rm HSESM} = m_1^2\Phi^\dagger\Phi + m_2^2 S^2 + \frac{\lambda_1}{2}\left(\Phi^\dagger\Phi\right)^2 + \frac{\lambda_2}{2}S^4 + \lambda_3\Phi^\dagger\Phi S^2, \tag{4}$$

with Φ being a Higgs doublet and S being a singlet field, and all parameters are real. We choose the following set of physical parameters:

basis	$V_{\rm HSESM}$	$\mathcal{L}_{\rm Gauge}$
before SSB	$m_1, m_2, \lambda_1, \lambda_2, \lambda_3$	g, g'
after SSB	$M_{H_1}, M_{H_{\rm h}}, s_\alpha, t_\beta, M_{\rm W}$	$e, M_{\rm Z}$

The angle α is defined in the same way as in the 2HDM and t_β is defined as $t_\beta = {\rm vev}_s/{\rm vev}$, i.e. the ratio of the vevs associated to S and Φ [19]. Again, RECOLA2 operates in the physical basis expressed by masses, mixing angles and electromagnetic coupling.

For comparison of phase conventions we list key couplings of type VVS ($g^{\mu\nu}$ omitted) and SSS in Table 1. The SM limit is reached for $s_{\alpha\beta} = -1$ and $s_\alpha = -1$ in the 2HDM and HSESM, respectively.

2.1.2. Yukawa sector

While for the HSESM the fermionic sector is the same as in the SM, the 2HDM allows for a richer structure with both doublet fields Φ_1 and Φ_2 coupling to fermions. Imposing a diagonal CKM matrix and the Z_2 symmetry, which is essential for suppressing flavour-changing neutral currents at

model	vertex	coupling
2HDM	$\mathrm{ZZ}H_\mathrm{l}$	$-\mathrm{i}s_{\alpha\beta}\frac{eM_\mathrm{Z}}{c_\mathrm{w}s_\mathrm{w}}$
	$\mathrm{ZZ}H_\mathrm{h}$	$+\mathrm{i}c_{\alpha\beta}\frac{eM_\mathrm{Z}}{c_\mathrm{w}s_\mathrm{w}}$
	$H_\mathrm{a}H_\mathrm{a}H_\mathrm{l}$	$+\mathrm{i}\frac{e}{2M_\mathrm{W}s_\mathrm{w}}\left(\left(2\left(M_{H_\mathrm{a}}^2-M_\mathrm{sb}^2\right)+M_{H_\mathrm{l}}^2\right)s_{\alpha\beta}-\left(M_{H_\mathrm{l}}^2-M_\mathrm{sb}^2\right)c_{\alpha\beta}\frac{1-t_\beta^2}{t_\beta}\right)$
	$H_\mathrm{a}H_\mathrm{a}H_\mathrm{h}$	$-\mathrm{i}\frac{e}{2M_\mathrm{W}s_\mathrm{w}}\left(\left(2\left(M_{H_\mathrm{a}}^2-M_\mathrm{sb}^2\right)+M_{H_\mathrm{h}}^2\right)c_{\alpha\beta}+\left(M_{H_\mathrm{h}}^2-M_\mathrm{sb}^2\right)s_{\alpha\beta}\frac{1-t_\beta^2}{t_\beta}\right)$
HSESM	$\mathrm{ZZ}H_\mathrm{l}$	$-\mathrm{i}s_{\alpha}\frac{eM_\mathrm{Z}}{c_\mathrm{w}s_\mathrm{w}}$
	$\mathrm{ZZ}H_\mathrm{h}$	$+\mathrm{i}c_{\alpha}\frac{eM_\mathrm{Z}}{c_\mathrm{w}s_\mathrm{w}}$
	$H_\mathrm{l}H_\mathrm{l}H_\mathrm{h}$	$-\mathrm{i}c_\alpha s_\alpha\frac{e}{2M_\mathrm{W}s_\mathrm{w}t_\beta}\left(M_{H_\mathrm{h}}^2+2M_{H_\mathrm{l}}^2\right)\left(c_\alpha+s_\alpha t_\beta\right)$
	$H_\mathrm{l}H_\mathrm{h}H_\mathrm{h}$	$-\mathrm{i}c_\alpha s_\alpha\frac{e}{2M_\mathrm{W}s_\mathrm{w}t_\beta}\left(M_{H_\mathrm{l}}^2+2M_{H_\mathrm{h}}^2\right)\left(s_\alpha-c_\alpha t_\beta\right)$

Table 1: Some key couplings in the 2HDM and HSESM.

tree-level, leads to the natural flavour-conserving models. The full Yukawa Lagrangian with all possible types reads

$$\begin{aligned}\mathcal{L}_\mathrm{Y} = &-\Gamma_\mathrm{d}\overline{Q}_\mathrm{L}\left(h_{1,\mathrm{d}}\Phi_1+h_{2,\mathrm{d}}\Phi_2\right)d_\mathrm{R}\\ &-\Gamma_\mathrm{u}\overline{Q}_\mathrm{L}\left(h_{1,\mathrm{u}}\tilde{\Phi}_1+h_{2,\mathrm{u}}\tilde{\Phi}_2\right)u_\mathrm{R}\\ &-\Gamma_\mathrm{l}\overline{L}_\mathrm{L}\left(h_{1,\mathrm{l}}\Phi_1+h_{2,\mathrm{l}}\Phi_2\right)l_\mathrm{R}+\mathrm{h.c.},\end{aligned} \tag{5}$$

with $\tilde{\Phi}_i$ being the charge conjugation of Φ_i, and Γ_d, Γ_u, Γ_l generically denote the up-type quark, down-type quark and lepton Yukawa couplings. The $\overline{Q}_\mathrm{L}$, $\overline{L}_\mathrm{L}$ and d_R, u_R, l_R denote the SM fermion doublet and singlet fields, respectively. The parameters $h_{i,F}$ trigger the desired Yukawa type as follows:

type	$h_{1,\mathrm{d}}$	$h_{2,\mathrm{d}}$	$h_{1,\mathrm{u}}$	$h_{2,\mathrm{u}}$	$h_{1,\mathrm{l}}$	$h_{2,\mathrm{l}}$
I	0	1	0	1	0	1
II	1	0	0	1	1	0
X	0	1	0	1	1	0
Y	1	0	0	1	0	1

In this convention, the Yukawa couplings are generically given by

vertex	coupling
$H_\mathrm{l}FF$	$-\mathrm{i}\frac{m_F e}{2M_\mathrm{W}s_\mathrm{w}t_\beta}\left(h_{2,F}\left(c_{\alpha\beta}-s_{\alpha\beta}t_\beta\right)-h_{1,F}t_\beta\left(s_{\alpha\beta}+c_{\alpha\beta}t_\beta\right)\right)$
$H_\mathrm{h}FF$	$-\mathrm{i}\frac{m_F e}{2M_\mathrm{W}s_\mathrm{w}t_\beta}\left(h_{2,F}\left(s_{\alpha\beta}+c_{\alpha\beta}t_\beta\right)+h_{1,F}t_\beta\left(c_{\alpha\beta}-s_{\alpha\beta}t_\beta\right)\right)$

with $h_{i,F}$ representing either $h_{i,\mathrm{d}}$, $h_{i,\mathrm{u}}$ or $h_{i,\mathrm{l}}$ depending on the fermion type F.

Note that the user can select the Yukawa type directly via `set_Z2_thdm_yukawa_type_rcl` (4.1.4), without having to worry about the values of $h_{i,F}$.

2.1.3. Renormalization schemes

The renormalization of the SM gauge couplings is performed as explained in Ref. [19]. We support the renormalization of α in the Thomson limit (α_0) or on the Z-pole (α_Z). Furthermore, α can be renormalized in the G_F scheme, neglecting the muon mass in vertex and box contributions.

In the extended Higgs sectors the renormalization of mixing angles and new couplings needs to be addressed. We support various renormalization schemes inspired by Refs. [28–36]. For all model files, except for the SM ones,[1] the tadpoles are by default renormalized in the *FJ Tadpole Scheme* [28, 35, 38] which, roughly speaking, affects all parameters depending on the vev. In particular, all counterterms to physical parameters are gauge-parameter independent. Nevertheless we support other tadpole-counterterm schemes via appropriate shifts with respect to counterterms defined in the *FJ Tadpole Scheme*. Note that the distinction of different tadpole-counterterm schemes is only relevant for the $\overline{\mathrm{MS}}$ renormalization of α, β and M_sb, but not for on-shell schemes or the $\overline{\mathrm{MS}}$ renormalization of parameters of the defining basis, i.e. λ_i, μ_i, because then the tadpoles necessarily drop out.[2] In summary, we distinguish between the following tadpole-counterterm schemes present in the literature:

FJTS: In the **F**leischer-**J**egerlehner **T**adpole counterterm **S**cheme tadpole counterterms are introduced via field redefinitions. See Appendix A in Ref. [38].

MDTS: In the **M**ass-**D**iagonal **T**adpole counterterm **S**cheme no tadpole counterterms emerge for two-point functions of physical fields. See the Feynman rules in Ref. [37].

[1] For the SM model files we renormalize the tadpoles as done in Ref. [37]. On request we can provide the model files in different tadpole schemes.

[2] There are subtleties for on-shell schemes defined in a particular gauge. See Appendix C in Ref. [19] for a discussion and simple solution of this issue.

MTS: In the **M**inimal **T**adpole counterterm **S**cheme the translation to the physical basis is done in such a way that the tadpole counterterms only appear in the quadratic terms of the Higgs potential. See Eq. (6) in Ref. [39].

The masses and fields of new (scalar) fields are renormalized in the on-shell scheme in the same way as in the SM. The only remaining parameters requiring renormalization are mixing angles and the soft-breaking scale $M_{\rm sb}$ in the 2HDM. Since α, β and $M_{\rm sb}$ are considered as independent parameters (or are related to direct derivatives thereof, e.g. $c_{\alpha\beta}, t_\beta, s_\alpha, \ldots$) we formulate all renormalization schemes directly for the corresponding counterterms $\delta\alpha$, $\delta\beta$ (δt_β) and $\delta M^2_{\rm sb}$.

- $\overline{\rm MS}$ schemes [28, 32, 33, 36]:
  We support standard $\overline{\rm MS}$ schemes. In each of these schemes terms proportional to $\Delta_{\rm UV}$ are subtracted as explained in Section 2.2. The UV finiteness of matrix elements can be tested by varying the scale $\mu_{\rm UV}$. The actual scale dependence of the scheme can be probed by varying the scale $\mu_{\rm MS}$. The counterterms can be derived from suited vertices of the theory, or from the pole part of the off-diagonal field renormalization constants.

  $\delta\alpha^{\overline{\rm MS}}$**:** In our conventions for the 2HDM and HSESM (see also Ref. [19]) we get for α in the FJTS:

$$\delta\alpha^{\overline{\rm MS}}_{\rm FJTS} := \delta\alpha^{\overline{\rm MS}} = \frac{\delta Z^{\overline{\rm MS}}_{H_{\rm h}H_{\rm l}} - \delta Z^{\overline{\rm MS}}_{H_{\rm l}H_{\rm h}}}{4}, \tag{6}$$

  which can be translated to other tadpole schemes as follows

$$\delta\alpha^{\overline{\rm MS}}_{\rm MDTS} := \delta\alpha^{\overline{\rm MS}} + \frac{t^{\rm fin}_{H_{\rm h}H_{\rm l}}}{M^2_{H_{\rm h}} - M^2_{H_{\rm l}}}, \tag{7}$$

$$\delta\alpha^{\overline{\rm MS}}_{\rm MTS} := \delta\alpha^{\overline{\rm MS}} + \frac{t^{\rm fin}_{H_{\rm h}H_{\rm l}} - t^{\rm fin}_{H_{\rm h}H_{\rm l},{\rm MTS}}}{M^2_{H_{\rm h}} - M^2_{H_{\rm l}}}, \tag{8}$$

  with $t_{H_{\rm h}H_{\rm l}}$ and $t_{H_{\rm h}H_{\rm l},{\rm MTS}}$ being the tadpole counterterms to the neutral mixing energy in the FJTS and MTS tadpole counterterm schemes, respectively.

$\delta\lambda_3^{\overline{\rm MS}}$, $\delta\lambda_{345}^{\overline{\rm MS}}$**:** Instead of α being renormalized $\overline{\rm MS}$, the user can choose between λ_3 and λ_{345}. In the 2HDM this is equivalent to define the counterterm of α in the following ways:[3]

$$\begin{aligned}
\delta\alpha^{\delta\lambda_{345}^{\overline{\rm MS}}} &= \delta\alpha^{\overline{\rm MS}} - \frac{c_\alpha s_\alpha}{c_\alpha^2 - s_\alpha^2} 2\delta Z_{\rm e}^{\rm fin} + \frac{c_\beta^2 - s_\beta^2}{c_\alpha^2 - s_\alpha^2}\frac{c_\alpha s_\alpha}{c_\beta s_\beta}\delta\beta^{\rm fin} \\
&- \frac{c_\alpha s_\alpha}{c_\alpha^2 - s_\alpha^2}\left[\frac{c_{\rm w}^2 - s_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm W}^{2,\rm fin}}{M_{\rm W}^2} - \frac{c_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm Z}^{2,\rm fin}}{M_{\rm Z}^2} + \frac{\delta m_{H_{\rm h}}^{2,\rm fin} - \delta m_{H_{\rm l}}^{2,\rm fin}}{M_{H_{\rm h}}^2 - M_{H_{\rm l}}^2}\right] \\
&- \frac{c_\beta s_\beta}{c_\alpha^2 - s_\alpha^2}\left[\frac{1}{M_{H_{\rm h}}^2 - M_{H_{\rm l}}^2}\Big[\delta M_{\rm sb}^{2,\rm fin}\right. \\
&\left. + M_{\rm sb}^2\left(2\delta Z_{\rm e}^{\rm fin} + \frac{c_{\rm w}^2 - s_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm W}^{2,\rm fin}}{M_{\rm W}^2} - \frac{c_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm Z}^{2,\rm fin}}{M_{\rm Z}^2}\right)\Big]\right], \qquad (9)
\end{aligned}$$

$$\begin{aligned}
\delta\alpha^{\delta\lambda_3^{\overline{\rm MS}}} &= \delta\alpha^{\delta\lambda_{345}^{\overline{\rm MS}}} \\
&- \frac{c_\beta s_\beta}{c_\alpha^2 - s_\alpha^2}\frac{2}{M_{H_{\rm h}}^2 - M_{H_{\rm l}}^2}\left[\delta M_{H^\pm}^{2,\rm fin} - \delta M_{\rm sb}^{2,\rm fin}\right. \\
&\left. + \left(M_{H^\pm}^2 - M_{\rm sb}^2\right)\left(\delta Z_{\rm e}^{\rm fin} + \frac{c_{\rm w}^2 - s_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm W}^{2,\rm fin}}{M_{\rm W}^2} - \frac{c_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm Z}^{2,\rm fin}}{M_{\rm Z}^2}\right)\right]. \qquad (10)
\end{aligned}$$

In the HSESM only the $\delta\lambda_3^{\overline{\rm MS}}$ scheme exists which can be defined via $\delta\alpha$ as follows:

$$\begin{aligned}
\delta\alpha^{\delta\lambda_3^{\overline{\rm MS}}} &= \delta\alpha^{\overline{\rm MS}} - \frac{c_\alpha s_\alpha}{c_\alpha^2 - s_\alpha^2} 2\delta Z_{\rm e}^{\rm fin} \\
&- \frac{c_\alpha s_\alpha}{c_\alpha^2 - s_\alpha^2}\left[\frac{c_{\rm w}^2 - s_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm W}^{2,\rm fin}}{M_{\rm W}^2} - \frac{c_{\rm w}^2}{s_{\rm w}^2}\frac{\delta M_{\rm Z}^{2,\rm fin}}{M_{\rm Z}^2} + \frac{\delta m_{H_{\rm h}}^{2,\rm fin} - \delta m_{H_{\rm l}}^{2,\rm fin}}{M_{H_{\rm h}}^2 - M_{H_{\rm l}}^2}\right]. \qquad (11)
\end{aligned}$$

$\delta\beta^{\overline{\rm MS}}$**:** In the 2HDM we get for β in the FJTS:

$$\delta\beta_{\rm FJTS}^{\overline{\rm MS}} := \delta\beta^{\overline{\rm MS}} = \frac{\delta Z_{H_{\rm a}G_0}^{\overline{\rm MS}} - \delta Z_{G_0H_{\rm a}}^{\overline{\rm MS}}}{4} = \frac{\delta Z_{H^\pm G^\pm}^{\overline{\rm MS}} - \delta Z_{G^\pm H^\pm}^{\overline{\rm MS}}}{4}, \qquad (12)$$

[3] See also Eqs. (4.38) and (4.39) in Ref. [36].

while the results in the other tadpole counterterm schemes read

$$\delta\beta^{\overline{\rm MS}}_{\rm MDTS} = \delta\beta^{\overline{\rm MS}}_{\rm MTS} = \delta\beta^{\overline{\rm MS}} + \left(c_{\alpha\beta} \frac{\delta t^{\rm fin}_{H_{\rm l}}}{M^2_{H_{\rm l}}} + s_{\alpha\beta} \frac{\delta t^{\rm fin}_{H_{\rm h}}}{M^2_{H_{\rm h}}} \right) \frac{e}{2 M_{\rm W} s_{\rm w}}. \tag{13}$$

The $\overline{\rm MS}$ scheme (13) corresponds to the popular $\overline{\rm DR}$ ($\overline{\rm MS}$) scheme in the Minimal Supersymmetric SM (see e.g. Ref. [32]). Note that the finite parts of the tadpoles are, in general, gauge dependent, and we fix them in the 't Hooft–Feynman gauge. The $\overline{\rm MS}$/MDTS scheme is used by <u>default</u> for $\delta\beta$.

$\delta t_\beta^{\overline{\rm MS}}$: In the HSESM we renormalize t_β in the $\overline{\rm MS}$ scheme using the vertex $V_{H_{\rm h} H_{\rm l} H_{\rm l}}$ and require the pole part (P.P.) to vanish:

$$V_{H_{\rm h} H_{\rm l} H_{\rm l}}|_{\rm P.P.} \stackrel{!}{=} 0 \quad \Rightarrow \quad \delta t_\beta^{\overline{\rm MS}} =: \delta t^{\overline{\rm MS}}_{\beta,{\rm FJTS}}. \tag{14}$$

For t_β we offer the possibility to switch to the MDTS scheme which is related to the FJTS via

$$\begin{aligned} \delta t^{\overline{\rm MS}}_{\beta,{\rm MDTS}} &= \delta t_\beta^{\overline{\rm MS}} \\ &\quad + \left((s_\alpha + c_\alpha t_\beta) \frac{\delta t^{\rm fin}_{H_{\rm l}}}{M^2_{H_{\rm l}}} - (c_\alpha - s_\alpha t_\beta) \frac{\delta t^{\rm fin}_{H_{\rm h}}}{M^2_{H_{\rm h}}} \right) \frac{e}{2 M_{\rm W} s_{\rm w}}. \end{aligned} \tag{15}$$

We do not support the MTS for t_β in the HSESM. The finite parts of the tadpoles are, in general, gauge dependent, and we fix them in the 't Hooft–Feynman gauge. The $\overline{\rm MS}$/MDTS scheme is used by <u>default</u> for δt_β in the HSESM.

$\delta M^{2,\overline{\rm MS}}_{\rm sb}$, $\delta m^{2,\overline{\rm MS}}_{12}$: We determine $\delta M^{2,\overline{\rm MS}}_{\rm sb}$ from the vertex $V_{H_{\rm h} H^\pm H^\pm}$ by requiring that the pole part (P.P.) vanishes:

$$V_{H_{\rm h} H^\pm H^\pm}|_{\rm P.P.} \stackrel{!}{=} 0 \quad \Rightarrow \quad \delta M^{2,\overline{\rm MS}}_{\rm sb} =: \delta M^{2,\overline{\rm MS}}_{\rm sb,FJTS}. \tag{16}$$

<u>This scheme is the default</u>. Alternatively, m^2_{12} can be renormalized $\overline{\rm MS}$ which is equivalent to define

$$\delta M^{2,\delta m^{2,\overline{\rm MS}}_{12}}_{\rm sb} = \delta M^{2,\overline{\rm MS}}_{\rm sb} + M^2_{\rm sb} \frac{s^2_\beta - c^2_\beta}{c_\beta s_\beta} \delta\beta^{\rm fin}. \tag{17}$$

- p^* schemes [19, 30, 31, 34, 35]:
  The p^* scheme is derived via the diagonalization of the effective mass matrix requiring vanishing scale dependence [30, 31]. In the 2HDM, this scheme can be used also for the renormalization of the mixing angle β yielding the following solutions:

  $\delta\alpha^{p^*}$:

$$\delta\alpha^{p^*} = \frac{\Sigma^{\text{1PI,BFM}}_{H_\text{h}H_\text{l}}\left(\frac{M^2_{H_\text{h}}+M^2_{H_\text{l}}}{2}\right) + t_{H_\text{l}H_\text{h}}}{M^2_{H_\text{h}} - M^2_{H_\text{l}}}, \tag{18}$$

  $\delta\beta^{p^*_1}$:

$$\delta\beta^{p^*_1} = -\frac{\Sigma^{\text{1PI,BFM}}_{H_\text{a}G_0}\left(\frac{M^2_{H_\text{a}}}{2}\right) + t_{H_\text{a}G_0}}{M^2_{H_\text{a}}}, \tag{19}$$

  $\delta\beta^{p^*_2}$:

$$\delta\beta^{p^*_2} = -\frac{\Sigma^{\text{1PI,BFM}}_{H^\pm G^\pm}\left(\frac{M^2_{H^\pm}}{2}\right) + t_{H^\pm G^\pm}}{M^2_{H^\pm}}. \tag{20}$$

  Here, $\Sigma_{H_\text{h}H_\text{l}}$, $\Sigma_{H_\text{a}G_0}$ and $\Sigma_{H^\pm G^\pm}$ denote the neutral, pseudo-scalar, and charged scalar mixing-energy, respectively, and $t_{H_\text{l}H_\text{h}}$, $t_{H_\text{a}G_0}$, $t_{H^\pm G^\pm}$ are the corresponding tadpole counterterms. The schemes are formulated via the BFM in the 't Hooft–Feynman gauge, i.e. for the quantum gauge parameter $\xi_Q = 1$ [40]. The scheme $\delta\alpha^{p^*}$ is valid in the 2HDM and HSESM, whereas the schemes $\delta\beta^{p^*_1}$ and $\delta\beta^{p^*_2}$ can only be used in the 2HDM.

- on-shell schemes Refs. [19, 29, 35]:
  In these schemes, the counterterms for the mixing angles are defined via on-shell mixing field-renormalization constants in the $\xi_Q = 1$ gauge in the BFM. We express the counterterms in terms of mixing energies as follows

  $\delta\alpha^{\text{OS}}$:

$$\delta\alpha^{\text{OS}} = \frac{\Sigma^{\text{1PI,BFM}}_{H_\text{h}H_\text{l}}\left(M^2_{H_\text{h}}\right) + \Sigma^{\text{1PI,BFM}}_{H_\text{l}H_\text{h}}\left(M^2_{H_\text{l}}\right) + 2t_{H_\text{l}H_\text{h}}}{2\left(M^2_{H_\text{h}} - M^2_{H_\text{l}}\right)}, \tag{21}$$

$\delta\beta^{\mathrm{OS}_1}$**:**

$$\delta\beta^{\mathrm{OS}_1} = -\frac{\Sigma^{\mathrm{1PI,BFM}}_{H_\mathrm{a}G_0}(0) + \Sigma^{\mathrm{1PI,BFM}}_{H_\mathrm{a}G_0}\left(M^2_{H_\mathrm{a}}\right) + 2t_{H_\mathrm{a}G_0}}{2M^2_{H_\mathrm{a}}}, \tag{22}$$

$\delta\beta^{\mathrm{OS}_2}$**:**

$$\delta\beta^{\mathrm{OS}_2} = -\frac{\Sigma^{\mathrm{1PI,BFM}}_{H^\pm G^\pm}(0) + \Sigma^{\mathrm{1PI,BFM}}_{H^\pm G^\pm}\left(M^2_{H^\pm}\right) + 2t_{H^\pm G^\pm}}{2M^2_{H^\pm}}. \tag{23}$$

The scheme $\delta\alpha^{\mathrm{OS}}$ is valid in the 2HDM and HSESM, whereas the schemes $\delta\beta^{\mathrm{OS}_1}$ and $\delta\beta^{\mathrm{OS}_2}$ can only be used in the 2HDM.

2.2. $\overline{\mathrm{MS}}$ *renormalization and scale dependence*

RECOLA2 distinguishes between poles in ϵ of infrared (IR) and ultraviolet (UV) origin by introducing the parameters μ_{UV}, ϵ_{UV} and μ_{IR}, ϵ_{IR} in all tensor integrals and counterterms, together with

$$\begin{aligned} \Delta_{\mathrm{UV}} &= \frac{(4\pi)^{\epsilon_{\mathrm{UV}}}\,\Gamma(1+\epsilon_{\mathrm{UV}})}{\epsilon_{\mathrm{UV}}}, \\ \Delta_{\mathrm{IR}} &= \frac{(4\pi)^{\epsilon_{\mathrm{IR}}}\,\Gamma(1+\epsilon_{\mathrm{IR}})}{\epsilon_{\mathrm{IR}}}, \qquad \Delta_{\mathrm{IR2}} = \frac{(4\pi)^{\epsilon_{\mathrm{IR}}}\,\Gamma(1+\epsilon_{\mathrm{IR}})}{\epsilon^2_{\mathrm{IR}}}. \end{aligned} \tag{24}$$

Following the conventions of COLLIER [26, 41] and Ref. [42], the parameters Δ_{UV}, Δ_{IR} and Δ_{IR2} that contain the poles in ϵ absorb a normalization factor of the form $1 + \mathcal{O}(\epsilon)$. In terms of these parameters, the one-loop amplitude $\mathcal{A}_1$ takes the general form

$$\mathcal{A}_1 = \Delta_{\mathrm{UV}}\,\mathcal{A}_1^{\mathrm{UV}} + \Delta_{\mathrm{IR2}}\,\mathcal{A}_1^{\mathrm{IR2}} + \Delta_{\mathrm{IR}}\,\mathcal{A}_1^{\mathrm{IR}}(\mu_{\mathrm{IR}}) + \mathcal{A}_1^{\mathrm{fin}}(\mu_{\mathrm{UV}}, \mu_{\mathrm{IR}}). \tag{25}$$

The term $\mathcal{A}_1^{\mathrm{UV}}$ vanishes after (UV) renormalization. The a priori unphysical scale μ_{UV} should either cancel between the tensor integrals and the counterterms, or it should receive a physical interpretation as for example when it is identified with the renormalization scale μ_{MS} in the $\overline{\mathrm{MS}}$ scheme[4] for the strong coupling constant g_s. In RECOLA2, the $\overline{\mathrm{MS}}$ renormalization scale μ_{MS}

[4] In the original RECOLA version as well in the literature the scale μ_{MS} is called Q.

is introduced as a new independent scale by a modified $\overline{\text{MS}}$ renormalization where instead of Δ_{UV} the term

$$\Delta_{\text{UV}} + \ln \frac{\mu_{\text{UV}}^2}{\mu_{\text{MS}}^2} \tag{26}$$

is subtracted in $\overline{\text{MS}}$ schemes. As a consequence, the renormalized one-loop amplitude becomes

$$\mathcal{A}_1 = \Delta_{\text{IR2}}\, \mathcal{A}_1^{\text{IR2}} + \Delta_{\text{IR}}\, \mathcal{A}_1^{\text{IR}}(\mu_{\text{IR}}) + \mathcal{A}_1^{\text{fin}}(\mu_{\text{MS}}, \mu_{\text{IR}}) \tag{27}$$

which is independent of μ_{UV} but can depend on μ_{MS} when parameters are renormalized $\overline{\text{MS}}$. Note that in the extended Higgs sectors besides the strong coupling constant g_s additional sources of scale dependence emerge. The numerical evaluation of the renormalized amplitude involves parts which depend on Δ_{UV} and μ_{UV} at intermediate steps. The independence of $\mathcal{A}_1$ on Δ_{UV} and μ_{UV} can be verified numerically by varying these parameters.

2.2.1. Soft and collinear singularities

In RECOLA2 collinear singularities are regularized as in RECOLA according to the input fermion masses which are forwarded to COLLIER, while soft singularities are always regularized dimensionally.[5] In the case of massless fermions dimensional regularization is used, and the scale dependence of regularized integrals is parametrized by the IR scale μ_{IR}. If, on the other hand, the fermion has been assigned a regulator mass, its collinear singularities are regularized by this mass parameter. In this case, the parameter μ_{IR} can be interpreted as a photon-mass regulator.

2.3. Background-Field Method

RECOLA2 supports the Background-Field Method (BFM) as a complementary method to the usual formulation of QFT. The implementation is realised for each individual model as a separate model file. Whether a model file is formulated in the BFM is printed in the `Gauge` entry in the initialisation of the model. For example, the SM BFM initialisation reads:

[5] In RECOLA the regularization of soft singularities is steered by the parameter `reg_soft`. Associated input routines can still be called in RECOLA2, but are deprecated and have no effect.

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                          _    _   _   _        _    _
                         | )  |_  |   | |  |   |_|   _|
                         | \  |_  |_  |_|  |_  | |  |_

                  REcursive Computation of One Loop Amplitudes

                                 Version 2.0.0

                           by A.Denner, J.-N.Lang, S.Uccirati

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

--------------------------------------------------------------------------------
 Active model:     SM
 Gauge:            't Hooft-Feynman BFM
 Model generation: Tue Aug 29 18:58:55 2017
--------------------------------------------------------------------------------
```

The BFM and the usual formulation of QFT are treated on equal footing, and, from the user perspective, models formulated in the BFM can be used as any other model since the same conventions for background fields in the BFM and the usual fields (Section 2.4) were chosen. The BFM comes with additional fields, so-called quantum fields, which cannot be selected as external states of matrix elements because they only appear inside loops. When drawing the currents to one-loop scattering processes the quantum fields can be seen at intermediate steps. For instance, the propagation of a gluon quantum field g_Q along the loop line (the line distinguished by the cross at the beginning and end) is visualized in Fig. 1. Note that for all model files formulated in the BFM the fermion fields take the role of both, quantum and background field and no explicit distinction is made [19].

All renormalization schemes available in the usual formulation are also available in the BFM allowing for powerful cross checks. There are, however, (higher-order) differences for mass counterterms due to the expansion performed within the complex-mass scheme (see Ref. [19]). For a precise numerical comparison of both methods it is recommended to set the widths of all particles to zero where these differences disappear. In practical calculations the BFM formulation is $10-15\%$ faster than the conventional formulation which is due to the need of less current structures. On the other hand, the generation of processes takes $\sim 50\%$ longer in the BFM.

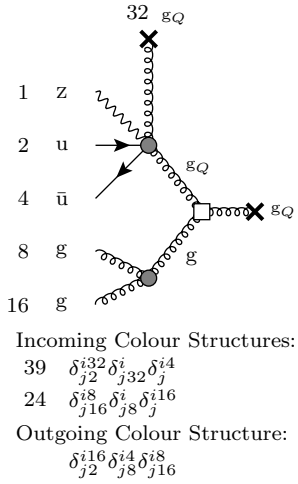

Figure 1: Visualization of a branch with a quantum field.

2.4. Conventions

RECOLA2 uses the same conventions for particle identifiers (of type `character`) as RECOLA [14] with the exception of the Goldstone bosons:

Scalars in the HSESM:	`'Hl', 'Hh'`
Scalars in the THDM:	`'Hl', 'Hh', 'Ha', 'H+', 'H-'`
Goldstone bosons:	`'G0', 'G+', 'G-'` .

Other SM fields as well as conventions for polarizations and the normalization of the cross section are given in Ref. [14].

3. Installation

RECOLA2 uses the COLLIER library and RECOLA2-specific model files. In order to facilitate building RECOLA2 we provide the following two options:

- RECOLA2-COLLIER package (Section 3.1):
  The configuration and compilation of RECOLA2, COLLIER, and a model file is performed at once, using a CMAKE script which resolves all dependencies automatically.

- RECOLA2-stand-alone package (Section 3.2):
  A stand-alone version which requires the user to resolve the dependence to COLLIER and model files by hand.

The RECOLA2 library and model files are available from the web site `http://recola.hepforge.org`. The compilation requires the CMAKE build system.

3.1. The RECOLA2-COLLIER *package*

The package `recola2-collier-X.Y.Z` contains the version `X.Y.Z` of the RECOLA2 library together with all model files. After downloading the file `recola2-collier-X.Y.Z.tar.gz`, extract the tarball in the current working directory with the shell command:

```
tar -zxvf recola2-collier-X.Y.Z.tar.gz
```

This operation creates the directory `recola2-collier-X.Y.Z` containing the following files and folders:

- `CMakeLists.txt`, `build`:
  CMAKE configuration for the compilation of the RECOLA2, COLLIER and model-file libraries in the proper order. The build directory, where CMAKE puts all necessary files for the creation of the library;
- `recola2-X.Y.Z`:
  main directory of the RECOLA2 package `recola2-X.Y.Z` (see Section 3.2 for details);
- `model-files-X.Y.Z`:
  directory containing the RECOLA2 model files:
  - `SM_X.Y.Z`, `SM_BFM_X.Y.Z`
  - `HS_X.Y.Z`, `HS_BFM_X.Y.Z`
  - `THDM_X.Y.Z`, `THDM_BFM_X.Y.Z`

  Each model carries the same version as the RECOLA2 library;
- `project_cmake`:
  a test configuration which demonstrates how to link RECOLA2 to an external project using CMAKE; this is discussed in Section 3.1.2.

By running

```
cd recola2-collier-X.Y.Z/build
cmake [options] .. -Dmodel=<model>
make [options]
```

the RECOLA2 library is compiled and linked with the model `<model>` and the COLLIER library. A predefined version of the COLLIER library is automatically downloaded[6] and extracted next to the RECOLA2 sources upon invoking the `make` command. The variable `<model>` can be one of the model ids `SM`, `SM_BFM`, `HS`, `HS_BFM`, `THDM`, `THDM_BFM`, or it can be an absolute or relative path pointing to the model-file sources. Selecting a different model file requires to rerun the CMAKE configuration and compilation with a different value for `model`. The `[options]` for the configuration (`cmake`) and compilation (`make`) are explained in Section 3.2 and summarized in Table 2. It is recommended to run the compilation parallelised with the Makefile option `-j`. Once the compilation of RECOLA2 is finished the demo files (see Section 3.1.1) can be run.

3.1.1. The RECOLA2 *demo files*

The RECOLA2 demo files are located in `recola2-X.Y.Z`/demos. In order to compile and run executables for the demo programs the command

```
./run <demofile>
```

can be executed with `<demofile>` being either `demo0_rcl`, `demo1_rcl`, `demo2_rcl`, `demo3_rcl`, `demo4_rcl`, or `demo5_rcl`. Alternatively, the user can execute

```
make <demofile>
```

in the build directory `recola2-X.Y.Z/build` to compile the respective executable.

The demo programs exemplify the usage of RECOLA2 for various purposes:

- `demo0_rcl`:
  Basic usage of RECOLA2.

- `demo1_rcl`:
  Usage of RECOLA2 for more than one process simultaneously, with explicit modification of input parameters and with selection of specific helicities for the external particles and of certain powers of the strong

[6] COLLIER is downloaded from http://collier.hepforge.org If the COLLIER source is already present no download step is performed.

CMake option	Value	Short description
`collier_path`	Path	Absolute or relative path to the COLLIER library.
`modelfile_path`	Path	Absolute or relative path to the RECOLA2 model file. Only available in RECOLA2 `CMakeLists.txt`.
`static`	On/Off	Compile the library as a shared or static library.
`with_python3`	On/Off	Choose PYTHON 3.X over PYTHON 2.7 to compile `pyrecola`. Only available in RECOLA2 `CMakeLists.txt`.
`with_smtests`	On/Off	Run tests against POLE and OPENLOOPS. Only available in RECOLA2 `CMakeLists.txt`.
`CMAKE_BUILD_TYPE`	Debug/Release	Set the compiler flags. By default Release flags (optimized) are used.
`CMAKE_Fortran_COMPILER`	Path/Name	Set the FORTRAN compiler either via executable name or the absolute path to executable.
`CMAKE_INSTALL_PREFIX`	Path	Set the installation prefix.
Makefile option	**Value**	**Short description**
`-j`	Integer	Number of threads for compilation.
`VERBOSE`	True/False	Enable *verbose* compilation. In this mode all compilation flags are visible to the user.

Table 2: Summary of the CMake and Makefile options.

coupling constant. In addition, files with LaTeX source code for diagrams are generated.

- `demo2_rcl`:
  Usage of RECOLA2 for the selection of resonant contributions and pole approximation.

- `demo3_rcl`:
  Usage of RECOLA2 for the computation of colour- and/or spin-correlation.

- `demo4_rcl`:
  Usage of RECOLA2 for the computation of decay widths using the example of light and heavy neutral Higgs decays into Higgs and gauge bosons:

  - $H_\mathrm{h} \to H_\mathrm{l} H_\mathrm{l}$,
  - $H_\mathrm{h}/H_\mathrm{l} \to \mathrm{ZZ}, \quad H_\mathrm{h}/H_\mathrm{l} \to \mathrm{Z}\gamma, \quad H_\mathrm{h}/H_\mathrm{l} \to \gamma\gamma$,
  - $H_\mathrm{h}/H_\mathrm{l} \to \mathrm{gg}$.

  Moreover, the use of different renormalization schemes in the extended Higgs sector is demonstrated.

- `demo5_rcl`:
  Usage of RECOLA2 for the selection of powers of coupling types in the new system at the example of a vector-boson-fusion partonic channel.

Note that the demo files `demo0_rcl`, `demo1_rcl`, `demo2_rcl`, `demo3_rcl` are identical to the ones in RECOLA, but can be used with any current RECOLA2 model file.

For each `<demofile>` corresponding C++ (`cdemo`) and PYTHON (`pydemo`) demo files are available. The C++ demo files are compiled in the same way, substituting `<demofile>` by the precise `cdemo` file name. The C++ interface is described in Section 4.5. The PYTHON demo file can be run directly without compilation. See Section 4.6 for more details on the PYTHON interface.

The `demos` directory also contains the shell script `draw-tex` which compiles all LaTeX files of the form `process_*.tex` present in the folder and creates the corresponding `.pdf` files. It can be run by invoking

```
./draw-tex
```

in the `demos` directory. RECOLA2 provides the same functionally for drawing currents as RECOLA and we refer to Ref. [14] for details.

3.1.2. A minimal executable with CMAKE

The purpose of this section is to demonstrate how to link RECOLA2 to a custom program using the CMAKE build system. The RECOLA2-COLLIER package comes with a FORTRAN95 and C++ test program in the directory `project_cmake`. We discuss the FORTRAN95 version, `program.f90`, which merely contains the lines of code:

```
program main
  use recola
end program
```

That is, the program includes RECOLA2 and can be extended by the user at will. For the compilation the essential part is the CMAKE configure file `CMakeLists.txt`. A template `CMakeLists.txt` is provided:

```
cmake_minimum_required(VERSION 2.8)
project(example Fortran C CXX)

add_executable(program program.f90)

find_package(recola
             "X.Y.Z" EXACT REQUIRED
             HINTS "../install")

target_link_libraries(program ${RECOLA_LIBRARY_PATH})
```

The first two lines define a new project and declare FORTRAN as the main language.[7] The third statement adds a new executable named `program` which is compiled from the source `program.f90`. Since the program uses RECOLA2 it needs to be linked against it. This is achieved in two steps. In the first step, the RECOLA2 package is searched for via the CMAKE command `find_package`. The version requirement `exact` can be relaxed by removing it and omitting the minor/patch version. On success, this command fills the CMAKE variables

- `RECOLA_LIBRARY_DIR`:
  directory containing the RECOLA2 library;
- `RECOLA_INCLUDE_DIR`:
  directory containing the RECOLA2 header files, i.e. the FORTRAN modules `*.mod`, the C header file `recola.h`, and the C++ header file `recola.hpp`;
- `RECOLA_LIBRARY_PATH`:
  the absolute path to the RECOLA2 library file `librecola.so` (or `librecola.a`, `librecola.dynlib`).

[7] Change the compiler language to C via `project(example C)`. Multiple languages at the same time are allowed, e.g. `project(example Fortran C CXX)`. CMAKE automatically decides which compiler to take based on the suffix of files.

On failure, the configuration stops printing an error. After successfully finding the RECOLA2 package the executable is linked with the RECOLA2 library.

Note this script works independent of whether RECOLA2 is compiled as shared or static library, whether the underlying program is FORTRAN, C or C++ based, and independent of the underlying operating system or compilers. For further tuning we refer to `https://cmake.org/Wiki/CMake`.

3.2. The RECOLA2 *stand-alone package*

In this section we discuss the compilation of the stand-alone RECOLA2 library, the compilation of the model files, and all available compiler options. We assume the user has acquired a local copy of COLLIER[8] and that he/she followed the COLLIER compilation instructions.

The archives

- `recola2-X.Y.Z.tar.gz`
- `SM_X.Y.Z.tar.gz`, `SM_BFM_X.Y.Z.tar.gz`
- `THDM_X.Y.Z.tar.gz`, `THDM_BFM_X.Y.Z.tar.gz`
- `HS_X.Y.Z.tar.gz`, `HS_BFM_X.Y.Z.tar.gz`

are available at `http://recola.hepforge.org` and represent the RECOLA2 library and model files in the version `X.Y.Z`.

3.2.1. The RECOLA2 *model-file compilation*

In the current version of RECOLA2 the model files are a dependency at compile time,[9] thus, the desired model needs to be compiled first. To this end, extract one of the model-file tarballs with the shell command:

`tar -zxvf` *modelfile*`-X.Y.Z.tar.gz`

This operation creates the directory *modelfile*`-X.Y.Z` containing the following files and folders:

- `CMakeLists.txt`, `config`, `build`:
  CMAKE configure files required for the generation of the RECOLA2 model file Makefile. The build directory is where CMAKE puts all necessary files for the creation of the library;

[8] COLLIER can be downloaded from http://collier.hepforge.org.
[9] Dynamic loading of model files at run-time is not supported.

- `src`:
  model-file source directory;

- `include`:
  all FORTRAN module files `*.mod` are placed inside this folder.

The compilation of the model file proceeds by changing to the `build` directory[10] and executing there the shell command `"cmake [options] .."` (creating a Makefile in *modelfile*`-X.Y.Z/build`), followed by `make`:

```
cd  modelfile-X.Y.Z/build
cmake [options] ..
make [options]
```

By default, the configuration will search for the COLLIER library in directories next to the model-file directory. It is possible to force the use of a particular COLLIER version by providing a path. The CMAKE variable `collier_path` can be used to pass the (absolute/relative) path to the directory containing the COLLIER library:

```
-Dcollier_path=<PATH_TO_COLLIER>
```

By using this option it is assumed that the COLLIER module files are located inside the path:

```
<PATH_TO_COLLIER>/modules
```

As a third alternative the environment variables `COLLIER_INCLUDE_DIR` and `COLLIER_LIBRARY_DIR` can be set to the directories including the COLLIER modules and COLLIER library, respectively. In this case the complete sequence of calls reads:

```
cd  modelfile-X.Y.Z/build
export COLLIER_LIBRARY_DIR=<PATH_TO_COLLIER_LIBRARY>
export COLLIER_INCLUDE_DIR=<PATH_TO_COLLIER_MODULES>
cmake [options] ..
make [options]
```

[10] Changing to the build directory is optional but recommended. CMAKE populates the working directory with many configuration and object files. With a build directory those files can be cleaned easily by removing the contents of the build folder.

We stress that the library paths should not point to the precise COLLIER library file, but only to the directory containing the library.

If no other options are specified, CMAKE automatically searches for installed FORTRAN compilers and chooses a suited one. The user can force CMAKE to use a specific compiler by appending to the `cmake` command the option

```
-DCMAKE_Fortran_COMPILER=<comp>
```

where `<comp>` can be `gfortran, ifort, pgf95, ...` or the full path to a compiler.

By default, the installation sequence generates the model file as a shared library `libmodelfile.so(/.dynlib)` in the directory *modelfile*-`X.Y.Z`, with the corresponding module files placed in the `include` subdirectory. The option

```
-Dstatic=ON
```

causes CMAKE to create the static library instead of the shared one.

After the configuration the compilation can be run parallelised which is recommended as the model files contain plenty of independent source files. Parallelised compilation is performed via

```
make -j <THREADS>
```

where `<THREADS>` is the number of parallel compilation units. The CMAKE Makefile allows the compilation command to be run in *verbose* mode via:

```
make VERBOSE=True
```

If desired, the model file can be installed into the system via the command:

```
make install
```

A default install prefix is set automatically by CMAKE,[11] but can be altered by configuring the Makefile via

```
-DCMAKE_INSTALL_PREFIX=<INSTALLATION_PATH>
```

with `<INSTALLATION_PATH>` being a custom installation path, e.g. `$HOME`.

[11] The default installation path is set to inside the root system and requires root rights.

3.2.2. The RECOLA2 *library compilation*

The configuration and compilation of RECOLA2 proceeds in the very same way as for model files, with the exception that, in addition, a specific model-file path has to be set.

In the first step, extract the RECOLA2 tarball with the shell command:

```
tar -zxvf recola2-X.Y.Z.tar.gz
```

This operation creates the directory `recola2-X.Y.Z` containing the following files and folders:

- `CMakeLists.txt`, `config`, `build`:
  CMAKE configure files required for the generation of the RECOLA2 Makefile. The build directory is where CMAKE puts all necessary files for the creation of the library;

- `src`:
  RECOLA2 source directory;

- `demos`:
  directory with demo programs illustrating the use of RECOLA2, including shell scripts for their compilation and execution;

- `include`:
  directory with C and C++ header files; all FORTRAN module files `*.mod` are placed inside this folder;

The standard sequence for the compilation reads

```
cd recola2-X.Y.Z/build
cmake [options] .. -Dmodelfile_path=<PATH_TO_MODELFILE>
                   -Dcollier_path=<PATH_TO_COLLIER>
make [options]
```

where `<PATH_TO_MODELFILE>` and `<PATH_TO_COLLIER>` is the path to the directory containing the compiled model-file and COLLIER library, respectively. By using this option for COLLIER it is assumed that the COLLIER module files are located inside the path:

```
<PATH_TO_COLLIER>/modules
```

Alternatively, the environment variable `MODELFILE_PATH` can be set to the directory including the model file CMake configure files[12] or the model-file library. In this case the sequence of calls reads:

```
cd recola2-X.Y.Z/build
export MODELFILE_PATH=<PATH_TO_MODELFILE_CONFIG>
cmake [options] .. -Dcollier_path=<PATH_TO_COLLIER>
make [options]
```

We support the same `[options]` for configuration and compilation of Recola2 as for the model files. Note also that the paths to the Collier library and module files can be set via environment variables as described in Section 3.2.1.

Starting with Recola2 we support a Python interface. By default, the CMake configuration will search for the Python 2.7 library, header and executable and will, on success, build the library `pyrecola` (see Section 4.6). We support Python 3.x which can be enabled in the configuration via

```
-Dwith_python3=On
```

and, evidently, requires the Python 3.x library, header and executable to be present in the system. We note that the Recola2 build is not affected if the Python dependencies are not fulfilled.

Finally, Recola2 is equipped with a few test routines allowing to check the proper integration of Recola2 into the system. In order to be able to compile the tests the configuration needs to be run with:

```
-Dwith_smtests=On
```

Then, after building Recola2, the tests can be run via

```
make test
```

or by invoking

```
ctest
```

in the build directory. Note that the tests should only be run with a SM model file, otherwise the tests will fail.

[12] CMake generates configure files for built or installed model files. They are named `modelfileConfig.cmake` and `modelfileConfigVersion.cmake` and are required for linking to the Recola2 library.

4. Usage of RECOLA2 in extended Higgs sectors

In order to use RECOLA2 in a FORTRAN program its modules have to be loaded by including the line

```
use recola
```

in the preamble of the respective code, and the library `librecola.so`, `librecola.dynlib` or `librecola.a` has to be supplied to the linker. This gives access to the public functions and subroutines of the RECOLA2 library described in the following subsections. The names of all these routines end with the suffix "`_rcl`". This name convention is supposed to avoid conflicts with routine names present in the master program and increases readability by allowing for an easy identification of command lines referring to the RECOLA2 library.

Typically, an application of RECOLA2 involves the following five steps:

- **Step 1: Setting input parameters (optional)**

  The input needed for the computation of processes can be set by calling dedicated subroutines as provided by RECOLA2. See Section 4.2 for the additional set of methods related to extended Higgs sectors. Since RECOLA2 provides default values for all input parameters, this first step is optional.

- **Step 2: Defining the processes**

  Before RECOLA2 can be employed to calculate matrix elements for one or more processes, each process must be declared and labelled with a unique identifier. This is done by calling the subroutine `define_process_rcl` for every process, as described in Ref. [14]. The functionality of the process definition has been extended which is documented in Section 4.3 for subroutines that are concerned.

- **Step 3: Generating the processes**

  In the next step the subroutine `generate_processes_rcl` is called which triggers the initialisation of the complete list of processes defined in step 2. As a result, all relevant building blocks for the recursive computation of off-shell currents are generated (see Ref. [14] for details).

- **Step 4: Computing the processes**

  The computation of the amplitude and of the squared amplitude is performed by means of the subroutine `compute_process_rcl`, which uses the process-dependent information on the recursive procedure derived in step 3. The subroutine `compute_process_rcl` is called with the momenta of the external particles provided by the user. In a Monte Carlo integration, the call of `compute_process_rcl` is repeated many times for different phase-space points.

  Detailed information on the subroutines that can be employed in step 4 can be found in Ref. [14]. The functionality of the process computation has been extended which is documented in Section 4.4 for subroutines that are concerned.

- **Step 5: resetting** RECOLA2

  Finally, by calling the subroutine `reset_recola_rcl`, the process-dependent information generated in steps 2–4 is deleted and the corresponding memory is deallocated. The input variables keep their values defined in step 1 before.

Note that these steps have to be followed in the order given above. In particular, after step 3 no new process can be defined unless RECOLA2 is reset (step 5). After step 5 the user can restart with step 1 or step 2. More information on the allowed sequence of calls can be found in Ref. [14].

Examples are found in the directory `demos` and are described in Section 3.1.1.

4.1. Input subroutines for parameters of extended Higgs sectors

The following input subroutines are complementary subroutines to the original ones in RECOLA and can be used to set input parameters and renormalization schemes in extended Higgs sectors. Note that these subroutines can only be called if supported by the selected model file, otherwise RECOLA2 prints an error message and stops.

4.1.1. `set_pole_mass_hl_hh_rcl (ml,gl,mh,gh)`

This subroutine sets the pole masses and widths (in GeV) of the light (H_l) and heavy (H_h) Higgs bosons to `ml`, `gl` and `mh`, `gh`, respectively (`ml`, `gl`, `mh` and `gh` are of type `real(dp)`). The pole masses must fulfil the condition `ml`$<$ `mh`. Note that the degenerate mass scenario is not supported.

Model support: 2HDM HSESM

4.1.2. `set_pole_mass_ha_rcl (m,g)`

This subroutine sets the pole mass and width (in GeV) of the pseudo scalar Higgs boson (H_a) to `m` and g, respectively (`m` and g are of type `real(dp)`).

Model support: 2HDM

4.1.3. `set_pole_mass_hc_rcl (m,g)`

This subroutine sets the pole mass and width (in GeV) of the charged Higgs boson ($H^\pm$) to `m` and g, respectively (`m` and g are of type `real(dp)`).

Model support: 2HDM

4.1.4. `set_Z2_thdm_yukawa_type_rcl (ytype)`

This subroutine sets the Yukawa type in the softly-broken Z2 symmetric 2HDM to `ytype`. The variable `ytype` is of type `integer` and accepts the following values:

`ytype`	label	zero parameters
1	Type-I	h1u=h1d=h1l=0
2	Type-II	h1u=h2d=h2l=0
3	Type-X	h1u=h1d=h2l=0
4	Type-Y	h1u=h2d=h1l=0

See Section 2.1.2 for more information.

Model support: 2HDM

4.1.5. `set_tb_cab_rcl (tb,cab)`

This subroutine sets the value of t_β to `tb`, and $c_{\alpha\beta}$ to `cab` [`tb` and `cab` are of type `real(dp)`]. The value of `tb` is strictly greater than zero and the value of `cab` must fulfil $-1 \leq$ `cab` ≤ 1. If these conditions are violated an error is raised. The renormalization for $c_{\alpha\beta}$ and t_β is fixed via the renormalization of α and β which can be set using the subroutines `use_mixing_alpha_`*rs*`_scheme_rcl` (4.1.6) and `use_mixing_beta_`*rs*`_scheme_rcl` (4.1.7), respectively. See Section 2.1.3 for more details.

Model support: 2HDM

4.1.6. `use_mixing_alpha_`*rs*`_scheme_rcl (`*s*`)`

These subroutines (*rs*=`msbar`,`onshell`) set the renormalization scheme for the mixing angle α or a derivative thereof to *s* (of type `character`). The following $\overline{\text{MS}}$ and on-shell schemes are supported:

rs	*s*	renormalization-scheme description
`msbar`	`'FJTS'`	α renormalized $\overline{\text{MS}}$, FJ tadpole scheme (6)
`msbar`	`'MDTS'`	α renormalized $\overline{\text{MS}}$, MD tadpole scheme (7)
`msbar`	`'MTS'`	α renormalized $\overline{\text{MS}}$, minimal tadpole scheme (8)
`msbar`	`'l3'`	λ_3 renormalized $\overline{\text{MS}}$ (10),(11)
`msbar`	`'l345'`	λ_{345} renormalized $\overline{\text{MS}}$ (9)
`onshell`	`'ps'`	$\delta\alpha$ defined in the p^* scheme (18)
`onshell`	`'os'`	$\delta\alpha$ defined in the on-shell scheme (21)

Note that only one of the schemes can be used at a time. The $\overline{\text{MS}}$ scheme `'l345'` can only be used with the 2HDM. For more information on the schemes consider Section 2.1.3.

Model support: 2HDM HSESM

4.1.7. `use_mixing_beta_`*rs*`_scheme_rcl (`*s*`)`

These subroutines (*rs*=`msbar`,`onshell`) set the renormalization scheme for the mixing angle β to *s* (of type `character`). The following $\overline{\text{MS}}$ and on-shell schemes are supported:

rs	*s*	renormalization-scheme description
`msbar`	`'FJTS'`	β is renormalized $\overline{\text{MS}}$, FJ tadpole scheme (12)
`msbar`	`'MDTS'`	β is renormalized $\overline{\text{MS}}$, MD tadpole scheme (13)
`onshell`	`'ps1'`	$\delta\beta$ defined in p^* scheme via mixing $H_{\text{a}}G_0$ (19)
`onshell`	`'ps2'`	$\delta\beta$ defined in p^* scheme via mixing $H^{\pm}G^{\pm}$ (20)
`onshell`	`'os1'`	$\delta\beta$ defined in on-shell scheme via mixing $H_{\text{a}}G_0$ (22)
`onshell`	`'os2'`	$\delta\beta$ defined in on-shell scheme via mixing $H^{\pm}G^{\pm}$ (23)

Note that only one of the schemes can be used at a time. For more information on the schemes consider Section 2.1.3.

Model support: 2HDM

4.1.8. `set_msb_rcl (msb)`

This subroutine sets the value of the soft-Z_2-breaking scale $M_{\rm sb}$ to `msb` (`msb` is of type `real(dp)`). The renormalization scheme for $M_{\rm sb}$ is set with `use_msb_`*rs*`_scheme_rcl` (4.1.9). See Section 2.1.1 for more details.

Model support: 2HDM

4.1.9. `use_msb_msbar_scheme_rcl(`*s*`)`

This subroutine sets the renormalization scheme for soft-Z_2-breaking scale $M_{\rm sb}$ or the soft-Z_2-breaking parameter m_{12} to s (of type `character`). The following schemes are supported:

s	renormalization-scheme description
`'MSB'`	$M_{\rm sb}$ is renormalized $\overline{\rm MS}$ (16)
`'m12'`	m_{12} is renormalized $\overline{\rm MS}$ (17)

Note that only one of the schemes can be used at a time. The input value of $M_{\rm sb}$ is set with `set_msb_rcl` (4.1.8). See Section 2.1.3 for more details.

Model support: 2HDM

4.1.10. `set_sa_rcl (sa)`

This subroutine sets the value of s_α to `sa` [`sa` is of type `real(dp)`]. The input value of `sa` must fulfil $-1 \leq$ `sa` ≤ 1, otherwise an error is raised. The renormalization scheme for s_α is fixed via the renormalization of α which can be set with `use_mixing_alpha_`*rs*`_scheme_rcl` (4.1.6). See Section 2.1.1 for more details.

Model support: HSESM

4.1.11. `set_tb_rcl (tb)`

This subroutine sets the value of t_β to `tb` [`tb` is of type `real(dp)`]. The passed value of `tb` must fulfil `tb` > 0, otherwise an error is raised. The renormalization scheme for t_β is set with `use_tb_`*rs*`_scheme_rcl` (4.1.12). See Section 2.1.1 for more details.

Model support: HSESM

4.1.12. `use_tb_msbar_scheme_rcl (s)`

This subroutine sets the renormalization scheme for t_β to s (of type `character`). The following schemes are supported:

s	renormalization-scheme description
`'FJTS'`	t_β is renormalized $\overline{\text{MS}}$, FJ tadpole scheme (14)
`'MDTS'`	t_β is renormalized $\overline{\text{MS}}$, MD tadpole scheme (15)

Note that only one of the schemes can be used at a time. The input value of t_β is set with `set_tb_rcl` (4.1.11). See Section 2.1.3 for more details.

Model support: HSESM

4.2. Compatibility with RECOLA *input subroutines*

In this section we list new subroutines, original ones with modified behaviour, and no longer supported ones that are kept for backward compatibility. We stress that all subroutines existing in RECOLA can be called in RECOLA2.

4.2.1. `use_gfermi_scheme_rcl (g,a)`

⚠ This subroutine (see Section 4.2.18 in Ref. [14]) has a different behaviour when being used in combination with the optional argument g which represents the Fermi constant G_F. Internally, g is translated to α according to

$$\alpha = \frac{\sqrt{2} g M_\text{W}^2}{\pi} \left(1 - \frac{M_\text{W}^2}{M_\text{Z}^2}\right), \tag{28}$$

with the currently active values for the (real) vector-boson masses M_W, M_Z. Therefore, the vector-boson masses should be set before calling this subroutine.

4.2.2. `set_parameter_rcl(param,value)`

This subroutine sets the value of `param` (of type `character`) to `value` (of type `complex(dp)`). Note that only independent couplings, masses and widths can be set via this subroutine, without any consistency checks being performed. An imaginary part of `value` can lead to undefined behaviour, and `value` should be real even though it is of type `complex(dp)`. The allowed values for `param` depend on the model file and can be looked up in

the subroutine `set_parameter_mdl` defined in `class_particles.f90` of the respective model file. For example, setting the light Higgs-boson mass M_{H_1} to 125 GeV in the 2HDM or HSESM is achieved as follows:

```
call set_parameter_rcl("MHL", complex(125d0, 0d0))
```

For extended Higgs sectors the dedicated subroutines in Section 4.1 should be used to set input parameters. When calling `set_parameter_mdl` with arguments that are incompatible with the selected model file, a warning is printed. The program does not stop, and the call has no effect.

4.2.3. `set_renoscheme_rcl(ctparam,renoscheme)`

This subroutine sets the renormalization scheme for the parameter with name `param` (of type `character`) to `renoscheme` (of type `character`). The allowed values for `ctparam` and `renoscheme` depend on the model file and can be looked up in the subroutine `set_renoscheme_mdl` defined in `fill_ctparameters.f90` of the respective model file. For example, setting the renormalization scheme of the mixing angle α to the p^* scheme (18) in the 2HDM or HSESM is achieved as follows:

```
call set_renoscheme_rcl("da_QED2", "ps_bfm")
```

For extended Higgs sectors the dedicated subroutines in Section 4.1 should be used to set renormalization schemes. When calling `set_renoscheme_mdl` with arguments that are incompatible with the selected model file, a warning is printed. The program does not stop, and the call has no effect.

4.2.4. `use_dim_reg_soft_rcl`,
`use_mass_reg_soft_rcl (m)`,
`set_mass_reg_soft_rcl (m)`

⚠ These subroutines are deprecated and calling any of them has no effect. The regularization of light fermions is determined automatically. See Section 2.2.1 for details.

4.2.5. `set_complex_mass_scheme_rcl`,
`set_on_shell_scheme_rcl`

⚠ These subroutines are deprecated and calling any of them has no effect. Note that the model files are all derived in the complex-mass scheme.

4.2.6. `set_dynamic_settings_rcl (n)`

⚠ This subroutine is not supported yet and calling it has no effect.

4.2.7. `set_print_level_parameters_rcl (n)`

This subroutine, which can be called before the process generation but also during the process computation, sets internal variables governing the output of input, derived and counterterms parameters:

- `n = 0`: No parameters are printed.
- `n = 1`: Input parameters are printed.
- `n = 2`: Input and derived parameters are printed.
- `n = 3`: Input, derived and counterterm parameters are printed.

By default, only input parameters are printed.

4.2.8. `set_print_level_RAM_rcl (n)`

⚠ This subroutine is not supported yet and calling it has no effect.

4.2.9.
```
scale_coupling3_rcl (fac,pa1,pa2,pa3),
scale_coupling4_rcl (fac,pa1,pa2,pa3,pa4),
switchoff_coupling3_rcl (pa1,pa2,pa3),
switchoff_coupling4_rcl (pa1,pa2,pa3,pa4)
```

⚠ These subroutines are not supported yet and calling them has no effect.

4.2.10. `set_collier_output_dir_rcl (dir)`

This subroutine, which can be called before the process generation, sets the COLLIER output directory to `dir` (`dir` is of type `character`), with `dir` being a relative or absolute path. The default COLLIER output directory can be enforced by passing `dir = 'default'`.

4.3. Updates on process definition

In this section we describe the new treatment of powers of types of fundamental couplings in RECOLA2. In theories with a SM gauge group structure amplitudes are proportional to $g_s^{n_s} e^{n-n_s}$, where g_s and e represent the strong and electroweak gauge couplings. For a given process and power n_s in g_s the power n is unambiguously determined. For this special case of theories we

support the original RECOLA methods `select_gs_power_*` where only the powers in g_s are selected,[13] but for more general theories the user has to employ the new general selection methods given in Sections 4.3.1 and 4.3.2, exclusively available in RECOLA2.

The treatment of different coupling types is kept general and uses the information on coupling powers as defined by model files in the UFO format. For all model files we choose the powers in the strong and EW gauge couplings as follows:

coupling type	position	coupling constant
`'QCD'`	1	g_s
`'QED'`	2	e

The position is relevant only when calling a subroutine which requires the couplings powers as an array of integers, e.g. the subroutines in Section 4.4. For the selection of powers a string identifier is used as described in the following.

4.3.1. `select_power_BornAmpl_rcl (npr,cid,power)`, `unselect_power_BornAmpl_rcl (npr,cid,power)`

This pair of subroutines allows to select/unselect the contribution to the Born amplitude proportional to general powers in coupling types c_{id}^n of the underlying theory. Here, c_{id} is set to `cid` (of type `character`), n is given by the `integer` argument `power`, and `npr` is the process identifier (of type `integer`). The variable `cid` accepts the following values:

`'QCD'`: power in g_s,
`'QED'`: power in e.

All other contributions to the Born amplitude keep their status (selected or unselected), according to previous calls of selection subroutines. The selection of the contributions to the loop amplitude remains unaffected as well. New values for `cid` will be introduced in the future for theories with additional types of couplings (as appear, for instance, in theories with new gauge couplings or in effective field theories). For SM-like theories with the only two fundamental coupling types `'QCD'` and `'QED'` the methods `select_gs_power_BornAmpl_rcl` and `unselect_gs_power_BornAmpl_rcl` (see Ref. [14]) can be used instead.

[13] See Sections 4.3.2–4.3.6 in Ref. [14].

4.3.2. `select_power_LoopAmpl_rcl (npr,cid,power)`,
`unselect_power_LoopAmpl_rcl (npr,cid,power)`

This pair of subroutines allows to select/unselect the contribution to the loop amplitude proportional to general powers in coupling types c_{id}^n of the underlying theory. Here, c_{id} is set to `cid` (of type `character`), n is given by the `integer` argument `power`, and `npr` is the process identifier (of type `integer`). The variable `cid` accepts the following values:

`'QCD'`: power in g_s,
`'QED'`: power in e.

All other contributions to the loop amplitude keep their status (selected or unselected), according to previous calls of selection subroutines. The selection of contributions to the Born amplitude remains unaffected as well. New values for `cid` will be introduced in the future for theories with additional (gauge) couplings or in effective field theory. For SM-like theories with the only two fundamental coupling types `'QCD'` and `'QED'` the methods `select_gs_power_LoopAmpl_rcl` and `unselect_gs_power_LoopAmpl_rcl` (see Ref. [14]) can be used instead.

4.3.3. `select_all_gs_powers_BornAmpl_rcl (npr)`,
`unselect_all_gs_powers_BornAmpl_rcl (npr)`,
`select_all_powers_BornAmpl_rcl (npr)`,
`unselect_all_powers_BornAmpl_rcl (npr)`

These subroutines allow to select/unselect all contributions to the Born amplitude (with any power of g_s or general order) for the process with identifier `npr` (of type `integer`). The selection of contributions to the loop amplitude remains unaffected. The methods with g_s in their name have the same effect and are only kept for backward compatibility. In fact, all methods (un-)select all contributions.

4.3.4. `select_all_gs_powers_LoopAmpl_rcl (npr)`,
`unselect_all_gs_powers_LoopAmpl_rcl (npr)`,
`select_all_powers_LoopAmpl_rcl (npr)`,
`unselect_all_powers_LoopAmpl_rcl (npr)`

This pair of subroutines allows to select/unselect all contributions to the loop amplitude (with any power of g_s or general order) for the process with identifier `npr` (of type `integer`). The selection of contributions to the Born amplitude remains unaffected. The methods with g_s in their name have the

same effect and are only kept for backward compatibility. In fact, all methods (un-)select all contributions.

4.4. Updates for process computation

The methods `get_*_amplitude_rcl` have been extended to support the generalized treatment of fundamental types of couplings. The following methods are concerned

- `get_amplitude_rcl (npr,pow,order,colour,hel,A)`
  `pow` is an overloaded argument and can be either an `integer` specifying the power in g_s, or an array of `integer` values specifying general powers in types of couplings:

`pow`=n:	selects the contribution g_s^n to the amplitude;
`pow=[n,m,...]`:	selects the contribution $[g_s^n, e^m, \ldots]$ to the amplitude.

- `get_squared_amplitude_rcl (npr,pow,order,A2)`,
  `get_polarized_squared_amplitude_rcl (npr,pow,order,hel,A2h)`,
  `get_colour_correlation_rcl (npr,pow,i1,i2,A2cc)`,
  `get_spin_correlation_rcl (npr,pow,A2sc)`,
  `get_spin_colour_correlation_rcl (npr,pow,i1,i2,A2scc)`
  `pow` is an overloaded argument and can be either an `integer` specifying the power in α_s, or an array of `integer` values, specifying general powers in in types of couplings:

`pow`=n:	selects the contribution α_s^n to the amplitude squared;
`pow=[n,m,...]`:	selects the contribution $[g_s^n, e^m, \ldots]$ to the amplitude squared.

4.5. C++ interface

The RECOLA2 package includes a C++ interface[14] with the same naming conventions for the subroutines as given in Sections 4.2–4.6 in Ref. [14] and

[14] This interface is also used to link the original RECOLA version to SHERPA.

the new subroutines in Sections 4.1, 4.2, and 4.3. The basic usage is identical to the one in FORTRAN95 and follows the computation flow presented at the beginning of Section 4. In order to use RECOLA2 in a C++ program the RECOLA2 header file needs to be included as follows:

```
#include "recola.hpp"
```

This gives access to the RECOLA namespace. The functions are called in the typical C++ syntax, e.g.:

```
Recola::define_process_rcl(1, "u u -> u u", "NLO");
```

The namespace identifier "`Recola::`" can be omitted by importing RECOLA as follows:

```
#include "recola.hpp"
using namespace Recola
```

The FORTRAN95 subroutines translate to functions in C++ with the argument types being identified according to:

FORTRAN95	C++
integer	int
integer, dimension(:)	int[ ]
logical	bool
real(dp)	double
real(dp), dimension(:)	double[ ]
complex(dp)	std::complex⟨double⟩
character(len=*)	std::string

For multi-dimensional arrays, such as the momenta `p`, the conventions for the order of the indices is the same as in FORTRAN95, and any necessary transposition is performed within the interface. The complex numbers are the only additional C++ Standard Library dependency used in the interface.

Return values are handled by call-by-reference, thus, all C++ functions are declared as void functions. Optional arguments are implemented via function overloading, i.e. missing arguments are replaced by default values. For instance, the call

```
Recola::use_alphaZ_scheme_rcl ();
```

enables the use of the $\alpha(M_Z)$ scheme (see Section 4.2.2 in Ref. [14]), using the default value for $\alpha(M_Z)$ which is hard-coded in RECOLA. Providing an explicit argument via

```
Recola::use_alphaZ_scheme_rcl (0.0078125);
```

allows to use a different value for $\alpha(M_Z)$ than the default in the running session.

The original FORTRAN95 demo files are available as C++ demo files. The compilation of the C++ demo files follows the same steps as given in Section 3.1.1, i.e. by either running

```
make <demofile>
```

in the build folder or directly via the run script

```
./run <demofile>
```

in the demo folder, with `<demofile>` taking the values `cdemo0_rcl`, `cdemo1_rcl`, `cdemo2_rcl`, `cdemo3_rcl`, `cdemo4_rcl`, or `cdemo5_rcl`. The content of each `cdemo` file is identical to the content of the corresponding (FORTRAN95) `demo` file.

A few C++ functions differ from the usage and naming convention of the FORTRAN95 subroutines owing to the conceptual difference of optional arguments and arrays in FORTRAN95 and C++. Thus, the functions

- `use_gfermi_scheme_rcl` (Section 4.2.18 in Ref. [14]),
- `set_gs_power_rcl` (Section 4.3.2 in Ref. [14]),

are replaced by the following ones:

4.5.1. `use_gfermi_scheme_rcl`

This C++ function takes no arguments and calls the subroutine `use_gfermi_scheme_rcl(g,a)` (FORTRAN95) neither setting a value for `g` nor `a`. This corresponds to selecting the G_F scheme as renormalization scheme for the EW coupling and using the default value for the Fermi constant G_F which is hard-coded in RECOLA.

4.5.2. `use_gfermi_scheme_and_set_gfermi_rcl(g)`

This C++ function calls the subroutine `use_gfermi_scheme_rcl(g,a)` (FORTRAN95), passing the value g of type `double`.

4.5.3. `use_gfermi_scheme_and_set_alpha_rcl(a)`

This C++ function calls the subroutine `use_gfermi_scheme_rcl(g,a)` (FORTRAN95), passing the value a of type `double`.

4.5.4. `set_gs_power_rcl(npr,gsarray,gslen)`

This C++ function calls the subroutine `set_gs_power_rcl(npr, gsarray)` (FORTRAN95), passing the value of `npr` (of type `int`) and `gsarray` (of type `int[][2]`). The value `gslen` is the length of the first index of `gsarray` which needs to be passed explicitly to FORTRAN95.

4.5.5. Missing subroutines

The subroutines `get_colour_configurations_rcl` (Section 4.5.5 in Ref. [14]) and `get_helicity_configurations_rcl` (Section 4.5.6 in Ref. [14]) are currently not included in the C++ interface. The authors are willing to provide a custom solution, upon request, to include their functionality.

4.6. PYTHON *interface*

The RECOLA2 package includes a PYTHON interface with the same naming conventions for the subroutines as given in Sections 4.2–4.6 in Ref. [14] and the new subroutines in Sections 4.1, 4.2, and 4.3. The PYTHON interface requires to build an additional library, called `pyrecola`, which is done automatically alongside the RECOLA2 library if the PYTHON libraries are found on the system and if RECOLA2 is build as a shared library. See Section 3.1.1 for more details on building `pyrecola`. The basic usage is identical to the one in FORTRAN95 and follows the computation flow presented at the beginning of Section 4. In order to use RECOLA2 in a PYTHON program the RECOLA2 library needs to be loaded as follows:

```
import pyrecola
```

This step requires the python library `pyrecola.so` to be present in the current working directory, or, alternatively, by updating the environment path `PYTHONPATH` as follows

```
export PYTHONPATH=$PYTHONPATH:<PATH_TO_PYRECOLA>
```

where `<PATH_TO_PYRECOLA>` is the absolute path to the directory containing `pyrecola.so`. Once the RECOLA2 namespace is accessible the functions are called in the typical PYTHON syntax, e.g.:

```
pyrecola.define_process_rcl(1, 'u u -> u u', 'NLO')
```

The namespace identifier "`pyrecola.`" can be omitted by importing RECOLA2 as follows:

```
from pyrecola import *
```

Instead of loading all methods via `*` specific ones can be imported, e.g.

```
from pyrecola import (define_process_rcl,
                      generate_process_rcl,
                      compute_process_rcl)
```

The FORTRAN95 subroutines translate to functions in PYTHON with the argument types being identified according to:

FORTRAN95	PYTHON
integer	int
integer, dimension(:)	list
logical	bool
real(dp)	float
real(dp), dimension(:)	list
complex(dp)	complex
character(len=*)	str

For multi-dimensional arrays, such as the momenta `p`, the conventions for the order of the indices is the same as in FORTRAN95, and any necessary transposition is performed within the interface. Note that explicit type checking is performed in the PYTHON/C API interface (`pyrecola.c`).

In contrast to the FORTRAN95 and C++ interface, computed results are returned by function calls. For example, the call

```
as = get_alphas_rcl()
```

returns the value of α_s and stores it in the variable `as`. In general, the return value is a tuple of variables. The return type of a function can be inquired from the documentation as described below or inferred from the PYTHON/C API interface `pyrecola.c`.

Furthermore, optional arguments are not implemented by operator overloading, but via keyword arguments which allow most of the PYTHON methods to be called in the same way as done in FORTRAN95 with a few exceptions discussed below. For instance, the call

```
pyrecola.use_alphaZ_scheme_rcl()
```

enables the use of the $\alpha(M_Z)$ scheme, using the default value for $\alpha(M_Z)$ which is hard-coded in RECOLA2. Providing an explicit argument as

```
pyrecola.use_alphaZ_scheme_rcl (a=0.0078125)
```

allows to use a different value for $\alpha(M_Z)$ than the default in the running session.

The PYTHON interface comes with a built-in documentation for every public method in RECOLA2 and can be accessed by calling `help` inside PYTHON on either the module `pyrecola` itself, or on specific methods. For instance, calling `help(use_alphaZ_scheme_rcl)` returns

```
>>> import pyrecola
>>> help(pyrecola.use_alphaZ_scheme_rcl)
Help on built-in function use_alphaZ_scheme_rcl in module pyrecola:

use_alphaZ_scheme_rcl(...)
    use_alphaZ_scheme_rcl(a=None)

    Sets the EW renormalization scheme to the alphaZ scheme.
    a -> Sets value of alpha to ‘a‘
    For ‘a=None‘ (DEFAULT) the hard-coded value for alpha is used.
```

The original FORTRAN95 demo files are available as PYTHON demo files. No compilation of the demo files is required and they can be run directly by executing

```
python <demofile>
```

in the demo folder, with `<demofile>` taking the values `pydemo0_rcl.py`, `pydemo1_rcl.py`, `pydemo2_rcl.py`, `pydemo3_rcl.py`, `pydemo4_rcl.py` or `pydemo5_rcl.py`. The content of each `pydemo` file is identical to the content of the corresponding (FORTRAN95) `demo` file.

A few PYTHON functions differ from the usage of the FORTRAN95 subroutines owing to the conceptual difference of optional arguments in FORTRAN95 and PYTHON. The following functions are concerned

- `get_amplitude_rcl`
  `(npr,order,colour,hel,pow=None,gs=None)`
- `get_squared_amplitude_rcl`
  `(npr,order,pow=None,als=None)`

- `get_polarized_squared_amplitude_rcl (npr,order,hel,pow=None,als=None)`
- `get_colour_correlation_rcl (npr,i1,i2,pow=None,als=None)`
- `get_spin_correlation_rcl (npr,pow=None,als=None)`
- `get_spin_colour_correlation_rcl (npr,i1,i2,pow=None,als=None)`

For all of these functions the difference is due to the new treatment of powers in RECOLA2 and the backward compatibility with RECOLA, as described in Section 4.3. The argument `pow`, which is overloaded in the FORTRAN95 interface, is replaced in PYTHON by the keyword arguments `pow` and `gs` (or `als`).[15] The functions are called in the usual way with the exception that either `pow` or `gs`/`als` is passed and labelled explicitly, e.g.:

```
A2 = pyrecola.get_squared_amplitude_rcl(..., pow=[2,4])
A2 = pyrecola.get_squared_amplitude_rcl(..., als=1)
```

4.6.1. Missing subroutines

The subroutines `get_colour_configurations_rcl` (Section 4.5.5 in Ref. [14]) and `get_helicity_configurations_rcl` (Section 4.5.6 in Ref. [14]) are currently not included in the PYTHON interface. The authors are willing to provide a custom solution, upon request, to include their functionality.

5. Conclusions

The RECOLA2 library computes amplitudes in the Standard Model of particle physics including QCD and electroweak interaction and in general quantum field theories at the tree and one-loop level with no a-priori restriction on the particle multiplicities, once corresponding model files are available. Amplitudes can be obtained for specific colour structures and helicities

[15] We refer to the subroutines in Section (4.5) in Ref. [14] for the description of the other arguments.

and squared amplitudes with or without summation/average over helicities. We provide subroutines for the computation of colour- and spin-correlated leading-order squared amplitudes that are required in the dipole subtraction formalism. Furthermore, the code supports the selection of resonant contributions allowing for the computation of factorizable corrections in pole approximations.

In this first release of RECOLA2 we include RECOLA2 model files for the computation of processes in the Two-Higgs-Doublet Model and the Higgs-singlet extension of the Standard Model. The model files are generated in the complex-mass scheme, and various renormalization schemes are supported for the electromagnetic coupling, the strong coupling and additional parameters in extended Higgs sectors.

The present version of RECOLA2 is restricted to theories with scalars, Dirac fermions and vector bosons. An enhanced version with Majorana-fermion support and further RECOLA2 model files, including anomalous couplings, is in preparation.

6. Acknowledgements

We thank B. Biedermann, M. Chiesa, R. Feger and M. Pellen for performing various checks of the code. A. D. and J.-N. L. acknowledge support from the German Research Foundation (DFG) via grants DE 623/4-1 and DE 623/5-1. The work of J.-N. L. is supported by the Studienstiftung des Deutschen Volkes. The work of S. U. was supported in part by the European Commission through the "HiggsTools" Initial Training Network PITN-GA-2012-316704. The research of A.D. and S.U. was supported in part by the Munich Institute for Astro- and Particle Physics (MIAPP) of the DFG cluster of excellence "Origin and Structure of the Universe".

Appendix A. Checks

RECOLA2 has been thoroughly tested against RECOLA, guaranteeing the consistency for all checks of Appendix B in Ref. [14] for the SM.

In the 2HDM all renormalized scalar two-point functions of the extended Higgs sector in the 2HDM have been verified off-shell against an independent approach in `QGRAF` [43] and `QGS`, which is an extension of `GraphShot` [44]. Furthermore, we have compared all partonic channels to Higgs decays

into four fermions against the independent calculation [36] based on FeynArts/FormCalc [1, 45]. Finally, all models have been tested against the BFM implementation [19].

References

[1] T. Hahn, Generating Feynman diagrams and amplitudes with FeynArts 3, Comput. Phys. Commun. 140 (2001) 418–431. arXiv:hep-ph/0012260, doi:10.1016/S0010-4655(01)00290-9.

[2] S. Agrawal, T. Hahn, E. Mirabella, FormCalc 7.5, PoS LL2012 (2012) 046. arXiv:1210.2628.

[3] C. Berger, et al., An automated implementation of on-shell methods for one-loop amplitudes, Phys. Rev. D78 (2008) 036003. arXiv:0803.4180, doi:10.1103/PhysRevD.78.036003.

[4] A. van Hameren, C. Papadopoulos, R. Pittau, Automated one-loop calculations: A proof of concept, JHEP 0909 (2009) 106. arXiv:0903.4665, doi:10.1088/1126-6708/2009/09/106.

[5] G. Cullen, et al., Automated one-loop calculations with GoSam, Eur. Phys. J. C72 (2012) 1889. arXiv:1111.2034, doi:10.1140/epjc/s10052-012-1889-1.

[6] S. Badger, B. Biedermann, P. Uwer, NGluon: a package to calculate one-loop multi-gluon amplitudes, Comput. Phys. Commun. 182 (2011) 1674–1692. arXiv:1011.2900, doi:10.1016/j.cpc.2011.04.008.

[7] S. Badger, B. Biedermann, P. Uwer, V. Yundin, Numerical evaluation of virtual corrections to multi-jet production in massless QCD, Comput. Phys. Commun. 184 (2013) 1981–1998. arXiv:1209.0100, doi:10.1016/j.cpc.2013.03.018.

[8] J. Alwall, et al., The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations, JHEP 07 (2014) 079. arXiv:1405.0301, doi:10.1007/JHEP07(2014)079.

[9] F. Cascioli, P. Maierhöfer, S. Pozzorini, Scattering amplitudes with Open Loops, Phys. Rev. Lett. 108 (2012) 111601. arXiv:1111.5206, doi:10.1103/PhysRevLett.108.111601.

[10] S. Actis, A. Denner, L. Hofer, A. Scharf, S. Uccirati, Recursive generation of one-loop amplitudes in the Standard Model, JHEP 1304 (2013) 037. arXiv:1211.6316, doi:10.1007/JHEP04(2013)037.

[11] S. Kallweit, J. M. Lindert, P. Maierhöfer, S. Pozzorini, M. Schönherr, NLO electroweak automation and precise predictions for W+multijet production at the LHC, JHEP 04 (2015) 012. arXiv:1412.5157, doi:10.1007/JHEP04(2015)012.

[12] S. Frixione, V. Hirschi, D. Pagani, H. S. Shao, M. Zaro, Electroweak and QCD corrections to top-pair hadroproduction in association with heavy bosons, JHEP 06 (2015) 184. arXiv:1504.03446, doi:10.1007/JHEP06(2015)184.

[13] M. Chiesa, N. Greiner, F. Tramontano, Automation of electroweak corrections for LHC processes, J. Phys. G43 (1) (2016) 013002. arXiv:1507.08579, doi:10.1088/0954-3899/43/1/013002.

[14] S. Actis, et al., RECOLA: REcursive Computation of One-Loop Amplitudes, Comput. Phys. Commun. 214 (2017) 140–173. arXiv:1605.01090, doi:10.1016/j.cpc.2017.01.004.

[15] N. D. Christensen, C. Duhr, FeynRules - Feynman rules made easy, Comput. Phys. Commun. 180 (2009) 1614–1641. arXiv:0806.4194, doi:10.1016/j.cpc.2009.02.018.

[16] N. D. Christensen, et al., A Comprehensive approach to new physics simulations, Eur. Phys. J. C71 (2011) 1541. arXiv:0906.2474, doi:10.1140/epjc/s10052-011-1541-5.

[17] F. Staub, SARAH 4: A tool for (not only SUSY) model builders, Comput. Phys. Commun. 185 (2014) 1773–1790. arXiv:1309.7223, doi:10.1016/j.cpc.2014.02.018.

[18] C. Degrande, Automatic evaluation of UV and R2 terms for beyond the Standard Model Lagrangians: a proof-of-principle, Comput. Phys.

Commun. 197 (2015) 239–262. arXiv:1406.3030, doi:10.1016/j.cpc.2015.08.015.

[19] A. Denner, J.-N. Lang, S. Uccirati, NLO electroweak corrections in extended Higgs Sectors with RECOLA2, JHEP 07 (2017) 087. arXiv:1705.06053, doi:10.1007/JHEP07(2017)087.

[20] J. A. M. Vermaseren, New features of FORM (2000). arXiv:math-ph/0010025.

[21] B. Ruijl, T. Ueda, J. Vermaseren, FORM version 4.2 (2017). arXiv:1707.06453.

[22] C. Degrande, et al., UFO - The Universal FeynRules Output, Comput. Phys. Commun. 183 (2012) 1201–1214. arXiv:1108.2040, doi:10.1016/j.cpc.2012.01.022.

[23] F. J. Dyson, The S matrix in quantum electrodynamics, Phys. Rev. 75 (1949) 1736–1755. doi:10.1103/PhysRev.75.1736.

[24] J. S. Schwinger, On the Green's functions of quantized fields. 1., Proc. Nat. Acad. Sci. 37 (1951) 452–455. doi:10.1073/pnas.37.7.452.

[25] J. S. Schwinger, On the Green's functions of quantized fields. 2., Proc. Nat. Acad. Sci. 37 (1951) 455–459. doi:10.1073/pnas.37.7.455.

[26] A. Denner, S. Dittmaier, L. Hofer, COLLIER: a fortran-based Complex One-Loop LIbrary in Extended Regularizations, Comput. Phys. Commun. 212 (2017) 220–238. arXiv:1604.06792, doi:10.1016/j.cpc.2016.10.013.

[27] J. F. Gunion, H. E. Haber, The CP conserving Two-Higgs-Doublet Model: the approach to the decoupling limit, Phys. Rev. D67 (2003) 075019. arXiv:hep-ph/0207010, doi:10.1103/PhysRevD.67.075019.

[28] A. Denner, L. Jenniches, J.-N. Lang, C. Sturm, Gauge-independent $\overline{MS}$ renormalization in the 2HDM, JHEP 09 (2016) 115. arXiv:1607.07352, doi:10.1007/JHEP09(2016)115.

[29] A. Denner, T. Sack, Renormalization of the Quark Mixing Matrix, Nucl. Phys. B347 (1990) 203–216. doi:10.1016/0550-3213(90)90557-T.

[30] J. R. Espinosa, I. Navarro, Scale independent mixing angles, Phys. Rev. D66 (2002) 016004. arXiv:hep-ph/0109126, doi:10.1103/PhysRevD.66.016004.

[31] J. R. Espinosa, Y. Yamada, Scale independent and gauge independent mixing angles for scalar particles, Phys. Rev. D67 (2003) 036003. arXiv:hep-ph/0207351, doi:10.1103/PhysRevD.67.036003.

[32] A. Freitas, D. Stöckinger, Gauge dependence and renormalization of $\tan\beta$ in the MSSM, Phys. Rev. D66 (2002) 095014. arXiv:hep-ph/0205281, doi:10.1103/PhysRevD.66.095014.

[33] M. Sperling, D. Stöckinger, A. Voigt, Renormalization of vacuum expectation values in spontaneously broken gauge theories, JHEP 07 (2013) 132. arXiv:1305.1548, doi:10.1007/JHEP07(2013)132.

[34] F. Bojarski, G. Chalons, D. Lopez-Val, T. Robens, Heavy to light Higgs boson decays at NLO in the Singlet Extension of the Standard Model, JHEP 02 (2016) 147. arXiv:1511.08120, doi:10.1007/JHEP02(2016)147.

[35] M. Krause, R. Lorenz, M. Mühlleitner, R. Santos, H. Ziesche, Gauge-independent Renormalization of the 2-Higgs-Doublet Model, JHEP 09 (2016) 143. arXiv:1605.04853, doi:10.1007/JHEP09(2016)143.

[36] L. Altenkamp, S. Dittmaier, H. Rzehak, Renormalization schemes for the Two-Higgs-Doublet Model and applications to h $\to$ WW/ZZ $\to$ 4fermions (2017). arXiv:1704.02645.

[37] A. Denner, Techniques for calculation of electroweak radiative corrections at the one-loop level and results for W physics at LEP-200, Fortsch. Phys. 41 (1993) 307–420. arXiv:0709.1075, doi:10.1002/prop.2190410402.

[38] J. Fleischer, F. Jegerlehner, Radiative corrections to Higgs decays in the extended Weinberg-Salam Model, Phys. Rev. D23 (1981) 2001–2026. doi:10.1103/PhysRevD.23.2001.

[39] S. Actis, A. Ferroglia, M. Passera, G. Passarino, Two-loop renormalization in the Standard Model. Part I: Prolegomena, Nucl. Phys. B777

(2007) 1–34. arXiv:hep-ph/0612122, doi:10.1016/j.nuclphysb.2007.04.021.

[40] A. Denner, G. Weiglein, S. Dittmaier, Application of the background field method to the electroweak standard model, Nucl. Phys. B440 (1995) 95–128. arXiv:hep-ph/9410338, doi:10.1016/0550-3213(95)00037-S.

[41] A. Denner, S. Dittmaier, L. Hofer, COLLIER - A fortran library for one-loop integrals, PoS LL2014 (2014) 071. arXiv:1407.0087.

[42] A. Denner, S. Dittmaier, Scalar one-loop 4-point integrals, Nucl. Phys. B844 (2011) 199–242. arXiv:1005.2076, doi:10.1016/j.nuclphysb.2010.11.002.

[43] P. Nogueira, Automatic Feynman graph generation, J. Comput. Phys. 105 (1993) 279–289. doi:10.1006/jcph.1993.1074.

[44] S. Actis, A. Ferroglia, G. Passarino, M. Passera, Ch. Sturm and S. Uccirati, GraphShot, a Form package for automatic generation and manipulation of one- and two-loop Feynman diagrams, unpublished.

[45] C. Groß, et al., New Developments in FormCalc 8.4, PoS LL2014. arXiv:1407.0235.