R E C O L A

REcursive Computation of One-Loop Amplitudes [☆]

Version 1.3

Stefano Actis[a], Ansgar Denner[b], Lars Hofer[c], Jean-Nicolas Lang[b], Andreas Scharf[b], Sandro Uccirati[d]

[a] *Lilienstrasse 7, 5200 Brugg AG, Switzerland*
[b] *Universität Würzburg, Institut für Theoretische Physik und Astrophysik, D-97074 Würzburg, Germany*
[c] *Department de Física Quàntica i Astrofísica (FQA), Institut de Ciències del Cosmos (ICCUB), Universitat de Barcelona (UB), Martí Franquès 1, E-08028 Barcelona, Spain*
[d] *Università di Torino e INFN, 10125 Torino, Italy*

Abstract

We present the Fortran95 program Recola for the perturbative computation of next-to-leading-order transition amplitudes in the Standard Model of particle physics. The code provides numerical results in the 't Hooft–Feynman gauge. It uses the complex-mass scheme and allows for a consistent isolation of resonant contributions. Dimensional regularization is employed for ultraviolet and infrared singularities, with the alternative possibility of treating collinear and soft singularities in mass regularization. Recola supports various renormalization schemes for the electromagnetic and a dynamical N_{f}-flavour scheme for the strong coupling constant. The calculation of next-to-leading-order squared amplitudes, summed over spin and colour, is supported as well as the computation of colour- and spin-correlated leading-order squared amplitudes needed in the dipole subtraction formalism.

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

Email addresses: stefano.actis@gmail.com (Stefano Actis), ansgar.denner@physik.uni-wuerzburg.de (Ansgar Denner), hofer@ecm.ub.edu (Lars Hofer), jlang@physik.uni-wuerzburg.de (Jean-Nicolas Lang), ascharf@physik.uni-wuerzburg.de (Andreas Scharf), uccirati@to.infn.it (Sandro Uccirati)

November 7, 2023

Contents

1 Introduction **6**

2 Basic features of RECOLA **8**
2.1 Collinear and soft singularities 9
2.2 Dimensional regularization 9
2.3 The strong coupling constant α_s 10
2.4 Electroweak renormalization 11
2.5 Amplitude structure 12
2.6 Squared amplitudes 14
2.7 Colour- and spin-correlated squared amplitudes 15
2.8 Selecting intermediate states and resonances 17
2.9 Conventions 19

3 Installation **20**
3.1 The RECOLA–COLLIER package 20
3.2 The RECOLA package 22

4 Usage of RECOLA **25**
4.1 Input variables 27
4.1.1 Pole masses and widths of the SM particles 27
4.1.2 Parameters governing the treatment of collinear singularities 28
4.1.3 Parameters governing the treatment of soft singularities 29
4.1.4 Dimensional regularization parameters 30
4.1.5 Parameters for the renormalization of the QCD coupling 30
4.1.6 Parameters for the renormalization of the EW coupling 31
4.1.7 Parameter for the renormalization of masses of unstable particles 31
4.1.8 Parameter for self-energies of resonant particles 32
4.1.9 Parameter for dynamic settings of parameters 32
4.1.10 Parameter for the automatic correction of external momenta 33
4.1.11 Options for the visualization of the off-shell currents . . 34
4.1.12 Output options 36
4.2 Input subroutines 45
4.2.1 `set_pole_mass_`*prtcl*`_rcl (m,g)` 45

4.2.2 set_pole_mass_*prtcl*_rcl (m) 45
4.2.3 set_onshell_mass_*v*_rcl (m,g) 45
4.2.4 set_light_fermions_rcl (m) 46
4.2.5 set_light_*ferm*_rcl 46
4.2.6 use_dim_reg_soft_rcl 46
4.2.7 use_mass_reg_soft_rcl (m) 46
4.2.8 set_mass_reg_soft_rcl (m) 46
4.2.9 set_delta_uv_rcl (d) 47
4.2.10 set_mu_uv_rcl (m) 47
4.2.11 set_delta_ir_rcl (d,d2) 47
4.2.12 set_mu_ir_rcl (m) 47
4.2.13 set_alphas_rcl (a,s,Nf) 47
4.2.14 get_alphas_rcl (a) 47
4.2.15 get_renormalization_scale_rcl (mu) 47
4.2.16 get_flavour_scheme_rcl (Nf) 48
4.2.17 set_alphas_masses_rcl (mc,mb,mt,gc,gb,gt) 48
4.2.18 use_gfermi_scheme_rcl (g,a,massesgf) 48
4.2.19 use_alpha0_scheme_rcl (a) 49
4.2.20 use_alphaz_scheme_rcl (a) 49
4.2.21 get_alpha_rcl (a) 49
4.2.22 set_complex_mass_scheme_rcl 49
4.2.23 set_on_shell_scheme_rcl 49
4.2.24 set_resonant_particle_rcl (pa) 49
4.2.25 switchon_resonant_selfenergies_rcl 50
4.2.26 switchoff_resonant_selfenergies_rcl 50
4.2.27 set_dynamic_settings_rcl (n) 50
4.2.28 set_momenta_correction_rcl (mc) 50
4.2.29 set_draw_level_branches_rcl (n) 50
4.2.30 set_print_level_amplitude_rcl (n) 50
4.2.31 set_print_level_squared_amplitude_rcl (n) 51
4.2.32 set_print_level_correlations_rcl (n) 51
4.2.33 set_print_level_RAM_rcl (n) 51
4.2.34 scale_coupling3_rcl (fac,pa1,pa2,pa3) 51
4.2.35 scale_coupling4_rcl (fac,pa1,pa2,pa3,pa4) 51
4.2.36 switchoff_coupling3_rcl (pa1,pa2,pa3) 51
4.2.37 switchoff_coupling4_rcl (pa1,pa2,pa3,pa4) 52
4.2.38 set_ifail_rcl (i) 52
4.2.39 get_ifail_rcl (i) 52

4.2.40 set_collier_output_dir_rcl (dir) 52
4.2.41 set_output_file_rcl (x) 53
4.3 Process definition 53
4.3.1 define_process_rcl (npr,processIn,order) 53
4.3.2 set_gs_power_rcl (npr,gsarray) 55
4.3.3 select_gs_power_BornAmpl_rcl (npr,gspower) . . . 55
4.3.4 select_gs_power_LoopAmpl_rcl (npr,gspower) . . . 55
4.3.5 select_all_gs_powers_BornAmpl_rcl (npr) 56
4.3.6 select_all_gs_powers_LoopAmpl_rcl (npr) 56
4.3.7 split_collier_cache_rcl (npr,n) 56
4.4 Process generation: generate_processes_rcl 56
4.5 Process computation 57
4.5.1 set_resonant_squared_momentum_rcl (npr,res,ps) . 58
4.5.2 compute_running_alphas_rcl (Q,Nf,lp) 59
4.5.3 compute_process_rcl (npr,p,order,A2,momenta_check) 59
4.5.4 rescale_process_rcl (npr,order,A2) 60
4.5.5 get_colour_configurations_rcl (npr,cols) 61
4.5.6 get_helicity_configurations_rcl (npr,hels) . . . 61
4.5.7 get_amplitude_rcl (npr,pow,order,colour,hel,A) 62
4.5.8 get_squared_amplitude_rcl (npr,pow,order,A2) . . 63
4.5.9 get_polarized_squared_amplitude_rcl (npr,pow,order,hel,A2h) 64
4.5.10 compute_colour_correlation_rcl (npr,p,i1,i2,A2cc,momenta_check) 64
4.5.11 compute_all_colour_correlations_rcl (npr,p,momenta_check) 65
4.5.12 rescale_colour_correlation_rcl (npr,i1,i2,A2cc) 65
4.5.13 rescale_all_colour_correlations_rcl (npr) 66
4.5.14 get_colour_correlation_rcl (npr,pow,i1,i2,A2cc) 66
4.5.15 compute_spin_correlation_rcl (npr,p,j,v,A2sc,momenta_check) 67
4.5.16 rescale_spin_correlation_rcl (npr,j,v,A2sc) . . . 67
4.5.17 get_spin_correlation_rcl (npr,pow,A2sc) 68
4.5.18 compute_spin_colour_correlation_rcl (npr,p,i1,i2,v,A2scc,momenta_check
4.5.19 rescale_spin_colour_correlation_rcl (npr,i1,i2,v,A2scc) 69
4.5.20 get_spin_colour_correlation_rcl (npr,pow,i1,i2,A2scc) 70
4.5.21 get_momenta_rcl (npr,p) 70
4.5.22 set_TIs_required_accuracy_rcl (acc) 70
4.5.23 get_TIs_required_accuracy_rcl (acc) 70
4.5.24 set_TIs_critical_accuracy_rcl (acc) 70

4.5.25 get_TIs_critical_accuracy_rcl (acc) 71
4.5.26 get_TIs_accuracy_flag_rcl (flag) 71
4.6 Warning summary: print_warning_summary_rcl 71
4.7 Reset: reset_recola_rcl 71
4.8 C++ interface 72
4.8.1 use_gfermi_scheme_rcl 73
4.8.2 use_gfermi_scheme_and_set_alpha_rcl(a) 74
4.8.3 use_gfermi_scheme_and_set_gfermi_rcl(g) 74
4.8.4 use_gfermi_scheme_complex_and_set_gfermi_rcl(g) . 74
4.8.5 use_gfermi_scheme_real_and_set_gfermi_rcl(g) . . . 74
4.8.6 set_gs_power_rcl(npr,gsarray,gslen) 74
4.8.7 Missing subroutines 74

5 Conclusions **74**

6 Acknowledgements **75**

Appendix A Explicit representations for spinors and polarization vectors **75**

Appendix B Checks **77**

1. Introduction

The experimental studies at present and future high-energy colliders are focused on the precise determination of the free parameters of the Standard Model (SM) and on the search for new physics. The interpretation of the data often relies on accurate theoretical predictions based on perturbation theory, requiring detailed calculations beyond the leading-order (LO) approximation. In the past years, many groups have concentrated their efforts on next-to-leading-order (NLO) calculations (see e.g. Refs. [1–5]), and alternative strategies to the traditional Feynman-diagrammatic approach have been developed, which helped to automatize and speed up the calculation of NLO amplitudes. One class of methods makes use of generalized unitarity relations or of amplitude reduction at the integrand level in order to directly express one-loop amplitudes in terms of scalar integrals [6–13]. Other methods instead rely on higher-rank tensor integrals, either via an improved diagrammatic approach [14] or employing one-loop recursion relations [15, 16]. Finally, yet another strategy consists in performing a simultaneous numerical integration over the phase space and the loop momentum of NLO amplitudes [17–19].

The traditional as well as the new techniques for the calculation of one-loop amplitudes have been implemented in many one-loop amplitude generators such as FeynArts/FormCalc [20–22], Blackhat [23], Helac-1loop [13], NGluon [24], NJet [25], Madloop [26], GoSam [27], and OpenLoops [14] . In this article we present the Fortran95 library Recola for the generation of tree-level and one-loop amplitudes in the SM. While almost all of the above-listed programs were developed with a focus on QCD corrections[1], Recola has been designed from the beginning with the main objective of facilitating an automated calculation of electroweak (EW) corrections. Recently, EW corrections have been included also in OpenLoops [28, 29], MadGraph5_aMC@NLO [30, 31], and GoSam [32]. Recola further differs from other public codes in the implemented method as it makes consequent use of a recursive construction of one-loop off-shell currents following the technique described in Ref. [16]. It has successfully applied for the calculation of EW corrections to the processes pp $\to 2\ell+\leq$2j

[1] FeynArts/FormCalc allows to perform calculations in more general scenarios in and beyond the SM, though with less emphasis on high multiplicities and CPU performance.

[33], $pp \to \nu_\mu \mu^+ \bar{\nu}_e e^-, \mu^+ \mu^- e^+ e^-, \mu^+ \mu^- \mu^+ \mu^-$ [34–36], $pp \to e^+ \nu_e \mu^- \bar{\nu}_\mu b\bar{b}$ [37], and $pp \to \mu^+ \nu_\mu e^+ \nu_e jj$ [38], and for the calculation of QCD corrections to the process $pp \to WWb\bar{b}H$ [39].

For the reduction of the amplitudes and the evaluation of the one-loop integrals, a task that demands high standards with respect to numerical stability and CPU performance, amplitude generators are either equipped with own internal implementations, or they rely on external libraries. These include libraries for amplitude reduction at the integrand level such as CutTools [40], Samurai [41], and Ninja [42], libraries for tensor-integral reduction like FF [43], LoopTools [44], PJFry [45], Golem95C [46], Collier [47, 48], and Package-X [49], as well as packages for the evaluation of scalar integrals like QCDLoop [50] and OneLOop [51],

In the case of Recola, the public Fortran95 library Collier is used which achieves a fast and stable calculation of tensor integrals via the strategies developed in Refs. [52–54].

The available one-loop generators do not only differ in the class of computations they can perform, but also in the level of automation and in the cost of performance (speed and memory). The latter is an essential aspect because typical Monte-Carlo simulations require a huge number of evaluations of the matrix element for each partonic process in order to obtain a sufficient statistical accuracy. To this end, in the development of Recola a big effort has been invested in the optimization of the performance in order to permit the fast "on-the-fly" generation and evaluation of NLO matrix elements. This strategy is complementary to the one used by other groups, as for example the Blackhat collaboration, who have developed a flexible storage format of pre-calculated matrix elements for partonic events in large Root N-tuple files [55], which are then read by the Monte Carlo generator.

This article is organized as follows: In Section 2 we describe the basic features of Recola; Section 3 gives the user the necessary information on how to download and install the Recola library. Section 4 explains the usage of Recola and gives a detailed description of its input parameters and of all subroutines that can be called by the user, and Section 5 contains the conclusions. Finally, in Appendix A we provide the explicit representations for spinors and polarization vectors used in Recola, and in Appendix B we list the processes that have been checked against other programs.

2. Basic features of RECOLA

RECOLA is a FORTRAN95 code for the computation of tree-level and one-loop scattering amplitudes in the SM, based on recursion relations [16].

The algorithm to compute the tree-level amplitude $\mathcal{A}_0$ is inspired by the Dyson–Schwinger equations [56–58]. The recursion relations for the one-loop amplitudes are more involved and rely on the decomposition of the one-loop amplitude $\mathcal{A}_1$ in terms of tensor integrals (TIs) $T_{(t)}^{\mu_1\cdots\mu_{r_t}}$ and tensor coefficients (TCs) $c_{\mu_1\cdots\mu_{r_t}}^{(t)}$:

$$\mathcal{A}_1 = \sum_t c_{\mu_1\cdots\mu_{r_t}}^{(t)} T_{(t)}^{\mu_1\cdots\mu_{r_t}} + \mathcal{A}_{\mathrm{CT}}\,. \tag{1}$$

Here, $\mathcal{A}_{\mathrm{CT}}$ is the contribution from the counterterms. In order to regularize ultraviolet (UV) singularities the TIs are treated in dimensional regularization by introducing the variable space-time dimension $\mathrm{D} = 4 - 2\epsilon$ together with the mass scale μ:

$$T_{(t)}^{\mu_1\cdots\mu_{r_t}} = \frac{(2\pi\mu)^{4-\mathrm{D}}}{\mathrm{i}\pi^2} \int \mathrm{d}^{\mathrm{D}}q\, \frac{q^{\mu_1}\cdots q^{\mu_{r_t}}}{D_0^{(t)}\cdots D_{k_t}^{(t)}}. \tag{2}$$

Here, k_t is the number of propagators in the loop, r_t the rank of $T_{(t)}$ and

$$D_i^{(t)} = (q + p_i^{(t)})^2 - (m_i^{(t)})^2, \qquad i = 0, \ldots, k_t, \qquad p_0^{(t)} = 0, \tag{3}$$

where $p_i^{(t)}$ is the momentum flowing in the i^{th} propagator. UV singularities of the TIs manifest themselves as poles in ϵ, and they are cancelled by analogous singularities present in the counterterm amplitude $\mathcal{A}_{\mathrm{CT}}$, which can be built from tree-level topologies involving counterterm vertices [59].

Based on an idea by van Hameren [15], a recursive procedure to compute the TCs numerically has been developed [16] and implemented in RECOLA. In this framework the indices $\mu_1, \ldots, \mu_{r_t}$ are taken strictly 4-dimensional (with values $0, 1, 2, 3$), and the $(D-4)$-dimensional part of the contraction between the TCs and TIs in $\mathcal{A}_1$ is taken into account in the form of an additional rational part $\mathcal{A}_{\mathrm{R2}}$ (of type R_2 [60]):

$$\mathcal{A}_1 = \mathcal{A}_{\mathrm{D4}} + \mathcal{A}_{\mathrm{R2}} + \mathcal{A}_{\mathrm{CT}}\,, \qquad \mathcal{A}_{\mathrm{D4}} = \sum_t c_{\hat{\mu}_1\cdots\hat{\mu}_{r_t}}^{(t)} T_{(t)}^{\hat{\mu}_1\cdots\hat{\mu}_{r_t}}\,. \tag{4}$$

The hat on the indices $\mu_1, \ldots, \mu_{r_t}$ indicates that they run over the 4 dimensions $\hat{\mu}_i = 0, 1, 2, 3$. The tensor integrals $T_{(t)}^{\hat{\mu}_1\cdots\hat{\mu}_{r_t}}$ are computed in RECOLA

by means of an interface with the COLLIER library [47]. The contribution $\mathcal{A}_{\rm R2}$ is determined evaluating tree-level-like topologies with special Feynman rules [60–63] similar to those for the counterterms.

2.1. Collinear and soft singularities

Collinear singularities originating from light fermions can be treated in RECOLA either in dimensional or in mass regularization. For each fermion an individual choice can be made: If a fermion is defined as massless, collinear divergences stemming from this fermion are regularized dimensionally. If on the other hand it has been assigned a (regulator) mass, its collinear singularities are regularized by the corresponding mass parameter.

Soft singularities are either regularized dimensionally or by assigning a mass regulator λ to photons and gluons. The second case is allowed in RECOLA only if collinear singularities are treated in mass regularization for all fermions.

2.2. Dimensional regularization

If dimensional regularization is used for collinear or soft singularities, poles in ϵ of infrared (IR) origin are generated in $\mathcal{A}_1$ together with a dependence on the scale μ. In order to distinguish this ϵ and μ dependence of IR origin from the one of UV origin, RECOLA introduces the separate parameters $\mu_{\rm UV}$, $\epsilon_{\rm UV}$ and $\mu_{\rm IR}$, $\epsilon_{\rm IR}$ in all TIs and counterterms, together with

$$\begin{aligned} \Delta_{\rm UV} &= \frac{(4\pi)^{\epsilon_{\rm UV}}\,\Gamma(1+\epsilon_{\rm UV})}{\epsilon_{\rm UV}}, \\ \Delta_{\rm IR} &= \frac{(4\pi)^{\epsilon_{\rm IR}}\,\Gamma(1+\epsilon_{\rm IR})}{\epsilon_{\rm IR}}, \qquad \Delta_{\rm IR2} = \frac{(4\pi)^{\epsilon_{\rm IR}}\,\Gamma(1+\epsilon_{\rm IR})}{\epsilon_{\rm IR}^2}. \end{aligned} \tag{5}$$

Following the conventions of COLLIER [47, 48] and Ref. [54], the parameters $\Delta_{\rm UV}$, $\Delta_{\rm IR}$ and $\Delta_{\rm IR2}$ 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 takes the general form

$$\mathcal{A}_1 = \Delta_{\rm UV}\,\mathcal{A}_1^{\rm UV} + \Delta_{\rm IR2}\,\mathcal{A}_1^{\rm IR2} + \Delta_{\rm IR}\,\mathcal{A}_1^{\rm IR}(\mu_{\rm IR}) + \mathcal{A}_1^{\rm fin}(\mu_{\rm UV},\mu_{\rm IR}). \tag{6}$$

The term $\mathcal{A}_1^{\rm UV}$ vanishes after renormalization. The a priori unphysical scale $\mu_{\rm UV}$ should either cancel between the TIs and the counterterms, or it should receive a physical interpretation as for example when it is identified with the renormalization scale Q in the $\overline{\rm MS}$ scheme for the strong coupling constant g_s.

In RECOLA, however, the $\overline{\text{MS}}$ renormalization scale Q is kept independent from μ_{UV}. In this way, also $\overline{\text{MS}}$-renormalized amplitudes are independent of μ_{UV} but depend on Q instead.

The counterterm δZ_{g_s}, relating the bare and the renormalized strong coupling constants g_s^0 and g_s according to $g_s^0 = g_s\,(1 + \delta Z_{g_s})$ is defined as

$$\delta Z_{g_s} = -\frac{\alpha_s(Q^2)}{4\pi}\left[\left(\frac{11}{2} - \frac{N_{\text{f}}}{3}\right)\left(\Delta_{\text{UV}} + \ln\frac{\mu_{\text{UV}}^2}{Q^2}\right) - \frac{1}{3}\sum_F \left(\Delta_{\text{UV}} + \ln\frac{\mu_{\text{UV}}^2}{|m_F|^2}\right)\right], \tag{7}$$

where N_{f} is the number of active (light) flavours and F runs over the inactive (heavy) flavours. According to Eq. (7), the contribution from active flavours is renormalized within the $\overline{\text{MS}}$ scheme, while the one from inactive flavours is subtracted at zero momentum transfer. The classification into *active* and *inactive* flavours defines the flavour scheme. In RECOLA the user can choose between:

- The *variable-flavour scheme*: All quark flavours lighter than Q are considered as active, the remaining ones are treated as inactive.
- The N_{f}*-flavour scheme*: The N_{f} lightest quarks are considered active, the remaining ones are treated as inactive. In this case, N_{f} cannot be chosen lower than the number of massless quarks.

After renormalization, the one-loop amplitude becomes

$$\mathcal{A}_1 = \Delta_{\text{IR2}}\,\mathcal{A}_1^{\text{IR2}} + \Delta_{\text{IR}}\,\mathcal{A}_1^{\text{IR}}(\mu_{\text{IR}}) + \mathcal{A}_1^{\text{fin}}(Q, \mu_{\text{IR}}). \tag{8}$$

RECOLA performs a numerical computation of the complete amplitude $\mathcal{A}_1$, with the values for Δ_{IR}, Δ_{IR2}, μ_{IR} and Q supplied by the user. The contribution $\mathcal{A}_1^{\text{fin}}$ can be obtained by setting $\Delta_{\text{IR}} = \Delta_{\text{IR2}} = 0$ (default). Since the computation of $\mathcal{A}_1$ involves objects depending on Δ_{UV} and μ_{UV} at intermediate steps, numerical values for these variables must be given as well. The independence of $\mathcal{A}_1$ on Δ_{UV} and μ_{UV} can be verified numerically by varying these parameters. RECOLA provides default values for all the above-mentioned parameters that can be changed by the user.

2.3. The strong coupling constant α_s

The renormalized strong coupling constant α_s depends on the renormalization scale Q. It is appropriate to choose a value for Q of the order of the

energy scale characteristic for the process in question and to take as coupling constant the corresponding value for $\alpha_s(Q^2)$.

Often, in the computation of physical processes the scale Q is defined from the momenta of the external particles and is thus assigned a different value for each phase-space point. The possibility to use such a dynamical scale Q is supported by RECOLA, and the respective values for the running $\alpha_s(Q^2)$ can either be supplied by the user or computed by RECOLA. In the latter case, the value for $\alpha_s(Q^2)$ at the scale Q is determined from its value $\alpha_s(Q_0^2)$ at the initialization scale Q_0, by means of the following one- and two-loop formulas in the N_f-flavour scheme [64]:

$$\text{1-loop running:} \qquad a = \frac{a_0}{1 + a_0\,\beta_0\,L}, \tag{9}$$

$$\text{2-loop running:} \qquad a = a_0 \left[1 + a_0\,\beta_0\,L + a_0\,b_1 \ln\left(1 + \frac{a_0\,\beta_0\,L}{1 + a_0\,b_1}\right)\right]^{-1}. \tag{10}$$

Here, we have introduced

$$a = \frac{\alpha_s(Q^2)}{4\pi}, \qquad a_0 = \frac{\alpha_s(Q_0^2)}{4\pi}, \qquad L = \ln\frac{Q^2}{Q_0^2}, \tag{11}$$

$$b_1 = \frac{\beta_1}{\beta_0}, \qquad \beta_0 = 11 - \frac{2}{3}N_\mathrm{f}, \qquad \beta_1 = 102 - \frac{38}{3}N_\mathrm{f}. \tag{12}$$

2.4. Electroweak renormalization

Due to the presence of massive and unstable gauge bosons in the EW sector of the SM, complex masses have to be introduced without spoiling gauge invariance. This is achieved in RECOLA by using the complex-mass scheme of Refs. [52, 53, 65]. The user can, however, also choose to proceed in the on-shell scheme, where only the real part of the self-energies is taken for the computation of the counterterms and the imaginary part of the masses is kept only in the denominator of propagators.

For the renormalization of the EW coupling constant α, the user can choose between three different schemes:

- G_F scheme:
  In this scheme the renormalized electromagnetic coupling α is derived from the Fermi constant G_F, measured in muon decay, and the masses of the W and Z bosons via the tree-level relation:

$$\alpha = \frac{\sqrt{2}\,G_\mathrm{F}}{\pi}\,\mathrm{Re}(M_\mathrm{W}^2)\,\left(1 - \frac{\mathrm{Re}(M_\mathrm{W}^2)}{\mathrm{Re}(M_\mathrm{Z}^2)}\right). \tag{13}$$

- $\alpha(0)$ scheme:
  In this scheme α is fixed from the value measured in Thomson scattering at $p^2 = 0$.

- $\alpha(M_{\rm Z})$ scheme:
  In this scheme α is renormalized at the Z pole, thus implicitly taking into account its running from $p^2 = 0$ to $p^2 = M_{\rm Z}^2$.

2.5. Amplitude structure

In general, the amplitude of a process depends on helicities and colours of external particles. For a process with l external legs we introduce the notation

$$\mathcal{A} \to \mathcal{A}_{a_1\cdots a_l}[h_1, \ldots, h_l]. \tag{14}$$

The variable h_n $(n = 1, \ldots, l)$ defines the helicity of the $n^{\rm th}$ external particle and can take the values $+$ or $-$ for fermions and massless vector bosons, and the values $+$, $-$ or 0 for massive vector bosons (see Section 2.9 for the conventions used in RECOLA). For scalar particles it necessarily assumes the value 0. The variable a_n $(n = 1, \ldots, l)$ defines the colour of the $n^{\rm th}$ external particle and can take the values $1, 2, 3$ for quarks and anti-quarks, and the values $1, \ldots, 8$ for gluons, while it is absent for colourless particles.

According to the SU(3) decomposition $3 \otimes \bar{3} = 8 \oplus 1$, the colour-octet representation of the gluon can be related to the product of the fundamental and anti-fundamental representation of quarks and anti-quarks. The colour content of the amplitude can thus alternatively be expressed in the so-called colour-flow representation, where the colour of the gluons is represented by means of a pair of indices taking the values $1, 2, 3$ (see Refs. [16, 66, 67] for details). Following this approach we write the amplitude as

$$\mathcal{A} \to \mathcal{A}^{i_1\cdots i_l}_{j_1\cdots j_l}[h_1, \ldots, h_l]. \tag{15}$$

The colour index i_n $(n = 1, \ldots, l)$ is absent for colourless particles, incoming quarks and outgoing anti-quarks, while it takes the values $1, 2, 3$ for gluons, incoming anti-quarks and outgoing quarks. Similarly, the anti-colour index j_n $(n = 1, \ldots, l)$ is absent for colourless particles, incoming anti-quarks and outgoing quarks, while it takes the values $1, 2, 3$ for gluons, incoming quarks and outgoing anti-quarks. The unphysical colour-singlet component, implicitly contained in the decomposition of the $3 \otimes \bar{3}$ representation of the gluons,

is eliminated by requiring

$$\sum_{i_m,j_m} \delta^{i_m}_{j_m}\, \mathcal{A}^{i_1\cdots i_m\ldots i_l}_{j_1\cdots j_m\ldots j_l} = 0, \tag{16}$$

if particle m is a gluon. The two parametrizations (colour-octet vs. colour-flow representation of the gluon) are related through

$$\mathcal{A}^{i_1\cdots i_l}_{j_1\cdots j_l}[h_1,\ldots,h_l] = \sum_{a_1,\ldots,a_l} (\Delta_{a_1})^{i_1}{}_{j_1} \cdots (\Delta_{a_l})^{i_l}{}_{j_l}\, \mathcal{A}_{a_1\cdots a_l}[h_1,\ldots,h_l], \tag{17}$$

$$\mathcal{A}_{a_1\cdots a_l}[h_1,\ldots,h_l] = \sum_{\substack{i_1,\ldots,i_l\\ j_1,\ldots,j_l}} (\Delta_{a_1})^{j_1}{}_{i_1} \cdots (\Delta_{a_l})^{j_l}{}_{i_l}\, \mathcal{A}^{i_1\cdots i_l}_{j_1\cdots j_l}[h_1,\ldots,h_l], \tag{18}$$

where the sums run over the colour indices present in $\mathcal{A}_{a_1\cdots a_l}$ and $\mathcal{A}^{i_1\cdots i_l}_{j_1\cdots j_l}$, respectively. The matrices $(\Delta_a)^i{}_j$ are given by

$$(\Delta_a)^i{}_j = \begin{cases} \delta^i_a & \text{for incoming anti-quarks and outgoing quarks (j is absent)} \\ \delta_{a\,j} & \text{for incoming quarks and outgoing anti-quarks (i is absent)} \\ \frac{1}{\sqrt{2}}(\lambda_a)^i{}_j & \text{for gluons} \end{cases}, \tag{19}$$

where λ_a ($a = 1,\ldots,8$) are the usual Gell-Mann matrices.[2] Note that, while RECOLA exclusively works in the colour-flow formalism, we give all formulae in both representations to allow for an easy translation from one to the other.

In the colour-flow representation the colour part of the Feynman rules contains products of Kronecker δs, and the colour structure of the amplitude can be obtained as a linear combination of all possible structures built from products of Kronecker δs carrying the colour indices of the external particles:

$$\mathcal{A}^{i_1\cdots i_l}_{j_1\cdots j_l}[h_1,\ldots,h_l] = \sum_P \delta^{i_1}_{j_{P(1)}}\, \delta^{i_2}_{j_{P(2)}} \cdots \delta^{i_l}_{j_{P(l)}}\, \mathcal{A}_{P(1)\,P(2)\cdots P(l)}[h_1,\ldots,h_l]. \tag{20}$$

Here, the sum runs over all possible permutations P of the labels of external gluons, incoming quarks, and outgoing anti-quarks. The amplitudes

[2] In our convention, the λ_a are normalized according to $\mathrm{Tr}(\lambda_a\lambda_b) = 2\,\delta_{ab}$.

$\mathcal{A}_{P(1)\,P(2)\cdots P(l)}[h_1,\ldots,h_l]$ are called structure-dressed amplitudes. For example in the process $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{Z}\,\mathrm{g}\,\mathrm{g}$, this decomposition is given by

$$\begin{aligned}\mathcal{A}^{i_2 i_4 i_5}_{j_1 j_4 j_5} \;=\;& \delta^{i_2}_{j_1}\,\delta^{i_4}_{j_4}\,\delta^{i_5}_{j_5}\,\mathcal{A}_{145} + \delta^{i_2}_{j_1}\,\delta^{i_4}_{j_5}\,\delta^{i_5}_{j_4}\,\mathcal{A}_{154} + \delta^{i_2}_{j_4}\,\delta^{i_4}_{j_1}\,\delta^{i_5}_{j_5}\,\mathcal{A}_{415} \\ &+ \delta^{i_2}_{j_4}\,\delta^{i_4}_{j_5}\,\delta^{i_5}_{j_1}\,\mathcal{A}_{451} + \delta^{i_2}_{j_5}\,\delta^{i_4}_{j_1}\,\delta^{i_5}_{j_4}\,\mathcal{A}_{514} + \delta^{i_2}_{j_5}\,\delta^{i_4}_{j_4}\,\delta^{i_5}_{j_1}\,\mathcal{A}_{541}, \end{aligned} \tag{21}$$

where j_1 denotes the colour index of the incoming quark, i_2 the one of the incoming anti-quark, and i_4, j_4 and i_5, j_5 the ones of the outgoing gluons. In order to render the notation more compact, we introduce the following l-dimensional vectors:

- $\vec{h}$ with components $h_1,\ldots,h_l$,
- $\vec{c}$ with components $c_1,\ldots,c_l$, where $c_n = P(n)$ if $\delta^{i_n}_{j_{P(n)}}$ is present in the colour structure (i.e. if particle n is a gluon, an incoming anti-quark or an outgoing quark) and $c_n = 0$ otherwise (i.e. if particle n is a colourless particle, an incoming quark or an outgoing anti-quark).

We then rewrite the structure-dressed amplitudes in the compact form

$$\mathcal{A}^{(\vec{c},\vec{h})} = \mathcal{A}_{P(1)\,P(2)\cdots P(l)}[h_1,\ldots,h_l]. \tag{22}$$

Recola computes the Born contribution $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and the one-loop contribution $\mathcal{A}_1^{(\vec{c},\vec{h})}$ to the structure-dressed amplitudes $\mathcal{A}^{(\vec{c},\vec{h})}$ for all values the vectors $\vec{c}$ and $\vec{h}$ can take according to the external particles of the process.

2.6. Squared amplitudes

Recola also computes the squared amplitude, summed over helicities and colours of the outgoing particles and averaged over helicities and colours of the incoming ones (the average is indicated by a "bar"):

$$\overline{\mathcal{A}^2} = \overline{\sum_{a_1,\ldots,a_l}}\;\overline{\sum_{h_1,\ldots,h_l}}\left|\mathcal{A}_{a_1\cdots a_l}[h_1,\ldots,h_l]\right|^2 = \overline{\sum_{\substack{i_1,\ldots,i_l\\ j_1,\ldots,j_l}}}\;\overline{\sum_{h_1,\ldots,h_l}}\left|\mathcal{A}^{i_1\cdots i_l}_{j_1\cdots j_l}[h_1,\ldots,h_l]\right|^2. \tag{23}$$

An order-by-order expansion of the previous formula defines the Born and one-loop contribution to the squared amplitude. Omitting for compactness the colour and helicity indices, we have

$$\overline{\mathcal{A}^2} = \overline{\sum}\left|\mathcal{A}_0+\mathcal{A}_1+\ldots\right|^2 = \overline{\sum}\left\{|\mathcal{A}_0|^2 + 2\,\mathrm{Re}\,(\mathcal{A}_1\,\mathcal{A}_0^*) + |\mathcal{A}_1|^2 + \ldots\right\}. \tag{24}$$

At the Born level RECOLA computes the squared amplitude $\left(\overline{\mathcal{A}^2}\right)_0$, given by

$$\left(\overline{\mathcal{A}^2}\right)_0 = \overline{\sum} |\mathcal{A}_0|^2, \tag{25}$$

at the one-loop level it computes the one-loop contribution $\left(\overline{\mathcal{A}^2}\right)_1$ to the squared amplitude. If the Born amplitude $\mathcal{A}_0$ does not vanish, $\left(\overline{\mathcal{A}^2}\right)_1$ is calculated as

$$\left(\overline{\mathcal{A}^2}\right)_1 = \overline{\sum} 2\,\mathrm{Re}\,(\mathcal{A}_1\,\mathcal{A}_0^*). \tag{26}$$

For processes with a vanishing Born amplitude $\mathcal{A}_0$ (but non-vanishing $\mathcal{A}_1$), RECOLA computes the first non-vanishing contribution to the squared amplitude, which in this case amounts to

$$\left(\overline{\mathcal{A}^2}\right)_1 = \overline{\sum} |\mathcal{A}_1|^2. \tag{27}$$

2.7. Colour- and spin-correlated squared amplitudes

In order to compute the subtraction terms in the Catani–Seymour dipole formalism [68, 69], colour- and spin-correlated squared tree-level amplitudes are needed.

Being $\mathcal{A}_{a_1\cdots a_l}$ the amplitude of a process, one builds for every coloured particle $n = 1,\ldots,l$ the amplitude $\mathcal{A}_{a_1\cdots a_l\,a}(n)$, where the original colour structures have been extended by an additional factor describing the emission of a gluon of colour a from particle n of the original process:

$$\mathcal{A}_{a_1\cdots a_n\cdots a_l\,a}(n) = \sum_{a'_n} (T_a)_{a_n\,a'_n}\,\mathcal{A}_{a_1\cdots a'_n\cdots a_l}, \qquad \text{with} \tag{28}$$

$$(T_a)_{a_n\,a'_n} = \begin{cases} +\frac{1}{\sqrt{2}}(\lambda_a)_{a_n\,a'_n} & \text{for incoming anti-quarks} \\ & \text{and outgoing quarks} \\ -\frac{1}{\sqrt{2}}(\lambda_a)_{a'_n\,a_n} & \text{for incoming quarks} \\ & \text{and outgoing anti-quarks} \\ \mathrm{i}\,f_{a_n\,a\,a'_n} & \text{for gluons} \end{cases}. \tag{29}$$

Here, $f_{a\,b\,c}$ are the structure constants[3] of $\mathrm{SU}(N_\mathrm{c})$ and $N_\mathrm{c} = 3$. In the colour-flow formalism used by RECOLA, Eqs. (28) and (29) translate into

$$\mathcal{A}^{i_1\cdots i_n\cdots i_l\,i}_{j_1\cdots j_n\cdots j_l\,j}(n) = \sum_{i'_n,j'_n} K^{i\,i_n j'_n}_{j\,j_n i'_n}\,\mathcal{A}^{i_1\cdots i'_n\cdots i_l}_{j_1\cdots j'_n\cdots j_l}, \tag{30}$$

[3]The normalization for f_{abc} is fixed by $[T_a, T_b] = \mathrm{i}\,f_{abc}\,T_c$.

and

$$K^{i\,i_n j'_n}_{j\,j_n i'_n} = \begin{cases} +\delta^{i_n}_{j}\delta^{i}_{i'_n} - \frac{1}{N_\mathrm{c}}\delta^{i_n}_{i'_n}\delta^{i}_{j} & \text{for incoming anti-quarks and outgoing quarks } (j_n \text{ and } j'_n \text{ are absent}) \\ -\delta^{j'_n}_{j}\delta^{i}_{j_n} + \frac{1}{N_\mathrm{c}}\delta^{j'_n}_{j_n}\delta^{i}_{j} & \text{for incoming quarks and outgoing anti-quarks } (i_n \text{ and } i'_n \text{ are absent}) \\ \delta^{i_n}_{j}\delta^{i}_{i'_n}\delta^{j'_n}_{j_n} - \delta^{i_n}_{i'_n}\delta^{i}_{j_n}\delta^{j'_n}_{j} & \text{for gluons} \end{cases} , \tag{31}$$

where $\mathcal{A}^{i_1\cdots i_l}_{j_1\cdots j_l}$ denotes the original amplitude, $\mathcal{A}^{i_1\cdots i_l\,i}_{j_1\cdots j_l\,j}(n)$ the modified one, and the additional gluon has colour indices (i,j).

Recola computes, at the Born level, the colour-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_\mathrm{c}(n,m)$ between particle n and m for all pairs (n,m), defined as

$$\begin{aligned} \left(\overline{\mathcal{A}^2}\right)_\mathrm{c}(n,m) &= c_n \overline{\sum} \left(\mathcal{A}_{a_1\cdots a_l\,a}(n)[h_1,\ldots,h_l]\right)^* \mathcal{A}_{a_1\cdots a_l\,a}(m)[h_1,\ldots,h_l] \\ &= c_n \overline{\sum} \left(\mathcal{A}^{i_1\cdots i_l\,i}_{j_1\cdots j_l\,j}(n)[h_1,\ldots,h_l]\right)^* \mathcal{A}^{i_1\cdots i_l\,i}_{j_1\cdots j_l\,j}(m)[h_1,\ldots,h_l], \end{aligned} \tag{32}$$

where the sum is over all colour and helicity indices. The global factor c_n is given by $c_n = 1/(2C_\mathrm{F})$ if particle n is a quark or an anti-quark and by $c_n = 1/(2C_\mathrm{A})$ if particle n is a gluon.[4] These choices are made such that $\left(\overline{\mathcal{A}^2}\right)_\mathrm{c}(n,m)$ is independent of the normalization chosen for $(T_a)_{a_n\,a'_n}$.

In the case of a $q\bar{q}$ or a gg splitting, the Catani–Seymour subtraction formalism requires also spin correlations to build the subtraction terms (the non-diagonal terms of Eqs. (5.8), (5.9), (5.40), (5.41), (5.67), (5.68), (5.99), (5.100), (5.147), (5.148), (5.167), (5.168), (5.185), (5.186) of Ref. [68]). These terms can essentially be obtained replacing the polarization vector of the splitting gluon by an appropriate four-vector. To this end, Recola provides, at the Born level, the spin–colour-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_\mathrm{sc}(n,m,v)$ for pairs (n,m) of external gluons n and external coloured particles m. It is given by the colour-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_\mathrm{c}(n,m)$ computed with the special vector v (to be provided by the user) instead of the usual polarization vector for gluon n.

[4] The Casimir operators in the fundamental and adjoint representation are defined as $\sum_a (T_a T_a)_{bc} = 2C_\mathrm{F}\delta_{bc}$ and $\sum_{c,d} f_{acd} f_{bcd} = 2C_\mathrm{A}\delta_{ab}$, respectively, such that $C_\mathrm{F} = (N_\mathrm{c}^2 - 1)/(2N_\mathrm{c})$ and $C_\mathrm{A} = N_\mathrm{c}$ as usual.

The QCD subtraction formalism of Ref. [68] can be adapted for application to QED by replacing gluons with photons and colour with electric charge. Due to the Abelian nature of QED, charge correlation does not involve special amplitudes (unlike colour correlation) and can be trivially built from the squared Born amplitude. In the subtraction method for QED, spin correlation is needed in the case of a photon splitting into a fermion anti-fermion pair, which, in an analogous manner to the QCD case, can be obtained by replacing the polarization vector of the corresponding photon with an appropriate four-vector. For this purpose, RECOLA provides, at the Born level, the spin-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_{\mathrm{s}}(n,v)$ for external photons n. It is given by the squared amplitude $\overline{\mathcal{A}^2}$ computed with a special vector v (to be provided by the user) instead of the usual polarization vector for photon n.

2.8. Selecting intermediate states and resonances

As a further useful feature, RECOLA offers the possibility to request specific intermediate states in the amplitude, i.e. to select those contributions where the final state of the process is reached via one or more definite intermediate particles. The cross-section of a process is often dominated by contributions where these intermediate states become resonant. Such contributions are related to specific combinations of production and decay subprocesses and can be additionally enhanced by the experimental cuts. A typical example is the production of fermion pairs originating from the decay of a gauge boson, e.g. in the process

$$\mathrm{u}\quad\bar{\mathrm{u}}\quad\to\quad\mathrm{g}\quad\mathrm{g}\quad\mathrm{Z}\quad\to\quad\mathrm{g}\quad\mathrm{g}\quad\mathrm{e}^+\quad\mathrm{e}^-\,, \tag{33}$$

with the Z boson decaying into the $\mathrm{e}^+\mathrm{e}^-$ pair. RECOLA allows to select such specific contributions, even with multiple and nested decays, like for example

$$\mathrm{e}^+\quad\mathrm{e}^-\quad\to\quad\mathrm{t}\,(\to\,\mathrm{W}^+\,(\to\,\mathrm{u}\,\bar{\mathrm{d}}\,)\quad\mathrm{b}\,)\quad\bar{\mathrm{t}}\,(\to\,\mathrm{e}^-\quad\bar{\nu}_e\quad\bar{\mathrm{b}}\,). \tag{34}$$

This feature can be used to extract the resonant parts of the amplitude. Moreover it can be employed to calculate matrix elements in the pole approximation. In this approximation, only the resonant parts of the amplitude are kept and the residues of the poles in the amplitude are calculated with on-shell momenta. To this end, the complex squared mass $\mu^2 = m^2 - \mathrm{i}m\Gamma$ of the resonant particle is replaced by its real part $\mathrm{Re}\,(\mu^2) = m^2$ everywhere in the amplitude except for the denominators of resonant propagators where the width Γ is kept. The latter are further evaluated off shell at $p^2 \neq m^2$, while

the rest of the amplitude is calculated with on-shell kinematics, i.e. $p^2 = m^2$. This mismatch is accounted for in RECOLA by the possibility of choosing a different value for p^2 in the denominator of the resonant propagator than in the rest of the amplitude. At NLO, the selection of resonant contributions to a process can be used to calculate the so-called factorizable corrections in the pole approximation[5]. In addition, RECOLA offers the possibility to switch off corrections of self-energy type related to the resonant propagators, which cancel in the pole approximation.[6]

In order to apply the pole approximation the following steps have to be performed:

- The potentially resonant contributions have to be selected in the process definition (see Section 4.3 for details).
- The potentially resonant particles have to be marked as resonant (see Section 4.2.24 for details).
- The user has to ensure that the momenta p of the resonant particles are on the mass shell, i.e. $p^2 = m^2$.
- The squared off-shell momenta in the denominators of the resonant propagators have to be set (see Section 4.5.1 for details).

In this way the matrix elements in the pole approximation can be obtained at LO and for the factorizable part at NLO. The non-factorizable NLO corrections are not provided by the present version of RECOLA.

Note that marking particles as resonant sets the widths of these particles to zero in all defined processes. In order to calculate some of the matrix elements within the pole approximation and others exactly, it is therefore necessary to reset RECOLA and perform the integrations of the different contributions sequentially.

[5]The implementation of the factorizable NLO corrections in RECOLA in the pole approximation has been validated for Drell–Yan processes with resonant Z and W boson, as well as for doubly resonant processes with intermediate Z and W bosons.

[6]While the omission of the self-energies related to the resonance improves the numerical stability in the pole approximation, these contributions have to be taken into account if the pole approximation is not used.

2.9. Conventions

RECOLA uses the following symbols (of type `character`) for the SM particles:

Higgs boson:	`'H'`
Goldstone bosons:	`'p0', 'p+', 'p-'`
Vector bosons:	`'g', 'A', 'Z', 'W+', 'W-'`
Neutrinos:	`'nu_e', 'nu_mu', 'nu_tau'`
Anti-Neutrinos:	`'nu_e~', 'nu_mu~', 'nu_tau~'`
Charged leptons:	`'e-', 'e+', 'mu-', 'mu+', 'tau-', 'tau+'`
Quarks:	`'u', 'd', 'c', 's', 't', 'b'`
Anti-Quarks:	`'u~', 'd~', 'c~', 's~', 't~', 'b~'` .

The helicities of the external particles are described in RECOLA by a variable of type `character` with the values `[+],[-],[0]`, or by an `integer` variable with values `+1,-1,0`. This value determines the spinor (polarization vector) to be attributed to the corresponding fermion (vector boson), and is fixed to zero for scalar particles:

Incoming fermion:	`[+]` or `+1`	$\to$	u_+
	`[-]` or `-1`	$\to$	u_-
Outgoing fermion:	`[+]` or `+1`	$\to$	$\bar{u}_+$
	`[-]` or `-1`	$\to$	$\bar{u}_-$
Incoming anti-fermion:	`[-]` or `-1`	$\to$	$\bar{v}_-$
	`[+]` or `+1`	$\to$	$\bar{v}_+$
Outgoing anti-fermion:	`[-]` or `-1`	$\to$	v_-
	`[+]` or `+1`	$\to$	v_+
Transverse vector boson:	`[+]` or `+1`	$\to$	ϵ_+
	`[-]` or `-1`	$\to$	ϵ_-
Longitudinal vector boson:	`[0]` or `0`	$\to$	ϵ_0
Scalar boson:	`[0]` or `0`	$\to$	1 .

The explicit expressions of the spinors and polarization vectors used in RECOLA can be found in Appendix A.

For the normalization of the amplitudes we use the conventions of Refs. [70, 71]. Accordingly, cross sections and decay widths are obtained

as

$$\sigma(P_1,P_2\to p_1,\ldots,p_n)=\frac{(2\pi)^4}{4\sqrt{(P_1\cdot P_2)^2-P_1^2P_2^2}}\int \mathrm{d}\Phi_n(P_1{+}P_2,p_1,\ldots,p_n)\,\overline{\mathcal{A}^2},$$

$$\Gamma(P\to p_1,\ldots,p_n)=\frac{(2\pi)^4}{2\sqrt{P^2}}\int \mathrm{d}\Phi_n(P,p_1,\ldots,p_n)\,\overline{\mathcal{A}^2}, \tag{35}$$

with

$$\mathrm{d}\Phi_n(p,p_1,\ldots,p_n)=\delta^4\Big(p-\sum_{i=1}^n p_i\Big)\prod_{i=1}^n\frac{\mathrm{d}^3p_i}{2E_i(2\pi)^3}\,. \tag{36}$$

The SM Feynman rules implemented in Recola follow the conventions of Refs. [59, 71].

3. Installation

Since Recola relies on the Collier library, the installation of both packages is required to have a working amplitude generator. The following two options are available:

- Download of the stand-alone Recola–Collier package for a combined installation of both libraries.
- Download of the Recola package alone which then has to be linked to a local Collier installation.

Both packages are available from the web site "http://recola.hepforge.org".

3.1. The Recola–Collier *package*

The stand-alone package `recola-collier-X.Y.Z` contains the version `X.Y.Z` of the Recola library together with a working copy of the Collier library. After downloading the file `recola-collier-X.Y.Z.tar.gz`, extract the tarball in the current working directory with the shell command

```
"tar -zxvf recola-collier-X.Y.Z.tar.gz" .
```

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

- `CMakeLists.txt`:
  CMake makefile to produce the Collier and Recola libraries;

- `build`:
  build directory, where CMake puts all files necessary for the creation of the libraries;

- `COLLIER-A.B.C`:
  main directory of the Collier package `COLLIER-A.B.C`;

- `recola-X.Y.Z`:
  main directory of the Recola package `recola-X.Y.Z` (see Section 3.2 for details).

The combined compilation of Collier and Recola proceeds by changing to the `build` directory and executing there the shell command "`cmake [options] ..`" (creating Makefiles for Collier and Recola located in `COLLIER-A.B.C/build` and `recola-X.Y.Z/build`, respectively), followed by `make`:

```
"cd recola-collier-X.Y.Z/build"
"cmake [options] .."
"make" .
```

This requires CMake to be installed on the system. If no options are specified, CMake automatically searches for installed Fortran compilers and chooses a suited one, e.g. `gfortran`. The user can force CMake to use a specific compiler by adding in the `cmake` command line the option

```
"-D CMAKE_Fortran_COMPILER=<comp>" ,
```

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

By default, the installation sequence generates Collier and Recola as shared libraries:

- `libcollier.so` is created in `recola-collier-X.Y.Z/COLLIER-A.B.C` and the corresponding module files are placed in the `modules` subdirectory within this folder.

- `librecola.so` is created in `recola-collier-X.Y.Z/recola-X.Y.Z` and the corresponding module files are placed in the `modules` subdirectory within this folder.

The option

```
"-D static=ON"
```

causes CMake to create the static libraries `libcollier.a` and `librecola.a` instead of the shared ones.

The packages Recola and Collier both contain a directory called `demos`, where the user can find programs that illustrate the usage of the codes. Issuing

```
"make <demofile>"
```

in the directory `recola-collier-X.Y.Z/build`, with `<demofile>` being the name (without extension) of one of these demofiles, will compile the corresponding program. The executable `<demofile>` is created in the respective directory `demos` of Recola or Collier. More details on Recola demo programs are given in Section 3.2; for more details on the demo programs of Collier we refer to Ref. [48].

3.2. The Recola *package*

If the user wants to use his local installation[7] of Collier with Recola, he can download the archive `recola-X.Y.Z.tar.gz` containing only the Recola package, in its version `X.Y.Z`. Extract the tarball in the current working directory with the shell command

```
"tar -zxvf recola-X.Y.Z.tar.gz" .
```

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

- `CMakeLists.txt`:
  CMake makefile, to produce Recola library;
- `build`:
  build directory, where CMake puts all necessary files for the creation of the library;
- `src`:
  Recola source directory, containing

[7] Collier can be downloaded from http://collier.hepforge.org.

  - the source files `input_rcl.f90`, `process_definition_rcl.f90`, `process_generation_rcl.f90`, `process_computation_rcl.f90`, and `reset_rcl.f90` with the global variables and public subroutines accessible to the user;
  - the directory `internal` with private source files which should not be modified by the user.

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

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

```
"cd recola-X.Y.Z/build"
"cmake [options] .."
"make [demofile]" .
```

This requires CMAKE to be installed on the system. If no 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 adding in the `cmake` command line the option

```
"-D CMAKE_Fortran_COMPILER=<comp>" ,
```

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

By default, the installation sequence generates RECOLA as shared library `librecola.so` in the directory `recola-X.Y.Z`, with the corresponding module files placed in the `modules` subdirectory. The option

```
"-D static=ON"
```

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

The installation procedure further links RECOLA with the COLLIER library. If not specified otherwise, CMAKE assumes the existence of a folder named `COLLIER` that is located in the parent directory of `recola-X.Y.Z` and that contains the COLLIER library `libcollier.so` and/or `libcollier.a`,

as well as the subdirectory `modules` with the module files of COLLIER. Note that, depending on whether RECOLA shall be generated as shared or as static library, the COLLIER library must be present in the same format.

While the location of the `libcollier.so/libcollier.a` and the module files within the COLLIER folder must be kept as described above, the overall path may deviate from the default setting. In this case, the full path to the COLLIER directory must be given to CMAKE via the option

```
"-D collier_path=<path to collier>" ,
```

where `<path to collier>` can be either an absolute or a relative path.

Moreover, by adding the option

```
"-D cmake_collier=ON"
```

to the `cmake` command line, the user can enforce that COLLIER is (re-)compiled when the installation sequence for RECOLA is performed[8]. In this case the CMAKE makefile of RECOLA calls the CMAKE makefile of COLLIER and generates the COLLIER Makefile (any existing COLLIER Makefile is overwritten). The subsequent execution of `make` in `recola-X.Y.Z/build` then generates the COLLIER library and module files (placed in the respective directory) in addition to RECOLA library and modules.

To create executables for the demo programs of RECOLA in the directory `demos`, the command

```
"make <demofile>"
```

should be issued in the directory `recola-X.Y.Z/build`, with `<demofile>` being either `demo0_rcl`, `demo1_rcl`, `demo2_rcl` or `demo3_rcl`. Alternatively, the user can execute (after issuing `cmake`) the shell scripts `run` with the command

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

in the `demos` directory. This generates and runs the respective executable `<demofile>`.

The demo programs `demo0_rcl`, `demo1_rcl`, `demo2_rcl`, `demo3_rcl` exemplify the usage of RECOLA for various purposes:

[8]This only works if the complete COLLIER package with all source files is provided in the respective folder.

- `demo0_rcl`:
  Basic usage of Recola.

- `demo1_rcl`:
  Usage of Recola 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 coupling constant. In addition, files with LaTeX source code for diagrams are generated.

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

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

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 (see Section 4.1.11 for more details). It can be run executing

```
./draw-tex
```

in the `demos` directory.

4. Usage of Recola

In order to use Recola 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` or `librecola.a` has to be supplied to the linker. This gives access to the public functions and subroutines of the Recola 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 Recola library.

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

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

  The input needed for the computation of SM processes can be set by the user in two ways: either by editing the file `input.f90`, changing there the values of the corresponding variables explicitly, or by making use of subroutines provided by RECOLA for this purpose. While the former option requires a recompilation of the program, the latter allows for dynamical changes of the input parameters within the same run of the program. Input variables and subroutines are described in Section 4.1 and Section 4.2, respectively. Since RECOLA provides default values for all input parameters, this first step is optional.

- **Step 2: Defining the processes**

  Before RECOLA 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 Section 4.3.

- **Step 3: Generating the processes**

  In the next step the subroutine `generate_processes_rcl` is called which triggers the initialization 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 Section 4.4 for details).

- **Step 4: Computing the processes**

  After the arrangements made in the previous steps, RECOLA is ready to calculate amplitudes for any of the processes defined in step 2. 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.

  RECOLA further provides subroutines that allow to obtain particular contributions of the amplitude or the squared amplitude. In particular,

it is possible to calculate colour- and/or spin-correlated squared amplitudes at the Born level. Making use of the subroutines `set_alphas_rcl` or `compute_running_alphas_rcl` one can also work with a running value for the strong coupling constant α_s.

Detailed information on the subroutines that can be employed in step 4 will be given in Section 4.5.

- **Step 5: resetting** RECOLA

  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 RECOLA 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 the description of the routines below.

Examples of calls of RECOLA can be found in the directory `demos`.

4.1. Input variables

The physical input parameters and the flags steering the output are declared and initialized in the file `input.f90`. This file contains default values for these variables, which can be changed by the user.

4.1.1. Pole masses and widths of the SM particles

The SM particles can be grouped into massive unstable particles, characterized by their mass and decay width, massive stable particles, characterized by their mass, and massless ones. In RECOLA, the gluon, the photon and the neutrinos are treated as strictly massless (though the photon and the gluon can actually get a fictitious mass to regularize soft singularities, see Section 4.1.3). The electron as well as the up, down and strange quarks are considered in RECOLA as (potentially) massive stable particles, implying the possibility to assign to them a non-zero mass. All other particles are considered as (potentially) massive unstable particles, so that apart from a non-zero mass they can also be assigned a non-zero width.

The mass and width variables declared in `input.f90` represent the pole mass $m_{\rm P}$ and the pole width $\Gamma_{\rm P}$, defined from the complex pole $s_{\rm P}$ of the propagator as

$$s_{\rm P} = m_{\rm P}^2 - {\rm i}\,\Gamma_{\rm P}\,m_{\rm P}. \tag{37}$$

They are related to the on-shell quantities $m_{\rm OS}$ and $\Gamma_{\rm OS}$ measured at the LEP and Tevatron experiments for the W and the Z boson via

$$m_{\rm P} = \frac{m_{\rm OS}}{\sqrt{1+\Gamma_{\rm OS}^2/m_{\rm OS}^2}}, \qquad \Gamma_{\rm P} = \frac{\Gamma_{\rm OS}}{\sqrt{1+\Gamma_{\rm OS}^2/m_{\rm OS}^2}}. \tag{38}$$

The default values (in GeV) for the pole masses and widths in RECOLA are:

```
real(dp) :: mass_z  =  91.153480619182744d0
real(dp) :: width_z  = 2.4942663787728243d0
real(dp) :: mass_w  =  80.357973609877547d0
real(dp) :: width_w  = 2.0842989982782196d0
real(dp) :: mass_h  = 125.d0,                width_h  = 0.d0
real(dp) :: mass_el =   0.d0
real(dp) :: mass_mu =   0.d0,                width_mu = 0.d0
real(dp) :: mass_ta =   0.d0,                width_ta = 0.d0
real(dp) :: mass_u  =   0.d0
real(dp) :: mass_d  =   0.d0
real(dp) :: mass_c  =   0.d0,                width_c  = 0.d0
real(dp) :: mass_s  =   0.d0
real(dp) :: mass_t  = 173.2d0,               width_t  = 0.d0
real(dp) :: mass_b  =   0.d0,                width_b  = 0.d0 ,
```

where `dp` indicates a double precision variable. The default values for the Z and W bosons have been computed from the on-shell values

$$\begin{aligned} M_{\rm Z}^{\rm OS} &= 91.1876\,{\rm GeV}, & \Gamma_{\rm Z}^{\rm OS} &= 2.4952\,{\rm GeV}, \\ M_{\rm W}^{\rm OS} &= 80.385\,{\rm GeV}, & \Gamma_{\rm W}^{\rm OS} &= 2.085\,{\rm GeV}. \end{aligned}$$

4.1.2. Parameters governing the treatment of collinear singularities

The treatment of collinear singularities caused by a fermion f is fixed from the value of its pole mass m_f: for $m_f = 0$, dimensional regularization is applied with the regularization parameters given in Section 4.1.4. For $m_f \neq 0$, the variable `light_`f of type `logical`, which can be set individually for each fermion $f =$ `el`, `mu`, `ta`, `u`, `d`, `c`, `s`, `t`, `b`, controls how RECOLA deals with the fermion mass m_f:

- `light_`f = .true.:
  The fermion f is considered as "light", and its non-zero pole mass m_f is kept only in mass-singular logarithms but neglected elsewhere. In this case, m_f is used as mass regulator.

- `light_`f = .false.:
  The fermion f is considered as heavy, and the full dependence on the pole mass m_f is kept.

If the pole mass of fermion f is set to $m_f = 0$, the actual value of `light_`f is irrelevant. The default values of `light_`f are

```
logical :: light_el = .true.
logical :: light_mu = .true.
logical :: light_ta = .true.
logical :: light_u  = .true.
logical :: light_d  = .true.
logical :: light_c  = .true.
logical :: light_s  = .true.
logical :: light_t  = .false.
logical :: light_b  = .true. .
```

4.1.3. Parameters governing the treatment of soft singularities

Soft singularities can be cured in RECOLA in two different ways, and the method to be applied is selected depending on the value of the `integer` variable `reg_soft`:

- `reg_soft` = 1:
  Dimensional regularization is used with the regularization parameters given in Section 4.1.4.

- `reg_soft` = 2:
  Mass regularization is used with regulator `lambda` for the photon/gluon mass.

In the case of dimensional regularization (`reg_soft` = 1), the value of the variable `lambda` is irrelevant. Note further that mass regularization for soft singularities (`reg_soft` = 2) is only allowed in combination with mass regularization for all collinear singularities. Therefore, in the case of `reg_soft` = 2, the masses of external charged fermions must be chosen different from

zero for all processes. The default values for `reg_soft` and `lambda` (in GeV) are

```
integer :: reg_soft = 1;   real(dp) :: lambda = 100d0 .
```

4.1.4. Dimensional regularization parameters

The conventions used by RECOLA for dimensional regularization have been introduced in Section 2.2. The regularization parameters are represented by the variables (of type `real(dp)`)

$$\texttt{DeltaUV} = \Delta_{\rm UV}, \qquad \texttt{muUV} = \mu_{\rm UV}, \tag{39}$$

$$\texttt{DeltaIR} = \Delta_{\rm IR}, \qquad \texttt{DeltaIR2} = \Delta_{\rm IR2}, \qquad \texttt{muIR} = \mu_{\rm IR}, \tag{40}$$

with the default values (in GeV for `muUV` and `muIR`)

```
real(dp) :: DeltaUV = 0d0,                    muUV = 100d0
real(dp) :: DeltaIR = 0d0, DeltaIR2 = 0d0, muIR = 100d0 .
```

4.1.5. Parameters for the renormalization of the QCD coupling

RECOLA uses an $\overline{\rm MS}$ prescription for the renormalization of the strong coupling constant (see Section 2.2 for details). The renormalization scheme is fixed by the choice of the scale Q, given by the `real(dp)` variable `Qren`, and the selection of the number of active flavours $N_{\rm f}$, given by the `integer` variable `Nfren` accepting the following values:

- `Nfren` $= -1$:
  The *variable-flavour scheme* is used, and all quarks with masses lower than `Qren` are considered active flavours.

- `Nfren` $= 3, 4, 5, 6$:
  A *fixed-flavour scheme* is used, and the $N_{\rm f}$ = `Nfren` lightest quarks (those with the smallest mass values) are treated as active flavours. Note that the other quarks, which are not taken as active flavours, must have a non-vanishing mass. By default the pole masses are used for inactive flavours, but the masses m_c, m_b and m_t of the three heaviest quarks can be set exclusively for α_s renormalization upon using the subroutine `set_alphas_masses_rcl` described in Section 4.2.17.

As numerical input, RECOLA needs the value for the `real(dp)` variable `als` corresponding the QCD coupling $\alpha_s(Q^2)$ at the scale Q in the selected flavour scheme (see also Section 2.3). The default values for the variables introduced in this section are (in GeV for `Qren`)

```
real(dp) :: als   = 0.118d0
real(dp) :: Qren  = 91.1876d0
integer  :: Nfren = 5 .
```

4.1.6. Parameters for the renormalization of the EW coupling

The `integer` variable `ew_reno_scheme` allows to switch between the three renormalization schemes for the EW coupling α implemented in RECOLA (see Section 2.4):

- `ew_reno_scheme` $= 1$:
  The G_F scheme is chosen, and α is determined from the Fermi constant G_F, given by the value of the variable `gf` of type `real(dp)` .

- `ew_reno_scheme` $= 2$:
  The $\alpha(0)$ scheme is chosen, and α is set to the value $\alpha(0)$ stored in the variable `al0` of type `real(dp)` .

- `ew_reno_scheme` $= 3$:
  The $\alpha(M_\mathrm{Z})$ scheme is chosen, and α is set to the value $\alpha(M_\mathrm{Z})$ stored in the variable `alZ` of type `real(dp)` .

Depending on the selected scheme, the EW coupling α is determined from the corresponding variable `gf`, `al0` or `alZ`, while the other two variables do not enter the calculation and their actual values are thus irrelevant. The default values (in GeV^{-2} for `gf`) are

```
integer  :: ew_reno_scheme = 1
real(dp) :: gf  = 1.16637d-5
real(dp) :: al0 = 1d0/137.035999679d0
real(dp) :: alZ = 1d0/128.936d0 .
```

4.1.7. Parameter for the renormalization of masses of unstable particles

For the renormalization of the masses of unstable particles, the `integer` variable `complex_mass_scheme` allows to choose between the complex-mass or the on-shell renormalization scheme (see Section 2.4):

- `complex_mass_scheme` = 1:
  The complex-mass scheme of Refs. [52, 53, 65] is used.

- `complex_mass_scheme` = 0:
  The on-shell scheme is used.

By default, the complex-mass scheme is selected:

```
integer :: complex_mass_scheme = 1 .
```

4.1.8. Parameter for self-energies of resonant particles

Whether self-energy insertions (including the corresponding counterterms) are taken into account for resonant particles, is determined by the variable `resSE` (of type `logical`):

- `resSE = .true.`:
  Self-energies of resonant particles are included in the computation of the NLO amplitude.

- `resSE = .false.`:
  Self-energies of resonant particles are excluded in the computation of the NLO amplitude.

By default, self-energies of resonant particles are included:

```
logical :: resSE = .true. .
```

While the self-energies should always be included if the resonant particles are off shell, their omission improves the numerical stability for on-shell resonances, where the self-energies should cancel exactly.

4.1.9. Parameter for dynamic settings of parameters

The variable `dynamic_settings` regulates which input parameters can be set dynamically, i.e. after the call of `generate_processes_rcl`:

- `dynamic_settings = 0`:
  Only `alpha_s` can be reset after `generate_processes_rcl`.

- `dynamic_settings = 1`:
  In addition to `alpha_s` also `lambda`, `DeltaUV`, `muUV`, `DeltaIR2`, `DeltaIR` and `muIR` can be reset after `generate_processes_rcl`.

The default value is:

```
integer :: dynamic_settings = 0 .
```

4.1.10. Parameter for the automatic correction of external momenta

The variable `momenta_correction` steers the intrinsic correction of external momenta `p(0:3,1:)` (passed to RECOLA by the user in the computation phase), if they do not fulfil the mass-shell condition or four-momentum conservation. Independently of the value of this variable RECOLA always performs the following checks:

(1) Check that the dimension of the second index of `p` coincides with the number of external particles.

(2) Check that all particles have positive energy larger than or equal to their mass and that the total incoming energy is sufficient to produce the (on-shell) outgoing particles.

(3) Check the mass-shell condition of external momenta and four-momentum conservation.

As a result of these checks, RECOLA behaves as follows:

- If (1) is not fulfilled, RECOLA writes an error message and stops, without trying to correct the input momenta.
- If (2) is not fulfilled, RECOLA writes an error message. If the discrepancy is larger than 10^{-7} RECOLA stops. If (2) is not fulfilled but the discrepancy is smaller than 10^{-7} the phase-space point is considered unphysical and all amplitudes are set to 0. Again RECOLA does not try any corrections at this point.
- If (3) is not fulfilled and `momenta_correction=.true.`, RECOLA writes a warning message and tries to correct the momenta. If the correction fails, the phase-space point is considered unphysical and all amplitudes are set to 0.
- If (3) is not fulfilled and `momenta_correction=.false.`, RECOLA writes a warning message, the phase-space point is considered unphysical and all computed amplitudes are set to 0.
- RECOLA never stops upon performing check (3).

The default value for `momenta_correction` is:

```
logical :: momenta_correction = .true.
```

4.1.11. Options for the visualization of the off-shell currents

The off-shell currents and branches used by `recola` as building blocks for the construction of the amplitude (see Ref. [16] for details) can be visualized as diagrams, similar to the Feynman-diagrammatic visualization of the amplitude in the traditional approach. The `integer` variable `draw` determines the options for the generation of a file with LaTeX source code for the corresponding diagrams:

- `draw = 0`:
  No LaTeX file is generated.

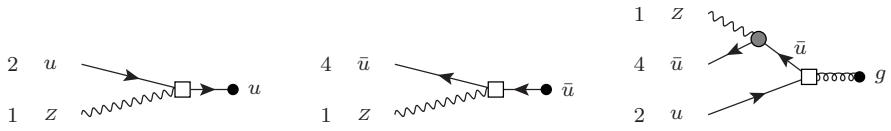
- `draw = 1`:
  For each process, a LaTeX file `process_`n`.tex` is created, where n is the identifier assigned to the process in the call of `define_process_rcl` (see Section 4.3).

  After a legend, the LaTeX file states the process under consideration. According to the internal conventions of Recola, outgoing particles are crossed into the initial state, and each particle carries an identifier in form of a binary number. For instance, for the process $u\bar{u} \to Zgg$ the output reads:

z	+	u	+	ū	+	g	+	g	→	0
1		2		4		8		16		

  Then all branches needed for the tree-level amplitude are drawn, listed following the order in which they are computed:

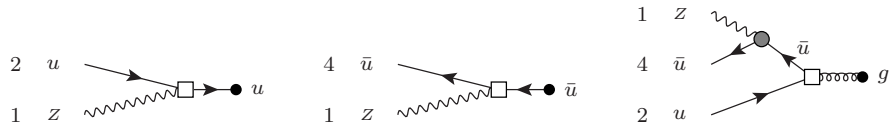
  Tree level



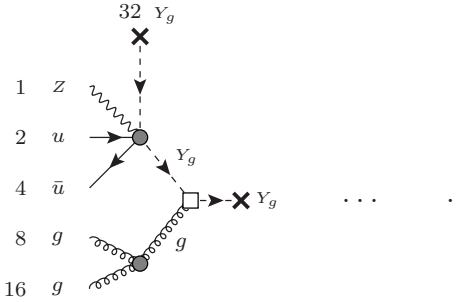
  . . . . . .

Next, the branches needed for the 4-dimensional bare one-loop contribution are drawn, grouped according to the respective cut-particle and listed following the order in which they are computed:

Bare loop; cut-particle: Y_g



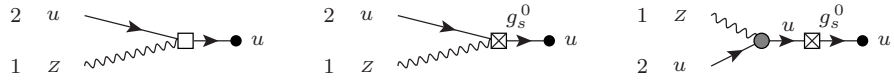
. . . . . .



. . . .

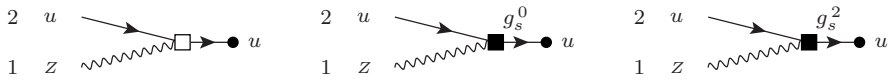
Finally, the branches for the counterterm contribution and for the contribution of the rational parts R_2 are drawn, again listed following the order in which they are computed:

Counterterms



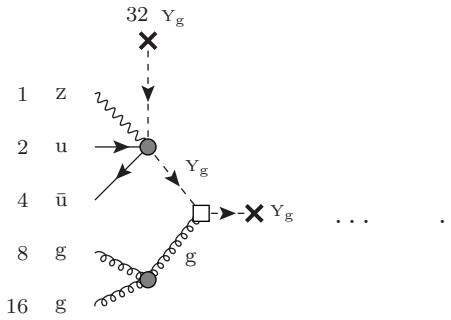
. . . . . ,

Rational terms



. . . . . .

- `draw = 2`:
  For each process, a LaTeX file is generated with the content as in the case `draw = 1`, and in addition the colour structures of the incoming and outgoing currents are explicitly written below each branch:



. . . . . .

The default value for the variable `draw` is

```
integer :: draw = 0 .
```

4.1.12. Output options

The `integer` variables `writeMat`, `writeMat2`, `writeCor`, and `writeRAM` define to which extent output is written by RECOLA into the standard output channel. Independently of the values of these parameters, RECOLA always prints an initialization message followed by the values of the input parameters. For default values, this output reads:

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

                 REcursive Computation of One Loop Amplitudes

                                 Version 1.2

       by S.Actis, A.Denner, L.Hofer, J.-N.Lang, A.Scharf, S.Uccirati

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

-------------------------------------------------------------------------------
 Pole masses and widths [GeV]:
 M_Z   =  91.153480619183               Width_Z   =  2.4942663787728
 M_W   =  80.357973609878               Width_W   =  2.0842989982782
 M_H   =  125.00000000000               Width_H   =  0.0000000000000
 m_e   =  0.0000000000000
 m_mu  =  0.0000000000000               Width_mu  =  0.0000000000000
 m_tau =  0.0000000000000               Width_tau =  0.0000000000000
 m_u   =  0.0000000000000
 m_d   =  0.0000000000000
 m_c   =  0.0000000000000               Width_c   =  0.0000000000000
 m_s   =  0.0000000000000
 m_t   =  173.20000000000               Width_t   =  0.0000000000000
 m_b   =  0.0000000000000               Width_b   =  0.0000000000000
-------------------------------------------------------------------------------
 Renormalization done in the complex-mass scheme
-------------------------------------------------------------------------------
  EW Renormalization Scheme: gfermi       Gf = 0.11663700000000E-04 GeV^-2
                                          alpha_Gf = 0.75552541674271E-02
-------------------------------------------------------------------------------
 alpha_s Renormalization Scheme: 5-flavours Scheme
 alpha_s(Q) = 0.11800000000000           Q =  91.187600000000     GeV
-------------------------------------------------------------------------------
 Delta_UV   =  0.0000000000000           mu_UV =  100.00000000000     GeV
 Delta_IR^2 =  0.0000000000000
 Delta_IR   =  0.0000000000000           mu_IR =  100.00000000000     GeV
-------------------------------------------------------------------------------
 Dimensional regularization for soft singularities
-------------------------------------------------------------------------------
```

If `writeMat`, `writeMat2`, or `writeCor` are different from `0`, the process under consideration and the momenta of the incoming and outgoing particles

are printed whenever a routine is called which computes amplitudes (see Section 2.9 for particle conventions). For example, for a phase-space point for the process $u\bar{u} \to Zgg$ the output reads:

```
u u~ -> Z g g

p1 = (4000.000000000,    0.000000000,    0.000000000, 4000.000000000) GeV
p2 = (4000.000000000,    0.000000000,    0.000000000,-4000.000000000) GeV
p3 = (2489.514720448,-2307.817376695,  306.489328734, -877.164509143) GeV
p4 = (2694.735905719, 1902.356873535, 1290.340022324,-1406.293907431) GeV
p5 = (2815.749373833,  405.460503160,-1596.829351058, 2283.458416573) GeV .
```

Output options for the amplitude

The value of the variable `writeMat` determines the level of detail at which the results for the structure-dressed amplitudes $\mathcal{A}^{(\vec{c},\vec{h})}$ (see Section 2.5) are printed for all computed processes:

- `writeMat = 0`:
  Nothing is printed.

- `writeMat = 1`:
  The results of the non-vanishing $\mathcal{A}^{(\vec{c},\vec{h})}$ are listed for each helicity configuration $\vec{h}$ and colour structure $\vec{c}$.

  In order to illustrate the notation employed for helicity configurations and colour structures, we give the output for the helicity configuration $\vec{h} = (+,-,+,+,+)$ and the colour structure $\delta^{i_2}_{j_4}\,\delta^{i_4}_{j_5}\,\delta^{i_5}_{j_1}$ (see Section 2.9 for helicity conventions) in the sample process $u\bar{u} \to Zgg$:

  ```
  AMPLITUDE [GeV^-1]

  Helicity configuration:   u[+] u~[-] -> Z[+] g[+] g[+]

                        i2    i4    i5
  Colour structure:   d     d     d     .
                        j4    j5    j1
  ```

  For each helicity configuration and colour structure, the results for the Born and, if computed, one-loop amplitude are written in separate tables:

```
 gs |               Born Amplitude A0
----------------------------------------------------
  0 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  1 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  2 | (-0.88324465151595E-08,-0.29277610069124E-07)
  3 | ( 0.00000000000000E+00, 0.00000000000000E+00)
----------------------------------------------------
SUM | (-0.88324465151595E-08,-0.29277610069124E-07)

 gs |              1-loop Amplitude A1
----------------------------------------------------
  0 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  1 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  2 | ( 0.14556032031458E-06, 0.82828108853855E-08)
  3 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  4 | (-0.15273533002682E-05, 0.60662554073983E-05)
  5 | ( 0.00000000000000E+00, 0.00000000000000E+00)
----------------------------------------------------
SUM | (-0.13817929799536E-05, 0.60745382182836E-05)   .
```

The tables display the decomposition of the amplitude into contributions with different powers of the strong coupling g_s (as indicated in the first column). The last line of the tables contains the sum of all the contributions, i.e. the full amplitude.

- `writeMat = 2`:
  In addition to the output of `writeMat = 1`, RECOLA prints a decomposition of the one-loop amplitude (if computed) into the 4-dimensional bare contribution (`4-dimensional bare-loop Amplitude A1d4`), the counterterm contribution (`CT Amplitude A1ct`) and the contribution of the rational parts R_2 (`R2 Amplitude A1r2`), each of them again tabulated in powers of g_s:

```
 gs |    4-dimensional bare-loop Amplitude A1d4
----------------------------------------------------
  0 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  1 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  2 | ( 0.13137505127039E-06, 0.84965453708879E-08)
  3 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  4 | (-0.73297625052077E-06, 0.61361188439947E-05)
  5 | ( 0.00000000000000E+00, 0.00000000000000E+00)
----------------------------------------------------
SUM | (-0.60160119925038E-06, 0.61446153893656E-05)
```

```
 gs |               CT Amplitude A1ct
----------------------------------------------------
  0 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  1 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  2 | ( 0.11671303642281E-09,-0.14787829304366E-08)
  3 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  4 | ( 0.52939559203394E-22,-0.99261673506363E-23)
  5 | ( 0.00000000000000E+00, 0.00000000000000E+00)
----------------------------------------------------
SUM | ( 0.11671303642286E-09,-0.14787829304366E-08)

 gs |               R2 Amplitude A1r2
----------------------------------------------------
  0 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  1 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  2 | ( 0.14068556007768E-07, 0.12650484449342E-08)
  3 | ( 0.00000000000000E+00, 0.00000000000000E+00)
  4 | (-0.79437704974740E-06,-0.69863436596445E-07)
  5 | ( 0.00000000000000E+00, 0.00000000000000E+00)
----------------------------------------------------
SUM | (-0.78030849373963E-06,-0.68598388151511E-07)   .
```

Output options for the squared amplitude

The value of the variable `writeMat2` determines the level of detail at which the results for the squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_0$ and $\left(\overline{\mathcal{A}^2}\right)_1$ (defined in Section 2.6) are printed for all computed processes:

- `writeMat2 = 0`:
  Nothing is printed.

- `writeMat2 = 1`:
  The result for $\left(\overline{\mathcal{A}^2}\right)_0$, denoted as `| A0 |^2`, and the result for $\left(\overline{\mathcal{A}^2}\right)_1$ (if computed), denoted as `2*Re{ A1 * A0^* }` in the case of an existing tree-level contribution and as `| A1 |^2` in the case of a loop-induced process, are written in separate tables:

```
UNPOLARIZED SQUARED AMPLITUDE [GeV^-2]

 als |        | A0 |^2              als |   2*Re{ A1 * A0^* }
-----------------------------      -----------------------------
   0 |  0.00000000000000E+00           0 |  0.00000000000000E+00
```

```
   1 |  0.00000000000000E+00          1 |  0.00000000000000E+00
   2 |  0.28817412651700E-06          2 | -0.12242757592358E-06
   3 |  0.00000000000000E+00          3 | -0.89053802197584E-06
     |                                4 |  0.00000000000000E+00
-----------------------------  -----------------------------
 SUM |  0.28817412651700E-06   SUM | -0.10129655978994E-05  .
```

The tables display the decomposition of the amplitude into contributions with different powers of the strong coupling α_s (as indicated in the first column). The last line of the tables contains the sum of all the contributions, i.e. the full squared amplitude.

- `writeMat2 = 2`:
  The same output as in the case `writeMat2 = 1` is returned, extended by a decomposition of the squared one-loop amplitude $\left(\overline{\mathcal{A}^2}\right)_1$ (if computed) into the 4-dimensional bare-loop contribution (`2*Re{A1d4*A0^*}`), the counterterm contribution (`2*Re{A1ct*A0^*}`) and the contribution of the rational parts R_2 (`2*Re{A1r2*A0^*}`), each of them again tabulated in powers of α_s:

```
 als |   2*Re{A1d4*A0^*}  |   2*Re{A1ct*A0^*}  |   2*Re{A1r2*A0^*}
-----------------------------------------------------------------------
   0 |  0.00000000000E+00 |  0.00000000000E+00 |  0.00000000000E+00
   1 |  0.00000000000E+00 |  0.00000000000E+00 |  0.00000000000E+00
   2 | -0.11688477171E-06 | -0.14775782556E-08 | -0.40652259615E-08
   3 | -0.86084967845E-06 | -0.10229117946E-24 | -0.29688343530E-07
   4 |  0.00000000000E+00 |  0.00000000000E+00 |  0.00000000000E+00
-----------------------------------------------------------------------
 SUM | -0.97773445015E-06 | -0.14775782556E-08 | -0.33753569492E-07 .
```

For loop-induced processes, six columns are printed containing the three squared contributions `|A1d4|^2`, `|A1ct|^2` and `|A1r2|^2`, and the three interference terms `2*Re{A1d4*A1ct^*}`, `2*Re{A1ct*A1r2^*}` and `2*Re{A1r2*A1d4^*}`.

- `writeMat2 = 3`:
  The same output as in the case `writeMat2 = 1` is returned, extended by a decomposition into the contributions $\overline{\mathcal{A}_h^2}$ from the single helicity configurations $\vec{h}$. In this case, the (averaged) sum is only performed over colours. Only helicity configurations with non-vanishing contributions are displayed (for the notation see the description for `writeMat = 1`):

```
POLARIZED SQUARED AMPLITUDE [GeV^-2]

Helicity configuration:   u[+] u~[-] -> Z[+] g[+] g[+]

 als |       | A0h |^2                als |  2*Re{ A1h * A0h^* }
-----------------------------       -----------------------------
   0 |  0.00000000000000E+00           0 |  0.00000000000000E+00
   1 |  0.00000000000000E+00           1 |  0.00000000000000E+00
   2 |  0.28191064190249E-15           2 | -0.81359616022317E-15
   3 |  0.00000000000000E+00           3 | -0.98362830083235E-13
     |                                 4 |  0.00000000000000E+00
-----------------------------       -----------------------------
 SUM |  0.28191064190249E-15         SUM  | -0.99176426243458E-13  .
```

Output options for spin- or colour-correlated squared amplitudes

The value of the variable `writeCor` determines the level of detail at which the results for the colour- and/or spin-correlated squared tree amplitudes (see Section 2.7) are printed for all computed processes:

- `writeCor = 0`:
  Nothing is printed.

- `writeCor = 1`:
  The colour-correlated squared Born amplitude $\left(\overline{\mathcal{A}^2}\right)_{\mathrm{c}}(n,m)$ between all pairs of particles n and m is shown, denoted as `| A0c(`n`,`m`) |^2`. For instance, for the colour correlation between particle 1 and 3 in the process $\mathrm{gg} \to \mathrm{d\bar{d}e^+e^-}$ the sample output reads:

  ```
  COLOUR-CORRELATED SQUARED AMPLITUDE [GeV^-4]

  als |    | A0c(1,3) |^2
  ---------------------------
    0 |  0.00000000000000E+00
    1 |  0.00000000000000E+00
    2 | -0.73462206785491E-11
    3 |  0.00000000000000E+00
    4 |  0.00000000000000E+00
  ---------------------------
  SUM | -0.73462206785491E-11   .
  ```

  The spin- and colour-correlated squared Born amplitude $\left(\overline{\mathcal{A}^2}\right)_{\mathrm{sc}}(n,m,v)$ between a gluon n with special polarization vector v and an arbitrary

coloured particle m is shown, denoted as `| A0sc(`n`,`m`) |^2`. For the example from above, it reads:

```
SPIN- AND COLOUR-CORRELATED SQUARED AMPLITUDE [GeV^-4]

Polarization vector v for particle 1:
v(0) =  ( -1151.5040504618346     ,  0.0000000000000000     )
v(1) =  ( -497.28644383654944     ,  0.0000000000000000     )
v(2) =  (  580.15008908958009     ,  0.0000000000000000     )
v(3) =  ( -1151.5040504618348     ,  0.0000000000000000     )

als |   | A0sc(1,3) |^2
----------------------------
  0 |  0.00000000000000E+00
  1 |  0.00000000000000E+00
  2 | -0.16797878275849E-05
  3 |  0.00000000000000E+00
  4 |  0.00000000000000E+00
----------------------------
SUM | -0.16797878275849E-05   ,
```

where `v(0:3)` is the polarization vector introduced by the user for the gluon n.

The spin-correlated squared Born amplitude $\left(\overline{\mathcal{A}^2}\right)_{\rm s}(n)$ for a photon n is shown, denoted as `| A0s(`n`) |^2`. For instance, for the spin correlation of the first photon in the process $\gamma\gamma \to \mu^+\nu^-{\rm e}^+{\rm e}^-$ the sample output reads:

```
SPIN-CORRELATED SQUARED AMPLITUDE [GeV^-4]

Polarization vector v for particle 1:
v(0) =  ( -1151.5040504618346     ,  0.0000000000000000     )
v(1) =  ( -497.28644383654944     ,  0.0000000000000000     )
v(2) =  (  580.15008908958009     ,  0.0000000000000000     )
v(3) =  ( -1151.5040504618348     ,  0.0000000000000000     )

als |     | A0s(1) |^2
----------------------------
  0 |  0.84005617404439E-06
  1 |  0.00000000000000E+00
  2 |  0.00000000000000E+00
  3 |  0.00000000000000E+00
```

```
  4 |  0.00000000000000E+00
----------------------------
SUM |  0.84005617404439E-06   .
```

By default, the output is switched off completely:

```
integer :: writeMat  = 0
integer :: writeMat2 = 0
integer :: writeCor  = 0 .
```

Output options for memory needed by RECOLA.

RECOLA can print information on the amount of memory (RAM) needed in the present run. If and to which extent this information is written, is determined by the variable `writeRAM`:

- `writeRAM = 0`:
  Nothing is printed.

- `writeRAM = 1`:
  The amount of RAM permanently blocked by the information stored during the process generation, and the amount of RAM needed for the process computation are written to the standard output channel. In both cases the information is written before the memory is actually occupied, so that it can be used in the case of a memory overflow to detect the origin. The amount of RAM for process computation is already shown during the process generation, i.e. before any future call of `compute_process_rcl` that could exceed the memory limit of the computer.

- `writeRAM = 2`:
  The same output as in the case `writeRAM=1` is returned, and in addition also the amount of RAM temporally blocked during the process generation is printed. This memory is freed again before the end of the generation step.

By default nothing is printed:

```
integer :: writeRAM  = 0 .
```

4.2. Input subroutines

The variables declared in the file `input.f90` and described in detail in the previous section can be modified by the user also during run time. This is done with the help of the following input subroutines (defined in `input.f90`), which can be called at any time before the process generation is prompted via the call of `generate_processes_rcl`. For most input subroutines, later calls have no effect unless RECOLA is reset (see Section 4.7). Only certain subroutines can also be employed at later stages, among them most notably the subroutine `set_alphas_rcl` which can be used to work with a dynamical renormalization scale for the strong coupling α_s. In the following descriptions, the possibility of calling a subroutine also after the process generation will be emphasized explicitly, i.e. unless stated otherwise the subroutine can only be employed before the process generation.

4.2.1. `set_pole_mass_`*prtcl*`_rcl (m,g)`

This subroutine sets the pole mass `mass_`p and width `width_`p (in GeV) of the particle *prtcl* to `m` and `g`, respectively (`m` and `g` are of type `real(dp)`). Here, *prtcl* and p can take the following values:

particle	Z	W	Higgs	muon	tauon	charm quark	top quark	bottom quark
prtcl	`z`	`w`	`h`	`muon`	`tau`	`charm`	`top`	`bottom`
p	`z`	`w`	`h`	`mu`	`ta`	`c`	`t`	`b`

.

4.2.2. `set_pole_mass_`*prtcl*`_rcl (m)`

This subroutine sets the pole mass `mass_`p (in GeV) of the particle *prtcl* to `m` (of type `real(dp)`). Here, *prtcl* and p can take the following values:

particle	electron	up quark	down quark	strange quark
prtcl	`electron`	`up`	`down`	`strange`
p	`el`	`u`	`d`	`s`

.

4.2.3. `set_onshell_mass_`v`_rcl (m,g)`

This subroutine sets the on-shell mass and width (in GeV) of the W or Z boson, $v =$ `w`, `z`, to `m` and `g`, respectively (`m` and `g` are of type `real(dp)`). Since RECOLA internally works with pole masses, the values of `m` and `g` are used to set the pole mass `mass_`v and the pole width `width_`v according to

$$\texttt{mass\_}v = \frac{\texttt{m}}{\sqrt{1+\texttt{g}^2/\texttt{m}^2}}, \qquad \texttt{width\_}v = \frac{\texttt{g}}{\sqrt{1+\texttt{g}^2/\texttt{m}^2}}. \tag{41}$$

4.2.4. `set_light_fermions_rcl (m)`

This subroutine declares all massive fermions with masses (in GeV) smaller than or equal to `m` (of type `real(dp)`) as light (i.e. the variable `light_`f is set to `.true.` for each fermion f with `mass_`$f \leq$ `m`). Massive fermions with masses larger than `m` are declared heavy (i.e. the variable `light_`f is set to `.false.` for each fermion f with `mass_`$f >$ `m`). The subroutine call has no effect on massless fermions. Details on the variables `light_`f are given in Section 4.1.2. Note that the assignments made by `set_light_fermions_rcl` are not adjusted in case the fermion masses should be changed later on.

4.2.5. `set_light_`*ferm*`_rcl`

, `unset_light_`*ferm*`_rcl`

The first subroutine marks the fermion *ferm* as light (i.e. it sets `light_`f `= .true.`), the second as heavy (i.e. it sets `light_`f `= .false.`). Here *ferm* and f can take the following values:

fermion	electron	muon	tauon
ferm	`electron`	`muon`	`tau`
f	`el`	`mu`	`ta`

fermion	up quark	down quark	charm quark	strange quark	top quark	bottom quark
ferm	`up`	`down`	`charm`	`strange`	`top`	`bottom`
f	`u`	`d`	`c`	`s`	`t`	`b`

.

If the fermion *ferm* is massless the subroutine call has no effect. Details on the variables `light_`f are given in Section 4.1.2.

4.2.6. `use_dim_reg_soft_rcl`

This subroutine selects dimensional regularization for soft singularities (i.e. it sets `reg_soft = 1`). See Section 4.1.3 for details.

4.2.7. `use_mass_reg_soft_rcl (m)`

This subroutine selects mass regularization for soft singularities (i.e. it sets `reg_soft = 2`), and it sets (in GeV) the mass regulator to `m` (of type `real(dp)`). See Section 4.1.3 for details.

4.2.8. `set_mass_reg_soft_rcl (m)`

This subroutine sets (in GeV) the mass regulator `lambda` for soft singularities to `m` (of type `real(dp)`). See Section 4.1.3 for details.

4.2.9. `set_delta_uv_rcl (d)`

This subroutine sets the variable `DeltaUV` parametrizing the UV pole to `d` (of type `real(dp)`). See Section 4.1.4 for details.

4.2.10. `set_mu_uv_rcl (m)`

This subroutine sets (in GeV) the UV scale `muUV` to `m` (of type `real(dp)`). See Section 4.1.4 for details.

4.2.11. `set_delta_ir_rcl (d,d2)`

This subroutine sets the variables `DeltaIR` and `DeltaIR2` parametrizing the IR poles to `d` and to `d2`, respectively (`d` and `d2` are of type `real(dp)`). See Section 4.1.4 for details.

4.2.12. `set_mu_ir_rcl (m)`

This subroutine sets (in GeV) the IR scale `muIR` to `m` (of type `real(dp)`). See Section 4.1.4 for details.

4.2.13. `set_alphas_rcl (a,s,Nf)`

This subroutine sets the value of α_s (`als`) to `a`, the renormalization scale `Qren` to `s` and the number of light quark flavours `Nfren` to `Nf` (`a` and `s` are of type `real(dp)`, `Nf` is of of type `integer`). Details on the parameters `als`, `Qren` and `Nfren` can be found Section 4.1.5. The supplied value of `a` should correspond to the physical value of α_s at the scale `s` in the `Nf`-flavour scheme. Alternatively, RECOLA can take care of the determination of the corresponding α_s by means of the subroutine `compute_running_alphas_rcl` introduced in Section 4.5.2. The subroutine `set_alphas_rcl (a,s,Nf)` can be called before the process generation but also during the process computation, which allows to work with running values for α_s.

4.2.14. `get_alphas_rcl (a)`

This subroutine, which can be called before the process generation but also during the process computation, returns the current value of α_s in its output argument `a` (of type `real(dp)`).

4.2.15. `get_renormalization_scale_rcl (mu)`

This subroutine, which can be called before the process generation but also during the process computation, returns the current value of the renormalization scale Q (in GeV) for α_s in its output argument `mu` (of type `real(dp)`).

4.2.16. `get_flavour_scheme_rcl (Nf)`

This subroutine, which can be called before the process generation but also during the process computation, returns the current value of the identifier of the flavour-scheme `Nfren` for the renormalization of α_s in its output argument `Nf` (of type `integer`). See Section 4.1.5 for details.

4.2.17. `set_alphas_masses_rcl (mc,mb,mt,gc,gb,gt)`

This subroutine overwrites the values for the charm, bottom, and top masses that enter the counterterm of α_s. By default the pole masses are used for the inactive flavours in Eq. (7). The mass values set by this subroutine are exclusively used in the counterterm of α_s and do not enter the computation elsewhere. This feature allows to treat inactive quarks as massless in the calculation of the matrix elements. If the optional arguments `gc,gb,gt` (of type `real(dp)`) are not present, the values for $|m_c|^2$, $|m_b|^2$ and $|m_t|^2$ in Eq. (7) are set to

$$|m_c|^2 = \texttt{mc}^2 \qquad |m_b|^2 = \texttt{mb}^2, \qquad |m_t|^2 = \texttt{mt}^2. \tag{42}$$

Otherwise they are set to

$$|m_c|^2 = \texttt{mc}^2\sqrt{1+\frac{\texttt{gc}^2}{\texttt{mc}^2}}, \qquad |m_b|^2 = \texttt{mb}^2\sqrt{1+\frac{\texttt{gb}^2}{\texttt{mb}^2}}, \qquad |m_t|^2 = \texttt{mt}^2\sqrt{1+\frac{\texttt{gt}^2}{\texttt{mt}^2}}. \tag{43}$$

The arguments `mc,mb,mt` (of type `real(dp)`) must satisfy the condition $0 \leq$ `mc` $\leq$ `mb` $\leq$ `mt`. The optional arguments `gc,gb,gt` take into account width effects in the counterterm of α_s. They can not be negative and are ignored by RECOLA if the on-shell renormalization scheme is used. See Sections 2.2 and 2.3 for details.

4.2.18. `use_gfermi_scheme_rcl (g,a,massesgf)`

This subroutine selects the G_F scheme as renormalization scheme for the EW coupling. If the optional argument `g` is present, the Fermi constant G_F (`gf`) is set to the value of `g` (of type `real(dp)`). If instead the optional argument `a` is present, the value of α is set directly to the value of `a` (of type `real(dp)`). Only one of the two optional arguments `g` and `a` can be present: if `use_gfermi_scheme_rcl` is called with two arguments RECOLA writes an error message and stops. See Sections 2.4 and 4.1.6 for details.

The optional argument `massesgf` steers how α is calculated from G_F in the complex-mass scheme. For `massesgf` $= 0$, real vector-boson masses are

used,

$$\alpha_{G_{\rm F}} = \frac{\sqrt{2}G_{\rm F}}{\pi}\,{\rm Re}(\texttt{mass\_w}^2)\left(1-\frac{{\rm Re}(\texttt{mass\_w}^2)}{{\rm Re}(\texttt{mass\_z}^2)}\right), \tag{44}$$

while for `massesgf` $= 1$ (default), complex vector-boson masses are used,

$$\alpha_{G_{\rm F}} = \frac{\sqrt{2}G_{\rm F}}{\pi}\left|(\texttt{mass\_w}^2\left(1-\frac{\texttt{mass\_w}^2}{\texttt{mass\_z}^2}\right)\right|. \tag{45}$$

4.2.19. `use_alpha0_scheme_rcl (a)`

This subroutine selects the $\alpha(0)$ scheme as renormalization scheme for the EW coupling. If the optional argument `a` is present, $\alpha(0)$ (`al0`) is set to the value of `a` (of type `real(dp)`). See Sections 2.4 and 4.1.6 for details.

4.2.20. `use_alphaz_scheme_rcl (a)`

This subroutine selects the $\alpha(M_{\rm Z})$ scheme as renormalization scheme for the EW coupling. If the optional argument `a` is present, $\alpha(M_{\rm Z})$ (`alZ`) is set to the value of `a` (of type `real(dp)`). See Sections 2.4 and 4.1.6 for details.

4.2.21. `get_alpha_rcl (a)`

This subroutine, which can be called before the process generation but also during the process computation, returns the current value of α in its output argument `a` (of type `real(dp)`).

4.2.22. `set_complex_mass_scheme_rcl`

This subroutine selects the complex-mass scheme of Refs. [52, 53, 65] for the renormalization of the masses of unstable particles (i.e. it sets `complex_mass_scheme = 1`). See Section 4.1.7 for details.

4.2.23. `set_on_shell_scheme_rcl`

This subroutine selects the on-shell scheme for the mass renormalization of unstable particles (i.e. it sets `complex_mass_scheme = 0`). See Section 4.1.7 for details.

4.2.24. `set_resonant_particle_rcl (pa)`

This subroutine is only needed if processes are defined with specific intermediate particles and the pole approximation shall be applied to the corresponding resonances (see Section 4.3.1 and Section 2.8). It marks particle `pa` as resonant. The argument `pa` can be any of the characters listed in Section 2.9 for the SM particles.

If a particle is labelled resonant, RECOLA sets the imaginary part of its mass to zero everywhere except in the denominator of the resonant propagators. If the width of the particle `pa` is zero, RECOLA stops. During computation, RECOLA furthermore checks that the phase-space point provided by the user fulfils the condition that the squared momenta of all resonant particles `pa` are on shell (if this does not hold within a precision of 10^{-7}, RECOLA stops).

Calling `set_resonant_particle_rcl (pa)` affects all `compute_`X`_rcl` calls of Section 4.5, i.e. the pole approximation is also applied to colour- and/or spin-correlated squared amplitudes.

4.2.25. `switchon_resonant_selfenergies_rcl`

This subroutine sets the value of `resSE` to `.true.` (see Section 4.1.8 for details).

4.2.26. `switchoff_resonant_selfenergies_rcl`

This subroutine sets the value of `resSE` to `.false.` (see Section 4.1.8 for details).

4.2.27. `set_dynamic_settings_rcl (n)`

This subroutine sets the variable `dynamic_settings`, governing the possibility to set dynamically some parameters, to `n` (of type `integer`). See Section 4.1.9 for details.

4.2.28. `set_momenta_correction_rcl (mc)`

This subroutine sets the variable `momenta_correction`, governing the automatic correction of external momenta, to `mc` (of type `logical`). See Section 4.1.10 for details.

4.2.29. `set_draw_level_branches_rcl (n)`

This subroutine sets the variable `draw` for the generation of a LaTeX file with diagrams of the off-shell currents to `n` (of type `integer`). See Section 4.1.11 for details.

4.2.30. `set_print_level_amplitude_rcl (n)`

This subroutine, which can be called before the process generation but also during the process computation, sets the variable `writeMat` governing the output of amplitudes to `n` (of type `integer`). See Section 4.1.12 for details.

4.2.31. `set_print_level_squared_amplitude_rcl (n)`

This subroutine, which can be called before the process generation but also during the process computation, sets the variable `writeMat2` governing the output of squared amplitudes to `n` (of type `integer`). See Section 4.1.12 for details.

4.2.32. `set_print_level_correlations_rcl (n)`

This subroutine, which can be called before the process generation but also during the process computation, sets the variable `writeCor` governing the output of colour- and spin-correlated squared amplitudes to `n` (of type `integer`). See Section 4.1.12 for details.

4.2.33. `set_print_level_RAM_rcl (n)`

This subroutine sets the variable `writeRAM` governing the output of information on the amount of RAM needed to `n` (of type `integer`). See Section 4.1.12 for details.

4.2.34. `scale_coupling3_rcl (fac,pa1,pa2,pa3)`

This subroutine scales the 3-particle coupling between particles `pa1`, `pa2`, and `pa3` by a factor `fac` (of type `complex(dp)`). The arguments `pa1`, `pa2`, and `pa3` can be any of the characters listed in Section 2.9 for the SM particles.
Remark: The rescaling of couplings in the counterterms and rational terms is not yet implemented.

4.2.35. `scale_coupling4_rcl (fac,pa1,pa2,pa3,pa4)`

This subroutine scales the 4-particle coupling between particles `pa1`, `pa2`, `pa3`, and `pa4` by a factor `fac` (of type `complex(dp)`). The arguments `pa1`, `pa2`, `pa3`, and `pa4` can be any of the characters listed in Section 2.9 for the SM particles.
Remark: The rescaling of couplings in the counterterms and rational terms is not yet implemented.

4.2.36. `switchoff_coupling3_rcl (pa1,pa2,pa3)`

This subroutine switches off the 3-particle coupling between `pa1`, `pa2`, and `pa3`. The arguments `pa1`, `pa2`, and `pa3` can be any of the characters listed in Section 2.9 for the SM particles.
Remark: The switching off of couplings in the counterterms and rational terms is not yet implemented.

4.2.37. `switchoff_coupling4_rcl (pa1,pa2,pa3,pa4)`

This subroutine switches off the 4-particle coupling between `pa1`, `pa2`, `pa3`, and `pa4`. The arguments `pa1`, `pa2`, `pa3`, and `pa4` can be any of the characters listed in Section 2.9 for the SM particles.
Remark: The switching off of couplings in the counterterms and rational terms is not yet implemented.

4.2.38. `set_ifail_rcl (i)`

This subroutine, which can be called before the process generation but also during the process computation, sets the flag `ifail` to the `integer` value `i` given by the user. On input the flag `ifail` determines the behaviour of Recola upon encountering an internal error:

- `ifail = -1`: Execution of the program does not stop if an error occurs;
- `ifail =  0`: Execution of the program stops if an error occurs.

By default, the flag `ifail` is set to `0`.

4.2.39. `get_ifail_rcl (i)`

This subroutine, which can be called before the process generation but also during the process computation, extracts the value of the flag `ifail`, returning it as `integer` argument `i` to the user. On output, the flag `ifail` provides information about the appearance of an error during the run of Recola.

- `ifail = 0`: No errors occurred;
- `ifail = 1`: An error occurred due to an inconsistent call of a subroutine by the user;
- `ifail = 2`: An error occurred due to an unexpected problem within Recola.

4.2.40. `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'`.

4.2.41. `set_output_file_rcl (x)`

This subroutine, which can be called before the process generation but also during the process computation, opens the output file named `x` (`x` is of type `character`). Any output produced by RECOLA is written to the output file `x`, which is created with the first call of `set_output_file_rcl` (if it already exists, its content is cleared). For any subsequent call of `set_output_file_rcl` with different argument a new output file is opened, and if it exists its content is cleared. Redirecting the output to an already open output file does not clear the content and new output is appended. Output files remain open unless recola is reset by the call of `reset_recola_rcl`.

If the argument `x = '*'` is passed to `set_output_file_rcl`, the output is written into the terminal (standard output channel).

By default, the output is written to the file `output.rcl`, placed in the current working directory. The call of `reset_recola_rcl` resets the output file name to the default value `output.rcl`.

4.3. Process definition

Processes are declared by calling the subroutine `define_process_rcl` defined in the file `process_definition.f90`. This file also contains subroutines for the selection/unselection of contributions to the amplitude with specific powers of the strong coupling g_s (defined by $g_s^2 = 4\pi\alpha_s$) and a subroutine for the splitting of the cache of COLLIER into several parts. For a process with identifier `npr`, these subroutines can be called at any point after the declaration of the process `npr` via `define_process_rcl`, and before the complete set of processes is generated by the call of `generate_processes_rcl`. If no specific selection is made by the user, by default the contributions of all powers of g_s are computed. Since the contributions to the amplitude are proportional to $g_s^{n_s} e^{n-n_s}$ with n unambiguously fixed from the chosen process (and the selection of LO or NLO), the powers of the electromagnetic coupling e (defined by $e^2 = 4\pi\alpha$) are implicitly determined by the powers of g_s.

4.3.1. `define_process_rcl (npr,processIn,order)`

With this subroutine, the user defines the process(es) he wants to be computed. The `integer` argument `npr` is assigned as identifier to the process specified by the argument `processIn` (of type `character`). Issuing repeated calls of `define_process_rcl`, it is possible to define more than one process. In this case, each process `processIn` must be given a different identifier `npr` (calls of `define_process_rcl` with an already existing identifier are ignored).

The argument `order` of type `character` can take the values `'LO'` and `'NLO'`: For `order='LO'`, the process will be generated such that RECOLA will be able to evaluate only LO amplitudes, while for `order='NLO'` it will be generated such that RECOLA will be able to evaluate both LO and NLO amplitudes.

The string `processIn` must consist of a list of particles separated by `'->'`, with the incoming (outgoing) particles on the left-hand (right-hand) side of `'->'`. Any number of incoming and outgoing particles is allowed. The symbols for the particles, listed in Section 2.9, as well as the symbol `'->'` must be separated by at least one blank character. Examples of allowed character chains for `processIn` are

```
'e+ e- -> mu+ mu-' ,
'u u~ -> W+ W- g' ,
'u d~ -> W+ g g g' ,
'u  g  ->  u  g  Z' ,
'u    g  -> u        g  tau- tau+' .
```

To each fermion or vector particle a specific helicity can be attributed by adding the corresponding symbol `'[-]'`, `'[+]'` or `'[0]'` (see Section 2.9 for the definition of the helicity symbols). Blank characters can (but need not) separate the helicity symbol from the particle name:

`'e+[+] e- [-] -> Z H'`:	`'e+'` is right-handed, `'e-'` is left-handed, `'Z'` is unpolarized.
`'u u~ -> W+[-] W-[+]'`:	`'u'` and `'u~'` are unpolarized, `'W+'` and `'W-'` are transverse.

Contributions with specific intermediate states can be selected (see Section 2.8), where intermediate states are particles decaying into any number of other particles. To this end, in the process declaration the decaying particle must be followed by round brackets `'( ... )'` containing the decay products. Multiple and nested decays are allowed. Blank characters can (but need not) separate the brackets `'('`, `')'` from the particle names:

```
'e+ e- -> W+ W-(e- nu_e~)' ,
'e+ e- -> Z H ( b~[+] b[-] )' ,
'e+ e- -> t( W+(u d~) b) t~(e- nu_e~ b~)' ,
'u  u~ -> Z ( mu+(e+ nu_e nu_mu~) mu-(e- nu_e~ nu_mu) ) H' .
```

The selection of specific intermediate particles is a prerequisite for the calculation of amplitudes within the pole approximation.

4.3.2. `set_gs_power_rcl (npr,gsarray)`

This subroutine selects specific powers of g_s, as specified by the argument `gsarray`, for the process with identifier `npr` (of type `integer`). The elements of the two-dimensional `integer` array `gsarray`$(0:,0:1)$ can take the values 0 or 1, with the first index representing the power of g_s and the second one representing the loop order (0 for LO, 1 for NLO). For instance, a call of `set_gs_power_rcl` with second argument

```
gsarray(0,0) = 1
gsarray(1,0) = 0
gsarray(2,0) = 1
gsarray(0,1) = 0
gsarray(1,1) = 0
gsarray(2,1) = 0
gsarray(3,1) = 0
gsarray(4,1) = 1
```

selects the contributions with g_s^0 and g_s^2 for the tree amplitude and the g_s^4 contribution for the loop amplitude. All other contributions will not be computed in later evaluations of the amplitudes. By default, all contributions are switched on.

4.3.3. `select_gs_power_BornAmpl_rcl (npr,gspower)`

,

`unselect_gs_power_BornAmpl_rcl (npr,gspower)`

This pair of subroutines allows to select/unselect the contribution to the Born amplitude proportional to g_s^n, where n is given by the `integer` argument `gspower`, for the process with identifier `npr` (of type `integer`). 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.

4.3.4. `select_gs_power_LoopAmpl_rcl (npr,gspower)`

,

`unselect_gs_power_LoopAmpl_rcl (npr,gspower)`

This pair of subroutines allows to select/unselect the contribution to the loop amplitude proportional to g_s^n, where n is given by the `integer` argument `gspower`, for the process with identifier `npr` (of type `integer`). All other contributions to the loop amplitude keep their status (selected or unselected),

according to previous calls of selection subroutines. The selection of the contributions to the Born amplitude remains unaffected as well.

4.3.5. `select_all_gs_powers_BornAmpl_rcl (npr)`

,
`unselect_all_gs_powers_BornAmpl_rcl (npr)`

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

4.3.6. `select_all_gs_powers_LoopAmpl_rcl (npr)`

,
`unselect_all_gs_powers_LoopAmpl_rcl (npr)`

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

4.3.7. `split_collier_cache_rcl (npr,n)`

This subroutine splits the cache of COLLIER for the process with identifier `npr` (of type `integer`) into `n` (of type `integer`) separate sub-caches. This allows the user to bypass a possible memory problem connected with the size of the cache of COLLIER for complicated processes. Note, however, that this procedure can lead to a reduction of the efficiency of the cache system and should thus only be employed if memory problems are encountered in the default run mode.

4.4. Process generation: `generate_processes_rcl`

The file `process_generation.f90` only contains the single subroutine `generate_processes_rcl` for the generation of the processes. This subroutine constructs the skeleton of the recursive procedure for all processes previously defined by the user, and stores it in global variables that later on are used by RECOLA for the computation of amplitudes. It must be called **once** after all processes are defined and before the subroutines for process computation contained in `process_computation.f90` can be called. It is typically called before the phase-space points are generated.

4.5. Process computation

The amplitude and the squared amplitude for a process are computed by calling the subroutine `compute_process_rcl`. Their values are stored in internal variables labelled by the process identifier and the loop order and can be read out with the subroutines `get_amplitude_rcl`, `get_polarized_squared_amplitude_rcl` and `get_squared_amplitude_rcl`. After the computation, the amplitudes and squared amplitudes can be rescaled for a new value of α_s with the subroutine `rescale_process_rcl` (which overwrites the stored results for the given process at the given order). The typical sequence is

```
call compute_process_rcl(npr,...)
call get_A_rcl(npr,...) ,
```

possibly followed (after a redefinition of α_s through `set_alphas_rcl` or `compute_running_alphas_rcl`) by several calls of

```
call rescale_process_rcl(npr,...)
call get_A_rcl(npr,...) ,
```

where `npr` is the process identifier and A stands for `amplitude`, `polarized_squared_amplitude` and/or `squared_amplitude`.

Further subroutines of type `compute_`X`_correlation_rcl` and `rescale_`X`_correlation_rcl` (where X stands for `colour`, `spin` or `spin_colour`) are provided for the computation of spin- and/or colour-correlated squared Born amplitudes. For colour correlation also the routines `compute_all_colour_correlations_rcl` and `rescale_all_colour_correlations_rcl` are available. These subroutines compute (or rescale) the LO amplitudes and the LO spin- and/or colour-correlated squared amplitudes, storing the results in internal variables (for LO amplitudes these variables are the same as for `compute_process_rcl` and `rescale_process_rcl`). Previously computed results of these LO objects for the given process are overwritten. The stored results for the spin- and/or colour-correlated squared Born amplitudes can be read out by the user with the subroutines of type `get_`X`_correlation_rcl`. The typical sequence for the evaluation of spin- and/or colour-correlated squared Born amplitudes is

```
call compute_X_correlation_rcl(npr,...)
call get_X_correlation_rcl(npr,...) ,
```

possibly followed (after a redefinition of α_s) by several calls

```
call rescale_X_correlation_rcl(npr,...)
call get_X_correlation_rcl(npr,...)
```

for rescaling the LO amplitudes and spin- and/or colour-correlated amplitudes.

Note that a call of `compute_`Y`_rcl` or `rescale_`Y`_rcl` followed by a call of `compute_`Y'`_rcl` or `rescale_`Y'`_rcl` (with Y,Y' = `process`, X`_correlation`, `all_colour_correlations`) will lead to a recalculation of LO amplitudes for process `npr` that overwrites results previously obtained in the calls for Y for that process. In this case the results for the amplitudes calculated with `compute_`Y`_rcl` or `rescale_`Y`_rcl` cannot be accessed anymore. It is thus advisable to conclude the call sequence for an object Y, before a different object Y' is evaluated. More details are given in the description of the individual subroutines.

All these subroutines are defined in the file `process_computation.f90`, together with the subroutine `compute_running_alphas_rcl` that allows for the dynamical computation of α_s and the subroutine `set_resonant_squared_momentum_rcl` which enables the user to set the squared momentum of the denominator of resonant propagators.

4.5.1. `set_resonant_squared_momentum_rcl (npr,res,ps)`

This subroutine is only relevant for processes defined with intermediate particles (see Section 4.3.1 and Section 2.8). It acts on the resonance with identifier `res` (of type `integer`) in the process with identifier `npr` (of type `integer`) in the following way:

- it checks whether the resonant particle corresponding to `res` has been marked resonant through the subroutine `set_resonant_particle_rcl` (see Section 4.2.24);
- it sets the squared momentum in the denominator of the resonant propagator to `ps` (of type `real(dp)`).

The resonance identifier `res` is derived from the process definition in the call of `define_process_rcl`. The first resonant particle defined there is labelled with `res` = 1, the second with `res` = 2, and so on. For example, in the process

```
'e+ e- -> t( W+(u d~) b) t~(e- nu_e~ b~)' ,
```

the first resonant particle (`res` = 1) is the top-quark, the second (`res` = 2) the W^+, and the third (`res` = 3) the anti-top quark. In order to take effect on the calculation, this subroutine must be called before `compute_process_rcl`.

4.5.2. `compute_running_alphas_rcl (Q,Nf,lp)`

This subroutine can be called to compute the value for α_s at the scale `Q` employing the renormalization-group evolution at `lp` loops (`Q` is of type `real(dp)`, the `integer` argument `lp` can take the value `lp=1` or `lp=2`). The `integer` argument `Nf` selects the flavour scheme according to the following rules (see also Sections 2.2 and 2.3):

- `Nf = -1`:
  The variable flavour scheme is selected, where the number of active flavours contributing to the running of α_s is set to the number of quarks lighter than the scale `Q`.

- `Nf = 3,4,5,6`:
  The fixed $N_{\rm f}$-flavour scheme is selected, where the number of active flavours contributing to the running of α_s is set to $N_{\rm f}$ = `Nf`. In this case `Nf` cannot be smaller than the number of massless quarks (otherwise the code stops).

By calling the subroutine `compute_running_alphas_rcl` the new value of α_s is internally computed by Recola according to the formulas given in Section 2.3. Using this subroutine (or alternatively `set_alphas_rcl`) after the call of `generate_processes_rcl`, the user can assign a different value to α_s at each phase-space point.

4.5.3. `compute_process_rcl (npr,p,order,A2,momenta_check)`

This subroutine computes the Born contribution $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and the one-loop contribution $\mathcal{A}_1^{(\vec{c},\vec{h})}$ to the structure-dressed amplitude $\mathcal{A}^{(\vec{c},\vec{h})}$ (see Section 2.5) of the process with identifier `npr` (of type `integer`) for all values of $\vec{c}$ and $\vec{h}$. It further computes the squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_0$ and $\left(\overline{\mathcal{A}^2}\right)_1$ as defined in Section 2.6. The argument `order` (of type `character`) can take the two values `'LO'` and `'NLO'`. The one-loop contributions are only computed if `order='NLO'` and if the process has been defined at NLO in the call of `define_process_rcl` (see Section 4.3.1). In the computation of the squared

amplitude a sum/average is performed over helicities and colours; for particles that have been defined with a specific helicity, no helicity sum/average is carried out. The results for the amplitudes and squared amplitudes are stored in internal variables (overwriting previous computed results for amplitudes and squared amplitudes for process `npr` up to order `order`) that can be read out by the user with the subroutines `get_amplitude_rcl`, `get_polarized_squared_amplitude_rcl` and `get_squared_amplitude_rcl`.

The input array `p(0:3,1:`l`)` of type `real(dp)` should be filled with the momenta p_i^μ of the l external particles of the process `npr`. The first index of `p` refers to the Lorentz index μ, the second to the particle index i of p_i^μ, with the particles ordered according to their position in the process definition. The optional output argument `A2(0:1)` is a vector of type `real(dp)`. Its two entries, `A2(0)` and `A2(1)`, return the values for $\left(\overline{\mathcal{A}^2}\right)_0$ and $\left(\overline{\mathcal{A}^2}\right)_1$, respectively, with the contributions from all selected powers of α_s summed up.

The optional output argument `momenta_check` (of type `logical`) is set to `.true.` if the momenta provided by the user have passed the checks (see Section 4.1.10), otherwise it is set to `.false.`. Note that, in case the checks are not fulfilled, no computation is performed and the amplitudes are set to zero.

4.5.4. `rescale_process_rcl (npr,order,A2)`

This subroutine allows to adjust the results calculated by `compute_process_rcl` for a new value of α_s, without recomputing the amplitudes (this is done by rescaling LO and NLO amplitudes and recomputing the counterterm for the strong coupling). The user first calls `compute_process_rcl` for a process with identifier `npr` (of type `integer`) at order `order` (of type `character` taking the two values `'LO'` or `'NLO'`), then sets a new value for α_s (by means of `set_alphas_rcl` or `compute_running_alphas_rcl`) and finally calls `rescale_process_rcl` with the same process identifier `npr`.

The call of `rescale_process_rcl` leads to rescaling of the stored results for amplitudes and squared amplitudes for process `npr` up to order `order` and overwrites previous results (`order='NLO'` requires that the original call of `compute_process_rcl` had been evaluated at NLO).

The adjusted results can be obtained with help of the subroutines `get_amplitude_rcl`, `get_polarized_squared_amplitude_rcl` and `get_squared_amplitude_rcl` or via the optional output argument `A2(0:1)` (defined as for `compute_process_rcl`).

4.5.5. `get_colour_configurations_rcl (npr,cols)`

This subroutine writes all colour configurations of the process with identifier `npr` (of type `integer`) to the output variable `cols` (of type `integer`, `allocatable`, `dimension(2)`), which must be unallocated. This variable is allocated by the subroutine, and the dimension of its second argument is set to the total number of colour structures, which depends on the process under consideration. The variable `cols(1:`l`,n)` describes the `n`th colour structure and is a vector of length l. Each position in the vector corresponds to one of the l external particles of process with identifier `npr` (ordered as in the process definition). For colourless particles, incoming quarks and outgoing anti-quarks the corresponding value in `cols` is 0. For all other particles (gluons, outgoing quarks and incoming anti-quarks), the entries are permutations, without repetition, of the positions of the gluons, incoming quarks or outgoing anti-quarks in the process definition. The variable `cols(1:`l`,n)` can be used as the input variable `colour` in the subroutine `get_amplitude_rcl`. See Section 2.5 for details.

Note that Recola does not deallocate the output variable `cols` which has to be taken care of by the user.

4.5.6. `get_helicity_configurations_rcl (npr,hels)`

This subroutine writes all helicity configurations of the process with identifier `npr` (of type `integer`) to the output variable `hels` (of type `integer`, `allocatable`, `dimension(2)`), which must be unallocated. This variable is allocated by the subroutine, and the dimension of its second argument is set to the total number of helicity configurations, which depends on the process under consideration. The variable `hels(1:`l`,n)` describes the `n`th helicity configuration and is a vector of length l. Each position in the vector corresponds to one of the l external particles of process with identifier `npr` (ordered as in the process definition). Its values are the helicities of the particles at the corresponding position in the vector. For scalar particles the value is 0. For fermions the value can be -1 and $+1$ for left and right-handed (anti-)fermions, respectively. For vector bosons the value can be -1, $+1$ and 0 for left-and right-transverse and longitudinal polarizations, respectively. The variable `hels(1:`l`,n)` can be used as the input variable `hel` in the subroutines `get_amplitude_rcl` and `get_polarized_squared_amplitude_rcl`. See Section 2.5 for details.

Note that Recola does not deallocate the output variable `hels` which has to be taken care of by the user.

4.5.7. `get_amplitude_rcl (npr,pow,order,colour,hel,A)`

This subroutine extracts a specific contribution to the structure-dressed amplitude $\mathcal{A}_0^{(\vec{c},\vec{h})}$ or $\mathcal{A}_1^{(\vec{c},\vec{h})}$ (see Section 2.5) of the process with identifier `npr` (of type `integer`), according to the values of the arguments `pow`, `order`, `colour` and `hel`:

- `pow` is of type `integer` and specifies the power of g_s of the contribution.
- `order` represents the loop order of the contribution. It is a variable of type `character` accepting precisely the following values:
  `'LO'`: Born amplitude,
  `'NLO'`: complete one-loop amplitude,
  `'NLO-D4'`: bare 4-dimensional one-loop amplitude,
  `'NLO-CT'`: counterterm contribution to the one-loop amplitude,
  `'NLO-R2'`: rational-term contribution to the one-loop amplitude.
- `colour(1:`l`)` describes the colour structure $\vec{c}$ and is a vector of type `integer` and length l, where each position in the vector corresponds to one of the l external particles of process `npr` (ordered as in the process definition). For colourless particles, incoming quarks and outgoing anti-quarks, the corresponding entry in `colour` must be 0. For all other particles (gluons, outgoing quarks and incoming anti-quarks), the entries must be, without repetition, the positions of the gluons, incoming quarks and outgoing anti-quarks in the process definition. If $k_1, k_2, \ldots, k_n$ ($n \leq l$) are the positions in `colour` that contain non-zero entries, and $m_1, m_2, \ldots, m_n$ are the respective entries, then `colour` represents the colour structure $\delta^{i_{k_1}}_{j_{m_1}} \, \delta^{i_{k_2}}_{j_{m_2}} \, \cdots \, \delta^{i_{k_n}}_{j_{m_n}}$.

  Example:

  Process: `'u u~ -> Z g g'`
  Position: `1 2 3 4 5` .

  The vector `colour` has length 5. The 1st and the 3rd entry corresponding to the up quark and the Z boson must be 0 (incoming quark and colourless particle). The other entries (2nd, 4th and 5th) must be filled with a permutation of the values $\{1, 4, 5\}$, i.e. the positions of the incoming up quark and the gluons.

  Hence, a possible `colour` vector is

`colour(1:5) = [0,1,0,5,4]`,
corresponding to the colour structure $\delta^{i_2}_{j_1}\,\delta^{i_4}_{j_5}\,\delta^{i_5}_{j_4}$.

- `hel(1:`l`)` describes the helicity configuration $\vec{h}$ and is a vector of type `integer` and length l, where each position in the vector corresponds to one of the l external particles of process `npr` (ordered as in the process definition). Its entries represent the helicities of the corresponding particles, according to the conventions of Section 2.9.

  Example:

  Process: `'u g -> W+ d'`

  A possible `hel` vector is
  `hel(1:4) = [-1,+1,-1,-1]`
  corresponding to the following helicities:
  up quark $\rightarrow$ helicity $= -1$,
  gluon $\rightarrow$ helicity $= +1$,
  W^+ $\rightarrow$ helicity $= -1$,
  down quark $\rightarrow$ helicity $= -1$.

The argument `A` of type `complex(dp)` delivers the output consisting of the value of the amplitude for process `npr` as stored in internal variables. Any call of a subroutine of type `compute_`X`_rcl` or `rescale_`X`_rcl` with the same process identifier `npr` (partially) overwrites these internal variables. In order to extract the results from the last process computation, the subroutine `get_amplitude_rcl` should thus be called after `compute_process_rcl` or `rescale_process_rcl` and before a different subroutine call of type `compute_`Y`_rcl` or `rescale_`Y`_rcl` is issued for the same process `npr`. When called at `order='LO'` after `compute_`Y`_rcl` or `rescale_`Y`_rcl` with Y being `colour_correlation`, `spin_correlation`, `spin_colour_correlation` or `all_colour_correlations`, the subroutine `get_amplitude_rcl` will return the LO matrix element entering the calculation of the corresponding correlated squared amplitude.

4.5.8. `get_squared_amplitude_rcl (npr,pow,order,A2)`

This subroutine extracts the computed value of the squared amplitude $\left(\overline{\mathcal{A}^2}\right)_0$ or $\left(\overline{\mathcal{A}^2}\right)_1$ (see Section 2.6 and Section 4.5.3 for details on the definition of squared amplitudes) for the process with identifier `npr` (of type `integer`), according to the values of the arguments `pow` and `order`:

- `pow` is of type `integer` and specifies the power of α_s of the contribution.
- `order` represents the loop order of the contribution. It is variable of type `character` accepting the following values:
  - `'LO'`: squared Born amplitude,
  - `'NLO'`: complete one-loop squared amplitude,
  - `'NLO-D4'`: bare 4-dimensional one-loop squared amplitude,
  - `'NLO-CT'`: counterterm contribution to the one-loop squared amplitude,
  - `'NLO-R2'`: rational-term contribution to the one-loop squared amplitude.

The argument `A2` of type `real(dp)` delivers the output consisting of the value of the squared amplitudes from the last call of `compute_process_rcl` or `rescale_process_rcl` for the process with process number `npr`.

4.5.9. `get_polarized_squared_amplitude_rcl (npr,pow,order,hel,A2h)`

This subroutine extracts the computed value for the contribution $\overline{\mathcal{A}_h^2}$ corresponding to the polarization $h =$ `hel` to the squared amplitude for the process with identifier `npr` (of type `integer`), according to the values of the arguments `pow` and `order`. The definition of the arguments `pow` and `order` is described in Section 4.5.8, while the definition of `hel` is described in Section 4.5.7.

The argument `A2h` of type `real(dp)` delivers the output consisting of the value of the polarized squared amplitude from the last call of `compute_process_rcl` or `rescale_process_rcl` for the process with process number `npr`.

4.5.10. `compute_colour_correlation_rcl (npr,p,i1,i2,A2cc,momenta_check)`

This subroutine computes the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and the specific colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\mathrm{c}}(i_1,i_2)$ for the pair (i_1,i_2) of external particles (see Section 2.7 for details) for the process with identifier `npr` (of type `integer`). The `integer` arguments `i1` and `i2` denote the position identifiers of the respective external particles (ordered as in the process definition). In the computation of the squared amplitude a sum/average is

performed over helicities and colours (see Section 2.6 for details on the definition of squared amplitudes); for particles that have been defined with a specific helicity, no helicity sum/average is performed.

The definition of the input argument `p(0:3,1:`l`)` of type `real(dp)` is described in Section 4.5.3.

The results for the LO amplitudes and the colour-correlated squared amplitudes are stored in internal variables (overwriting previously computed results for LO amplitudes and colour-correlated squared amplitudes for process `npr` and particle pair `(i1,i2)`). The values of the colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i_1,i_2)$ can be read out by the user with the subroutine `get_colour_correlation_rcl`. The optional output argument `A2cc` (of type `real(dp)`) returns the value for the colour-correlated squared amplitude, with the contributions from all selected powers of α_s summed up.

The optional output argument `momenta_check` (of type `logical`) behaves as explained at the end of Section 4.5.3.

4.5.11. `compute_all_colour_correlations_rcl (npr,p,momenta_check)`

This subroutine computes the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and the colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i,j)$ for all pairs (i,j) of external coloured particles (see Section 2.7 for details) for the process with identifier `npr` (of type `integer`). In the computation of the squared amplitude a sum/average is performed over helicities and colours (see Section 2.6 for details on the definition of squared amplitudes); for particles that have been defined with a specific helicity, no helicity sum/average is performed.

The definition of input argument `p(0:3,1:`l`)` of type `real(dp)` is described in Section 4.5.3.

The results for the LO amplitudes and the colour-correlated squared amplitudes are stored in internal variables (overwriting previously computed results for LO amplitudes and colour-correlated squared amplitudes for process `npr`). The values of the colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i,j)$ can be read out by the user with the subroutine `get_colour_correlation_rcl`.

The optional output argument `momenta_check` (of type `logical`) behaves as explained at the end of Section 4.5.3.

4.5.12. `rescale_colour_correlation_rcl (npr,i1,i2,A2cc)`

This subroutine can be employed to rescale the results calculated by the subroutines `compute_all_colour_correlations_rcl` or

`compute_colour_correlation_rcl` to a different value of α_s. To this end, the user first sets a new value for α_s (by means of `set_alphas_rcl` or `compute_running_alphas_rcl`) and then calls `rescale_colour_correlation_rcl` with a process identifier `npr` and position identifiers `i1` and `i2` for the respective external particles (`npr`, `i1`, and `i2` are of type `integer`). This leads to a rescaling of the stored results for the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and for the colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i_1,i_2)$ for the process `npr` and the pair `(i1,i2)` of coloured external particles, overwriting previous results. The rescaled values of the colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i_1,i_2)$ can be either obtained with help of the subroutine `get_colour_correlation_rcl` or via the optional output argument `A2cc` (defined as for `compute_colour_correlation_rcl`).

4.5.13. `rescale_all_colour_correlations_rcl (npr)`

This subroutine can be used to rescale the results calculated by the subroutine `compute_all_colour_correlations_rcl` or `compute_colour_correlation_rcl` to a different value of α_s. To this end, the user first sets a new value for α_s (by means of `set_alphas_rcl` or `compute_running_alphas_rcl`) and then calls `rescale_all_colour_correlations_rcl` with a process identifier `npr` (of type `integer`). This leads to a rescaling of the stored results for the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and for the colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i,j)$ for the process `npr` and all pairs (i,j) of external coloured particles, overwriting previous results. The rescaled values of the colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i,j)$ can be obtained with help of the subroutine `get_colour_correlation_rcl`.

4.5.14. `get_colour_correlation_rcl (npr,pow,i1,i2,A2cc)`

This subroutine extracts the computed value of the LO colour-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_{\rm c}(i_1,i_2)$ for the pair (i_1,i_2) of external particles (see Section 2.7 for details) for the process with identifier `npr` (of type `integer`). The `integer` arguments `i1` and `i2` denote the position identifiers of the respective external particles (ordered as in the process definition). The `integer` argument `pow` specifies the power of α_s of the contribution to be extracted.

The argument `A2cc` of type `real(dp)` delivers the output of the subroutine consisting of the value of the LO colour-correlated

squared amplitudes for particle pair (i1,i2) from the last call of compute_colour_correlation_rcl or rescale_colour_correlation_rcl for the process with process number npr and external coloured particles (i1,i2), or from the last call of compute_all_colour_correlations_rcl or rescale_all_colour_correlations_rcl for the process with process number npr.

4.5.15. compute_spin_correlation_rcl (npr,p,j,v,A2sc,momenta_check)

This subroutine computes the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and the spin-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_{\rm s}(j)$ for the photon j in the process with identifier npr (of type integer), using as polarization vector for the photon j the four-vector v provided by the user (see Section 2.7 for details). The integer argument j denotes the position identifier of the respective photon (following the order of external particles in the process definition). The vector v(0:3) (of type complex(dp)) denotes the user-defined four-vector substituting the polarization vector of photon j. In the computation of the squared amplitude a sum/average is performed over helicities and colours (see Section 2.6 for details on the definition of squared amplitudes); for particles that have been defined with a specific helicity as well as for the photon j, no helicity sum/average is performed.

The results for the LO amplitudes and the spin-correlated squared amplitudes are stored in internal variables (overwriting previously computed results for LO amplitudes and spin-correlated squared amplitudes for process npr). The values of the spin-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm s}(j)$ can be read out by the user with the subroutine get_spin_correlation_rcl. The optional output argument A2sc (of type real(dp)) returns the value for the spin-correlated squared amplitude with the contributions from all selected powers of α_s summed up.

The optional output argument momenta_check (of type logical) behaves as explained at the end of Section 4.5.3.

4.5.16. rescale_spin_correlation_rcl (npr,j,v,A2sc)

This subroutine allows the user to rescale the results calculated by compute_spin_correlation_rcl to a different value of α_s. To this end, the user first sets a new value for α_s (by means of set_alphas_rcl or compute_running_alphas_rcl) and then calls

`rescale_spin_correlation_rcl` with a process identifier `npr` and an identifier `j` for the position of the external photon (`npr` and `j` are of type `integer`). This leads to a rescaling of the stored results for the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and for the spin-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm s}(j)$ for the process `npr` and the photon `j`, overwriting previous results. The vector `v(0:3)` (of type `complex(dp)`) plays the same role as in `compute_spin_correlation_rcl`. While the rescaling of the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ is not affected by the value of `v`, the spin-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\rm s}(j)$ are computed by `rescale_spin_correlation_rcl` for the current value of `v`, independently of previous computations of spin correlations. The rescaled result can be either obtained with help of the subroutine `get_spin_correlation_rcl` or via the optional output argument `A2sc` (defined as for `compute_spin_correlation_rcl`).

4.5.17. `get_spin_correlation_rcl (npr,pow,A2sc)`

This subroutine extracts the computed value of the LO spin-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_{\rm s}$ (see Section 2.7 for details) for the process with identifier `npr` (of type `integer`). The `integer` argument `pow` specifies the power of α_s of the contribution to be extracted.

The argument `A2sc` of type `real(dp)` delivers the output of the subroutine consisting of the value of the LO spin-correlated squared amplitudes from the last call of `compute_spin_correlation_rcl` or `rescale_spin_correlation_rcl` for the process with process number `npr`.

4.5.18. `compute_spin_colour_correlation_rcl (npr,p,i1,i2,v,A2scc,momenta_check)`

This subroutine computes the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and the specific colour-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_{\rm sc}(i_1,i_2)$ for the pair (i_1,i_2) of an external gluon i_1 and a spectator i_2 (see Section 2.7 for details) for the process with identifier `npr` (of type `integer`). The `integer` arguments `i1` and `i2` denote the position identifiers of the respective external particles (ordered as in the process definition). The polarization vector of the gluon i_1 is substituted by the four-vector `v(0:3)` (of type `complex(dp)`) provided by the user (see Section 2.7 for details). In the computation of the squared amplitude a sum/average is performed over helicities and colours (see Section 2.6 for details on the definition of squared amplitudes); for particles that have been defined with a specific helicity and for the gluon i_1, no helicity sum/average

is performed.

The definition of input argument `p(0:3,1:`l`)` of type `real(dp)` is described in Section 4.5.3.

The results for the LO amplitudes and the spin–colour-correlated squared amplitudes are stored in internal variables (overwriting previously computed results for LO amplitudes and spin–colour-correlated squared amplitudes for process `npr` and particle pair `(i1,i2)`). The values of the spin–colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\text{sc}}(i_1, i_2)$ can be read out by the user with the subroutine `get_spin_colour_correlation_rcl`. The optional output argument `A2scc` (of type `real(dp)`) returns the value for the spin–colour-correlated squared amplitude for particle pair `(i1,i2)` with the contributions from all selected powers of α_s summed up.

The optional output argument `momenta_check` (of type `logical`) behaves as explained at the end of Section 4.5.3.

4.5.19. `rescale_spin_colour_correlation_rcl (npr,i1,i2,v,A2scc)`

This subroutine can be used to rescale the results calculated by `compute_spin_colour_correlation_rcl` to a different value of α_s. To this end, the user first sets a new value for α_s (by means of `set_alphas_rcl` or `compute_running_alphas_rcl`) and then calls `rescale_spin_colour_correlation_rcl` with a process identifier `npr` and position identifiers `i1` for the gluon and `i2` for the spectator (`npr`, `i1`, and `i2` are of type `integer`). This leads to a rescaling of the stored results for the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ and for the spin–colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\text{sc}}(i_1, i_2)$ for the process `npr` and the particle pair `(i1,i2)`, overwriting previous results. The vector `v(0:3)` (of type `complex(dp)`) plays the same role as in `compute_spin_colour_correlation_rcl`. While the rescaling of the LO amplitudes $\mathcal{A}_0^{(\vec{c},\vec{h})}$ is not affected by the value of `v`, the spin–colour-correlated squared amplitudes $\left(\overline{\mathcal{A}^2}\right)_{\text{sc}}(i_1, i_2)$ are computed by `rescale_spin_correlation_rcl` for the current value of `v`, independently of previous computations of spin–colour correlations. The rescaled result can be either obtained with help of the subroutine `get_spin_colour_correlation_rcl` or via the optional output argument `A2scc` (defined as for `compute_spin_colour_correlation_rcl`).

4.5.20. `get_spin_colour_correlation_rcl (npr,pow,i1,i2,A2scc)`

This subroutine extracts the computed value of the LO spin–colour-correlated squared amplitude $\left(\overline{\mathcal{A}^2}\right)_{\rm sc}(i_1,i_2)$ for the pair (i_1,i_2) of gluon i_1 and spectator i_2 (see Section 2.7 for details) for the process with identifier `npr` (of type `integer`). The `integer` arguments `i1` and `i2` denote the position identifiers of the respective external particles (ordered as in the process definition). The `integer` argument `pow` specifies the power of α_s of the contribution to be extracted. The argument `A2scc` of type `real(dp)` delivers the output of the subroutine consisting of the value of the LO spin–colour-correlated squared amplitudes as stored in the internal variables by the last call of `compute_spin_colour_correlation_rcl`, or `rescale_spin_colour_correlation_rcl` for the process with process number `npr` and particle pair `(i1,i2)`.

4.5.21. `get_momenta_rcl (npr,p)`

This subroutine extracts the momenta of the process with identifier `npr` (of type `integer`), stored from the last call of a subroutine of type `compute_..._rcl` for process `npr`. The output array `p` is of type `real(dp)` and has the format `p(0:3,1:`l`)`, where l is the number of external particles of process `npr`.

Note that RECOLA adjusts the momenta provided by the user if it detects violation of momentum conservation or mass-shell conditions.

4.5.22. `set_TIs_required_accuracy_rcl (acc)`

This subroutine sets the required accuracy for TIs to the value `acc` (of type `real(dp)`). This parameter is passed to COLLIER as target accuracy η_{req} (see Ref. [48] for details).

4.5.23. `get_TIs_required_accuracy_rcl (acc)`

This subroutine extracts the value of the required accuracy η_{req} for TIs from COLLIER (see Ref. [48] for details) and returns it as value of the output variable `acc` (of type `real(dp)`).

4.5.24. `set_TIs_critical_accuracy_rcl (acc)`

This subroutine sets the critical accuracy for TIs to the value `acc` (of type `real(dp)`). This parameter is passed to COLLIER as critical accuracy η_{crit} (see Ref. [48] for details).

4.5.25. `get_TIs_critical_accuracy_rcl (acc)`

This subroutine extracts the value of the critical accuracy η_{crit} for TIs from COLLIER (see Ref. [48] for details) and returns it as value of the output variable `acc` (of type `real(dp)`).

4.5.26. `get_TIs_accuracy_flag_rcl (flag)`

This subroutine extracts the value of the accuracy flag $\sigma_{\rm acc}$ for TIs from COLLIER (see Ref. [48] for details). The output variable `flag` (of type `integer`) returns global information on the accuracy of the TIs evaluated in the last call of `compute_process_rcl`:

- `flag = 0`: For all TIs, the accuracy is estimated to be better than the required value.
- `flag = -1`: For at least one TI, the accuracy is estimated to be worse than the required value, but for all TIs, the accuracy is estimated to be better than the critical value.
- `flag = -2`: For at least one TI, the accuracy is estimated to be worse than the critical values.

The value of variable `flag` is determined based on internal uncertainty estimations performed by COLLIER.

4.6. Warning summary: `print_warning_summary_rcl`

The subroutine `print_warning_recola_rcl` prints a summary of all RECOLA warnings that showed up during the run and should be called at the end of a Monte Carlo run. This summary allows to judge whether the results are to be trusted or not.

4.7. Reset: `reset_recola_rcl`

The file `reset.f90` only contains the single subroutine `reset_recola_rcl` which can be called to free memory and to allow for the definition of a new set of processes in the same run of the program. A call of this subroutine deallocates all global allocatable arrays internally generated by RECOLA and restores the initialization values for a bunch of internal variables, allowing to restart the application of RECOLA with step 1 or 2 of the sequence defined at the beginning of Section 4. The input variables in `input.f90` keep their actual values. The call of `reset_recola_rcl` resets the name of the output file to the default value `output.rcl`.

4.8. C++ interface

The RECOLA package includes a C++ interface with the same naming conventions for the subroutines as given in Sections 4.2–4.7.[9] 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 RECOLA in a C++ program the RECOLA header file needs to be included as follows:

```
#include "recola.h"
```

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.h"
using namespace Recola
```

The FORTRAN95 subroutines in Sections 4.2–4.7 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` in Section 4.5.3, 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.

[9] This interface is used to link RECOLA to SHERPA.

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 4.2.20), using the default value for $\alpha(M_Z)$ which is hard-coded in RECOLA. Providing an explicit argument as

```
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 Sections 3.1 and 3.2, 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`, or `cdemo3_rcl`. The content of each `cdemo` file is identical to the content of the corresponding `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),
- `set_gs_power_rcl` (Section 4.3.2),

are replaced by the following ones:

4.8.1. `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.

4.8.2. `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`.

4.8.3. `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` and using the default, i.e. complex, vector-boson masses.

4.8.4. `use_gfermi_scheme_complex_and_set_gfermi_rcl(g)`

This C++ function calls the subroutine `use_gfermi_scheme_rcl(g, a, massesgf)` (FORTRAN95), passing the value g of type `double` and `massesgf` $= 1$, i.e. using complex vector-boson masses.

4.8.5. `use_gfermi_scheme_real_and_set_gfermi_rcl(g)`

This C++ function calls the subroutine `use_gfermi_scheme_rcl(g, a, massesgf)` (FORTRAN95), passing the value g of type `double` and `massesgf` $= 0$, i.e. using real vector-boson masses.

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

This C++ function calls the subroutine `set_gs_power_rcl(npr, gsarray)` (FORTRAN95), passing the value for `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.

4.8.7. Missing subroutines

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

5. Conclusions

The FORTRAN-based library RECOLA calculates amplitudes and squared amplitudes in the Standard Model of particle physics including QCD and the electroweak interaction at the tree and one-loop level with no a-priori restriction on the particle multiplicities. Amplitudes can be obtained for specific colour structures and helicities and squared amplitudes with or without

summation/average over helicities. Moreover, colour- and spin-correlated leading-order squared amplitudes for dipole subtraction are provided.

Renormalization is performed in the complex-mass scheme, or alternatively in the on-shell scheme, and various renormalization schemes are supported for the electromagnetic coupling. For the strong coupling, fixed or dynamical N_f-flavour schemes are available. Infrared singularites can be regularized dimensionally or with infinitesimal fermion and photon/gluon masses. The code allows to select contributions involving specific resonances.

The present version of the code is restricted to the Standard Model in the 't Hooft–Feynman gauge. A version for more general theories is in preparation.

6. Acknowledgements

We thank B. Biedermann, R. Feger, and M. Pellen for performing various checks of the code. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under reference number DE 623/2-1. The work of L.H. was supported by the grants FPA2013-46570-C2-1-P and 2014-SGR-104, and partially by the Spanish MINECO under the project MDM-2014-0369 of ICCUB (Unidad de Excelencia "María de Maeztu"). The work of S.U. was supported in part by the European Commission through the 'HiggsTools' Initial Training Network PITN-GA-2012-316704.

Appendix A. Explicit representations for spinors and polarization vectors

Here we list the explicit expressions of the spinors and polarization vectors used in RECOLA, which are in the chiral representation for the Dirac matrices:

- Spinors for massive fermions:

$$u_+(p) = \frac{1}{r}\begin{pmatrix} a_+ b_+ \\ s a_+ b_- \hat{p}_+ \\ -s a_- b_+ \\ -a_- b_- \hat{p}_+ \end{pmatrix}, \quad u_-(p) = \frac{1}{r}\begin{pmatrix} a_- b_- \hat{p}_- \\ -s a_- b_+ \\ -s a_+ b_- \hat{p}_- \\ a_+ b_+ \end{pmatrix},$$

$$v_+(p) = \frac{1}{r}\begin{pmatrix} -a_- b_- \hat{p}_- \\ s a_- b_+ \\ -s a_+ b_- \hat{p}_- \\ a_+ b_+ \end{pmatrix}, \quad v_-(p) = \frac{1}{r}\begin{pmatrix} a_+ b_+ \\ s a_+ b_- \hat{p}_+ \\ s a_- b_+ \\ a_- b_- \hat{p}_+ \end{pmatrix},$$

$$
\begin{aligned}
\bar{u}_+(p) &= \frac{1}{r}\Big(sa_-b_+,\ a_-b_-\hat{p}_-,\ -a_+b_+,\ -sa_+b_-\hat{p}_-\Big),\\
\bar{u}_-(p) &= \frac{1}{r}\Big(sa_+b_-\hat{p}_+,\ -a_+b_+,\ -a_-b_-\hat{p}_+,\ sa_-b_+\Big),\\
\bar{v}_+(p) &= \frac{1}{r}\Big(sa_+b_-\hat{p}_+,\ -a_+b_+,\ a_-b_-\hat{p}_+,\ -sa_-b_+\Big),\\
\bar{v}_-(p) &= \frac{1}{r}\Big(-sa_-b_+,\ -a_-b_-\hat{p}_-,\ -a_+b_+,\ -sa_+b_-\hat{p}_-\Big), \qquad \text{(A.1)}
\end{aligned}
$$

with

$$
r=\sqrt{2|\vec{p}|},\qquad a_\pm=\sqrt{|p_0|\pm|\vec{p}|},\qquad b_\pm=\sqrt{|\vec{p}|\pm s\,p_z},\qquad s=\mathrm{sign}(p_0),
$$

$$
\hat{p}_\pm=\frac{p_\pm}{p_{_T}},\qquad p_\pm=p_x\pm \mathrm{i}\,p_y,\qquad p_{_T}=\sqrt{p_x^2+p_y^2},\qquad |\vec{p}|=\sqrt{p_{_T}^2+p_z^2}. \qquad \text{(A.2)}
$$

- Spinors for massless fermions:

$$
u_+(p)=v_-(p)=\begin{pmatrix} b_+\\ sb_-\hat{p}_+\\ 0\\ 0\end{pmatrix},\qquad u_-(p)=v_+(p)=\begin{pmatrix} 0\\ 0\\ -sb_-\hat{p}_-\\ b_+\end{pmatrix},
$$

$$
\begin{aligned}
\bar{u}_+(p)=\bar{v}_-(p)&=\Big(0,\ 0,\ -b_+,\ -sb_-\hat{p}_-\Big),\\
\bar{u}_-(p)=\bar{v}_+(p)&=\Big(sb_-\hat{p}_+,\ -b_+,\ 0,\ 0\Big). \qquad \text{(A.3)}
\end{aligned}
$$

- Transverse polarization vectors (for massless and massive vector bosons):

$$
\epsilon_\pm=\frac{1}{\sqrt{2}\,|\vec{p}|\,p_{_T}}\Big(0,\ \mp s\,p_x\,p_z+\mathrm{i}\,p_y\,|\vec{p}|,\ \mp s\,p_y\,p_z-\mathrm{i}\,p_x\,|\vec{p}|,\ \pm s\,p_{_T}^2\Big). \qquad \text{(A.4)}
$$

- Longitudinal polarization vectors (for massive vector bosons with mass M):

$$
\epsilon_0=\frac{1}{M\,|\vec{p}|}\Big(|\vec{p}|^{\,2},\ p_x\,p_0,\ p_y\,p_0,\ p_z\,p_0\Big). \qquad \text{(A.5)}
$$

Appendix B. Checks

A variety of processes has been checked at LO and NLO with the in-house code Pole [72] and the code OpenLoops [14]. The typical agreement for NLO matrix elements ranges from 10^{-14} for processes with 4 external legs to 10^{-10} for processes with 6 external legs in the comparison with Pole, and from 10^{-12} to 10^{-8} in the comparison with OpenLoops. Note that the precision of the NLO matrix element strongly depends on the phase-space point for which it is evaluated, and is limited by the accuracy of the tensor intagrals, compared to which numerical uncertainties from the tensor coefficients calculated by Recola are completely negligible.

The following processes have been checked with the code Pole in a Monte Carlo integration for the calculations in Refs. [16, 33, 35, 36]:

- 5-leg processes (all powers of α_s at LO, $\mathcal{O}(\alpha_s^2\alpha)$ and $\mathcal{O}(\alpha_s\alpha^2)$ at NLO):
  $\mathrm{u\,g} \to \mathrm{u\,g\,Z}$
  $\mathrm{d\,g} \to \mathrm{d\,g\,Z}$,

- 6-leg process (all powers of α_s at LO, $\mathcal{O}(\alpha_s^2\alpha)$ and $\mathcal{O}(\alpha_s\alpha^2)$ at NLO):
  $\mathrm{u\,d} \to \mathrm{u\,d\,e^+\,e^-}$,

- 6-leg processes (all powers of α_s at LO and NLO):
  $\mathrm{u\,\bar{u}} \to \mu^+\,\mu^-\,\mathrm{e^+\,e^-}$
  $\mathrm{u\,\bar{u}} \to \mu^+\,\mu^-\,\mu^+\,\mu^-$
  $\mathrm{b\,\bar{b}} \to \mu^+\,\mu^-\,\mu^+\,\mu^-$.

The following processes have been checked for single phase-space points:

- 4-leg processes (EW) checked with the code Pole:
  $\bar{\mathrm{u}}\,\mathrm{u} \to \bar{\nu}_e\,\nu_e$
  $\mathrm{u\,\bar{d}} \to \nu_e\,\mathrm{e^+}$
  $\mathrm{e^+\,e^-} \to \bar{\nu}_e\,\nu_e$
  $\mathrm{e^+\,e^-} \to \mathrm{W^+\,W^-}$
  $\mathrm{e^+\,e^-} \to \mathrm{Z\,H}$,

- 4-leg processes (QCD) checked with the code OpenLoops:
  $\mathrm{u\,\bar{d}} \to \mathrm{W^+\,g}$
  $\mathrm{g\,g} \to \mathrm{g\,g}$
  $\mathrm{b\,\bar{b}} \to \mathrm{t\,\bar{t}}$
  $\mathrm{g\,g} \to \mathrm{b\,\bar{b}}$,

- 4-leg processes (EW+QCD) checked with the code Pole:
  $\bar{\mathrm{d}}\,\mathrm{d} \to \bar{\mathrm{u}}\,\mathrm{u}$
  $\mathrm{u}\,\bar{\mathrm{d}} \to \mathrm{W}^+\,\mathrm{H}$,

- 5-leg processes (EW) checked with the code Pole:
  $\mathrm{u}\,\bar{\mathrm{d}} \to \mathrm{e}^+\,\nu_e\,\gamma$,

- 5-leg processes (QCD) checked with the code OpenLoops:
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{W}^+\,\mathrm{W}^-\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{Z}\,\mathrm{Z}\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{Z}\,\gamma\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \gamma\,\gamma\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{d}} \to \mathrm{W}^+\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{d}} \to \mathrm{W}^+\,\mathrm{t}\,\bar{\mathrm{t}}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{Z}\,\mathrm{t}\,\bar{\mathrm{t}}$
  $\mathrm{d}\,\bar{\mathrm{d}} \to \mathrm{Z}\,\mathrm{t}\,\bar{\mathrm{t}}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{W}^+\,\mathrm{b}\,\bar{\mathrm{t}}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{Z}\,\mathrm{t}\,\bar{\mathrm{t}}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{Z}\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{g}\,\mathrm{t}\,\bar{\mathrm{t}}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{g}\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{d}\,\bar{\mathrm{d}} \to \mathrm{d}\,\bar{\mathrm{d}}\,\mathrm{g}$
  $\mathrm{d}\,\bar{\mathrm{d}} \to \mathrm{t}\,\bar{\mathrm{t}}\,\mathrm{g}$
  $\mathrm{b}\,\bar{\mathrm{b}} \to \mathrm{t}\,\bar{\mathrm{t}}\,\mathrm{g}$,

- 6-leg processes (QCD) checked with the code OpenLoops:
  $\mathrm{u}\,\bar{\mathrm{d}} \to \mathrm{W}^+\,\mathrm{g}\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{Z}\,\mathrm{g}\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{W}^+\,\mathrm{W}^-\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{Z}\,\mathrm{Z}\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{d}\,\bar{\mathrm{d}} \to \mathrm{t}\,\bar{\mathrm{t}}\,\mathrm{b}\,\bar{\mathrm{b}}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{t}\,\bar{\mathrm{t}}\,\mathrm{b}\,\bar{\mathrm{b}}$
  $\mathrm{u}\,\bar{\mathrm{u}} \to \mathrm{u}\,\bar{\mathrm{u}}\,\mathrm{u}\,\bar{\mathrm{u}}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{u}\,\bar{\mathrm{u}}\,\mathrm{u}\,\bar{\mathrm{u}}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{u}\,\bar{\mathrm{u}}\,\mathrm{d}\,\bar{\mathrm{d}}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{u}\,\bar{\mathrm{u}}\,\mathrm{g}\,\mathrm{g}$
  $\mathrm{g}\,\mathrm{g} \to \mathrm{t}\,\bar{\mathrm{t}}\,\mathrm{g}\,\mathrm{g}$.

Thereby (EW) refers to contributions to the squared amplitudes with minimal power of α_s at LO and at NLO, (QCD) refers to contributions to

the squared amplitudes with maximal power of α_s at LO and at NLO and (EW+QCD) refers to the sum of all contributions to the squared amplitudes at LO and at NLO.

References

[1] J. M. Campbell, et al., Working Group Report: Quantum Chromodynamics, in: Proceedings, 2013 Community Summer Study on the Future of U.S. Particle Physics: Snowmass on the Mississippi (CSS2013): Minneapolis, MN, USA, July 29-August 6, 2013, 2013. arXiv:1310.5189.
URL https://inspirehep.net/record/1261432/files/arXiv:1310.5189.pdf

[2] J. R. Andersen, et al., Les Houches 2013: Physics at TeV Colliders, Standard Model Working Group Report (2014). arXiv:1405.1067.

[3] Z. Bern, et al., The NLO multileg working group, Summary report (2008). arXiv:0803.0494.

[4] T. Binoth, et al., The SM and NLO Multileg Working Group: Summary report (2010). arXiv:1003.1241.

[5] J. Alcaraz Maestre, et al., The SM and NLO Multileg and SM MC Working Groups: Summary Report (2012). arXiv:1203.6803.

[6] Z. Bern, L. J. Dixon, D. C. Dunbar, D. A. Kosower, One-loop n-point gauge theory amplitudes, unitarity and collinear limits, Nucl. Phys. B425 (1994) 217–260. arXiv:hep-ph/9403226, doi:10.1016/0550-3213(94)90179-1.

[7] Z. Bern, L. J. Dixon, D. C. Dunbar, D. A. Kosower, Fusing gauge theory tree amplitudes into loop amplitudes, Nucl. Phys. B435 (1995) 59–101. arXiv:hep-ph/9409265, doi:10.1016/0550-3213(94)00488-Z.

[8] R. Britto, F. Cachazo, B. Feng, Generalized unitarity and one-loop amplitudes in N=4 super-Yang-Mills, Nucl. Phys. B725 (2005) 275–305. arXiv:hep-th/0412103, doi:10.1016/j.nuclphysb.2005.07.014.

[9] G. Ossola, C. G. Papadopoulos, R. Pittau, Reducing full one-loop amplitudes to scalar integrals at the integrand level, Nucl. Phys. B763 (2007) 147–169. arXiv:hep-ph/0609007.

[10] R. K. Ellis, W. Giele, Z. Kunszt, A numerical unitarity formalism for evaluating one-loop amplitudes, JHEP 0803 (2008) 003. arXiv:0708.2398, doi:10.1088/1126-6708/2008/03/003.

[11] W. T. Giele, Z. Kunszt, K. Melnikov, Full one-loop amplitudes from tree amplitudes, JHEP 0804 (2008) 049. arXiv:0801.2237, doi:10.1088/1126-6708/2008/04/049.

[12] R. K. Ellis, W. T. Giele, Z. Kunszt, K. Melnikov, Masses, fermions and generalized D-dimensional unitarity, Nucl. Phys. B822 (2009) 270–282. arXiv:0806.3467, doi:10.1016/j.nuclphysb.2009.07.023.

[13] 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.

[14] 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.

[15] A. van Hameren, Multi-gluon one-loop amplitudes using tensor integrals, JHEP 0907 (2009) 088. arXiv:0905.1005, doi:10.1088/1126-6708/2009/07/088.

[16] 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.

[17] Z. Nagy, D. E. Soper, General subtraction method for numerical calculation of one loop QCD matrix elements, JHEP 09 (2003) 055. arXiv:hep-ph/0308127, doi:10.1088/1126-6708/2003/09/055.

[18] S. Becker, C. Reuschle, S. Weinzierl, Numerical NLO QCD calculations, JHEP 12 (2010) 013. arXiv:1010.4187, doi:10.1007/JHEP12(2010)013.

[19] G. Duplancic, B. Klajn, Direct numerical approach to one-loop amplitudes, Phys. Rev. D95 (1) (2017) 016002. arXiv:1604.07022, doi:10.1103/PhysRevD.95.016002.

[20] 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.

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

[22] B. Chokoufe Nejad, T. Hahn, J.-N. Lang, E. Mirabella, FormCalc 8: Better algebra and vectorization, J. Phys. Conf. Ser. 523 (2014) 012050. arXiv:1310.0274, doi:10.1088/1742-6596/523/1/012050.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] S. Kallweit, J. M. Lindert, S. Pozzorini, M. Schönherr, P. Maierhöfer, NLO QCD+EW predictions for V+jets including off-shell vector-boson decays and multijet merging, JHEP 04 (2016) 021. arXiv:1511.08692, doi:10.1007/JHEP04(2016)021.

[30] S. Frixione, V. Hirschi, D. Pagani, H. S. Shao, M. Zaro, Weak corrections to Higgs hadroproduction in association with a top-quark pair, JHEP 09 (2014) 065. arXiv:1407.0823, doi:10.1007/JHEP09(2014)065.

[31] 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.

[32] 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.

[33] A. Denner, L. Hofer, A. Scharf, S. Uccirati, Electroweak corrections to lepton-pair production in association with two hard jets at the LHC, JHEP 01 (2015) 094. arXiv:1411.0916, doi:10.1007/JHEP01(2015)094.

[34] B. Biedermann, et al., Next-to-leading-order electroweak corrections to $pp \to W^+W^- \to 4$ leptons at the LHC, JHEP 06 (2016) 065. arXiv:1605.03419, doi:10.1007/JHEP06(2016)065.

[35] B. Biedermann, A. Denner, S. Dittmaier, L. Hofer, B. Jäger, Electroweak corrections to pp $\to \mu^+\mu^-\mathrm{e}^+\mathrm{e}^- + X$ at the LHC – a Higgs background study, Phys. Rev. Lett. 116 (16) (2016) 161803. arXiv:1601.07787, doi:10.1103/PhysRevLett.116.161803.

[36] B. Biedermann, A. Denner, S. Dittmaier, L. Hofer, B. Jäger, Next-to-leading-order electroweak corrections to the production of four charged leptons at the LHC, JHEP 01 (2017) 033. arXiv:1611.05338, doi:10.1007/JHEP01(2017)033.

[37] A. Denner, M. Pellen, NLO electroweak corrections to off-shell top-antitop production with leptonic decays at the LHC, JHEP 08 (2016) 155. arXiv:1607.05571, doi:10.1007/JHEP08(2016)155.

[38] B. Biedermann, A. Denner, M. Pellen, Large electroweak corrections to vector-boson scattering at the Large Hadron ColliderarXiv:1611.02951.

[39] A. Denner, R. Feger, NLO QCD corrections to off-shell top-antitop production with leptonic decays in association with a Higgs boson at the LHC, JHEP 11 (2015) 209. arXiv:1506.07448, doi:10.1007/JHEP11(2015)209.

[40] G. Ossola, C. G. Papadopoulos, R. Pittau, CutTools: a program implementing the OPP reduction method to compute one-loop amplitudes, JHEP 0803 (2008) 042. arXiv:0711.3596, doi:10.1088/1126-6708/2008/03/042.

[41] P. Mastrolia, G. Ossola, T. Reiter, F. Tramontano, Scattering amplitudes from unitarity-based reduction algorithm at the integrand-level, JHEP 1008 (2010) 080. arXiv:1006.0710, doi:10.1007/JHEP08(2010)080.

[42] T. Peraro, Ninja: Automated Integrand Reduction via Laurent Expansion for One-Loop Amplitudes, Comput. Phys. Commun. 185 (2014) 2771–2797. arXiv:1403.1229, doi:10.1016/j.cpc.2014.06.017.

[43] G. van Oldenborgh, FF: A Package to evaluate one-loop Feynman diagrams, Comput. Phys. Commun. 66 (1991) 1–15. doi:10.1016/0010-4655(91)90002-3.

[44] T. Hahn, M. Perez-Victoria, Automatized one-loop calculations in four-dimensions and D-dimensions, Comput. Phys. Commun. 118 (1999) 153–165. arXiv:hep-ph/9807565, doi:10.1016/S0010-4655(98)00173-8.

[45] J. Fleischer, T. Riemann, A Complete algebraic reduction of one-loop tensor Feynman integrals, Phys. Rev. D83 (2011) 073004. arXiv:1009.4436, doi:10.1103/PhysRevD.83.073004.

[46] G. Cullen, et al., Golem95C: A library for one-loop integrals with complex masses, Comput. Phys. Commun. 182 (2011) 2276–2284. arXiv:1101.5595, doi:10.1016/j.cpc.2011.05.015.

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

[48] 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.

[49] H. H. Patel, Package-X: A Mathematica package for the analytic calculation of one-loop integrals, Comput. Phys. Commun. 197 (2015) 276–290. arXiv:1503.01469, doi:10.1016/j.cpc.2015.08.017.

[50] R. K. Ellis, G. Zanderighi, Scalar one-loop integrals for QCD, JHEP 0802 (2008) 002. arXiv:0712.1851, doi:10.1088/1126-6708/2008/02/002.

[51] A. van Hameren, OneLOop: For the evaluation of one-loop scalar functions, Comput. Phys. Commun. 182 (2011) 2427–2438. arXiv:1007.4716, doi:10.1016/j.cpc.2011.06.011.

[52] A. Denner, S. Dittmaier, M. Roth, D. Wackeroth, Predictions for all processes $e^+e^- \to 4$ fermions $+\gamma$, Nucl. Phys. B560 (1999) 33–65. arXiv:hep-ph/9904472, doi:10.1016/S0550-3213(99)00437-X.

[53] A. Denner, S. Dittmaier, M. Roth, L. Wieders, Electroweak corrections to charged-current $e^+e^- \to 4$ fermion processes: Technical details and further results, Nucl. Phys. B724 (2005) 247–294, erratum-ibid. **B854** (2012) 504–507. arXiv:hep-ph/0505042, doi:10.1016/j.nuclphysb.2005.06.033,10.1016/j.nuclphysb.2011.09.001.

[54] 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.

[55] Z. Bern, et al., Ntuples for NLO events at hadron colliders, Comput. Phys. Commun. 185 (2014) 1443–1460. arXiv:1310.7439, doi:10.1016/j.cpc.2014.01.011.

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

[57] 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.

[58] 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.

[59] 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.

[60] G. Ossola, C. G. Papadopoulos, R. Pittau, On the rational terms of the one-loop amplitudes, JHEP 05 (2008) 004. arXiv:0802.1876, doi:10.1088/1126-6708/2008/05/004.

[61] P. Draggiotis, M. V. Garzelli, C. G. Papadopoulos, R. Pittau, Feynman rules for the rational part of the QCD 1-loop amplitudes, JHEP 04 (2009) 072. arXiv:0903.0356, doi:10.1088/1126-6708/2009/04/072.

[62] M. V. Garzelli, I. Malamos, R. Pittau, Feynman rules for the rational part of the electroweak 1-loop amplitudes, JHEP 01 (2010) 040, [Erratum: JHEP10,097(2010)]. arXiv:0910.3130, doi:10.1007/JHEP10(2010)097,10.1007/JHEP01(2010)040.

[63] H.-S. Shao, Y.-J. Zhang, K.-T. Chao, Feynman rules for the rational part of the Standard Model one-loop amplitudes in the 't Hooft-Veltman γ_5 scheme, JHEP 09 (2011) 048. arXiv:1106.5030, doi:10.1007/JHEP09(2011)048.

[64] R. K. Ellis, W. J. Stirling, B. R. Webber, QCD and collider physics, Camb. Monogr. Part. Phys. Nucl. Phys. Cosmol. 8 (1996) 1–435.

[65] A. Denner, S. Dittmaier, The complex-mass scheme for perturbative calculations with unstable particles, Nucl. Phys. Proc. Suppl. 160 (2006) 22–26. arXiv:hep-ph/0605312, doi:10.1016/j.nuclphysbps.2006.09.025.

[66] A. Kanaki, C. G. Papadopoulos, HELAC-PHEGAS: Automatic computation of helicity amplitudes and cross-sections, [AIP Conf. Proc.583,169(2001)] (2000). arXiv:hep-ph/0012004, doi:10.1063/1.1405294.

[67] F. Maltoni, K. Paul, T. Stelzer, S. Willenbrock, Color flow decomposition of QCD amplitudes, Phys. Rev. D67 (2003) 014026. arXiv:hep-ph/0209271, doi:10.1103/PhysRevD.67.014026.

[68] S. Catani, M. H. Seymour, A general algorithm for calculating jet cross sections in NLO QCD, Nucl. Phys. B485 (1997) 291–419. arXiv:hep-ph/9605323, doi:10.1016/S0550-3213(96)00589-5.

[69] S. Catani, S. Dittmaier, M. H. Seymour, Z. Trocsanyi, The dipole formalism for next-to-leading order QCD calculations with massive partons, Nucl. Phys. B627 (2002) 189–265. arXiv:hep-ph/0201036, doi:10.1016/S0550-3213(02)00098-6.

[70] K. A. Olive, et al., Review of Particle Physics, Chin. Phys. C38 (2014) 090001. doi:10.1088/1674-1137/38/9/090001.

[71] M. Böhm, A. Denner, H. Joos, Gauge theories of the strong and electroweak interaction, Teubner, Stuttgart, Germany, 2001.

[72] E. Accomando, A. Denner, C. Meier, Electroweak corrections to Wγ and Zγ production at the LHC, Eur. Phys. J. C47 (2006) 125–146. arXiv:hep-ph/0509234, doi:10.1140/epjc/s2006-02521-y.