GRIFFIN 1.1: Manual

Lisong Chen[a] and Ayres Freitas[b]

[a]Institute for Theoretical Particle Physics (TTP), Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
[b]Pittsburgh Particle-physics Astro-physics & Cosmology Center (PITT-PACC),, Department of Physics & Astronomy, University of Pittsburgh, Pittsburgh, PA 15260, USA

Program release: GRIFFIN version 1.1 (September xx, 2024).

Language: C++11 (2011 ISO C++), for example `gcc 4.8.1` and beyond.

Download: `http://github.com/lisongc/GRIFFIN/releases/tag/v1.1.0`

Changes with respect to version 1.0

- Implementation of fermion scattering for identical initial- and final-state fermions (*e.g.* Bhabha scattering).
- Implementation of $\mathcal{O}(\alpha_f\alpha_s)$ corrections in off-resonant contributions to fermion scattering, where α_f denotes electroweak (EW) corrections with closed fermion loops.
- Correction of bug in `D0.cc` that affects b-quark initial states.
- More lightweight implementation of dilogarithm, leading to smaller code sizes.
- Adoption of `cmake` build system; possibility to build a GRIFFIN link library; use of `namespace`.

GRIFFIN 1.1 in downward compatible with GRIFFIN 1.0, except for the items noted below with "Version change".

1 Theoretical framework

GRIFFIN 1.1 provides a description of fermion scattering processes $f\bar{f} \to f'\bar{f}'$ with specific attention to a consistent gauge-invariant description of the Z resonance. The matrix element consists of two parts,

$$M_{ij} = \frac{R_{ij}}{s - s_0} + M_{ij}^{\text{offZ}}, \qquad (i,j = \text{V},\text{A},\text{S},\text{P}), \tag{1}$$

where V, A, S, P refer to vector ($\bar{\psi}\gamma^\mu\psi$), axial-vector ($\bar{\psi}\gamma^\mu\gamma^5\psi$), scalar ($\bar{\psi}\psi$) and pseudoscalar ($\bar{\psi}\gamma^5\psi$) currents, respectively, and i (j) is the index for the initial-state $f\bar{f}$ (final-state $f'\bar{f}'$) current. Possible combinations for massless external fermions are $(i,j) = (V,V)$, (V,A), (A,V), (A,A), (S,S), (P,P). S and P only occur for $f = f'$.

The first term in (1) is the first term in a complex-mass about the Z pole,

$$M_{ij}^{\text{exp},s_0} = \frac{R_{ij}}{s - s_0} + S_{ij} + (s - s_0)S'_{ij} + ... \tag{2}$$

where $s_0 \equiv M_\text{Z}^2 - iM_\text{Z}\Gamma_\text{Z}$. The second term, M_{ij}^{offZ}, provides the description away from the Z resonance. It consists itself of two parts,

$$M_{ij}^{\text{offZ}} = M_{ij}^{\text{noexp}} - M_{ij}^{\text{R},M_\text{Z}^2}. \tag{3}$$

Here M_{ij}^{noexp} is the matrix element without any expansion in s and Dyson summation. It has an (unphysical) singularity at $s = M_\text{Z}^2$. To remove this singularity and avoid double counting with M_{ij}^{exp,s_0}, an expanded version of M_{ij}^{noexp} must be subtracted[1]:

$$M_{ij}^{\text{R},M_\text{Z}^2} = \frac{\overline{R}'_{ij}}{(s - M_\text{Z}^2)^2} + \frac{\overline{R}_{ij}}{s - M_\text{Z}^2}. \tag{4}$$

The coefficients in (2), (4) and M_{ij}^{noexp} can be computed including higher-order SM corrections[2], see [1] for details.

Version change: In GRIFFIN 1.0, M_{ij}^{offZ} was defined as the difference $M_{ij} - M_{ij}^{\text{exp},s_0}$, *i.e.* the higher-order terms S and S' were excluded from M_{ij}^{offZ} and added separately. The full matrix element M_{ij} is nevertheless the same in versions 1.0 and 1.1.

The leading coefficient R in (2) can also be reexpressed in terms of the effective weak mixing angle $\sin^2\theta_\text{eff}^f$ and the axial-vector form factor F_A^f:

$$\begin{aligned} R_{ij} = 4I_f^3 I_{f'}^3 \sqrt{F_A^f F_A^{f'}} \Big[&\tilde{Q}_i^f \tilde{Q}_j^{f'}(1 + \delta A) + i(\tilde{Q}_i^f I_{j,f'} + \tilde{Q}_j^{f'} I_{i,f})(1 + \delta B) \\ &- I_{i,f} I_{j,f'}(1 + \delta C) \Big] + \delta D\,, \end{aligned} \tag{5}$$

where

$$\tilde{Q}_V^f = 1 - 4|Q_f| \sin^2\theta_\text{eff}^f, \qquad \tilde{Q}_A^f = 1, \tag{6}$$

$$I_{V,f} = \text{Im}\,\frac{Z_{Vf}}{Z_{Af}}, \qquad I_{A,f} = 0, \tag{7}$$

[1] Alternatively, and more rigorously, $M_{ij}^{\text{R},M_\text{Z}^2}$ can be defined via $M_{ij}^{\text{R},M_\text{Z}^2} = \mathcal{T}_\alpha\left\{\left[\frac{R_{ij}}{s - s_0}\right]_{s_0 = M_\text{Z}^2 - iM_\text{Z}\alpha\Gamma_\text{Z}^{(1)}}\right\}$, where $\mathcal{T}_x$ is a Taylor operator in the variable x, and we have written $\Gamma_\text{Z} \equiv \alpha\Gamma_\text{Z}^{(1)}$ since the leading contribution to Γ_Z in preturbation theory is first order.

[2] In principle, BSM models can also be implemented, but they are not included in GRIFFIN v1.1.

and I_f^3 is the weak isopsin of fermion f, whereas Z_{Vf} and Z_{Af} are the effective vector and axial-vector Zff couplings, respectively. Furthermore, δA, δB, δC, δD denote radiative corrections that first appear at NNLO, see [1] for more details.

The quantities F_A^f and $\sin^2\theta_{\rm eff}^f$ are defined as follows:

$$\sin^2\theta_{\rm eff}^f = \frac{1}{4|Q_f|}\left[1 - \mathrm{Re}\,\frac{Z_{Vf}}{Z_{Af}}\right]_{s=M_{\rm Z}^2}, \tag{8}$$

$$F_A^f = \left[\frac{|Z_{Af}|^2}{1+\mathrm{Re}\,\Sigma'_Z} - \frac{1}{2}M_{\rm Z}\Gamma_{\rm Z}|a^{\rm Z}_{f(0)}|^2\,\mathrm{Im}\,\Sigma''_Z\right]_{s=M_{\rm Z}^2} + \mathcal{O}(\alpha^3), \tag{9}$$

One can also define a vector form factor F_V^f, which however is not independent of F_A^f and $\sin^2\theta_{\rm eff}^f$:

$$F_V^f = \left[\frac{|Z_{Vf}|^2}{1+\mathrm{Re}\,\Sigma'_Z} - \frac{1}{2}M_{\rm Z}\Gamma_{\rm Z}|v^{\rm Z}_{f(0)}|^2\,\mathrm{Im}\,\Sigma''_Z\right]_{s=M_{\rm Z}^2} + \mathcal{O}(\alpha^3) \tag{10}$$

$$= F_A^f\left[(1-4|Q_f|\sin^2\theta_{\rm eff}^f)^2 + I_{V,f}^2\right]. \tag{11}$$

2 The structure of the C++ implementation

The theory framework is implemented within a structure of classes in C++. In v1.1, the SM predictions for EWPOs, the muon decay process, and polarized matrix elements near the Z-peak up to full NNLO and partial NNNLO have been implemented. Yet in principle the GRIFFIN framework can also accommodate predictions for alternative observables, BSM models up to arbitrary higher orders. This section will document the structure of the classes by introducing each class and where to find and modify them in the code. The library has two base classes defined in accordance with input and output. Table 1 shows the type of quantities that fall into the realm of either base class (assuming the SM for inputs).

`class inval`		`class psobs`	
input parameters (in the SM)		output observables	
Boson masses and widths	$M_{\rm W,Z,H}$ $\Gamma_{\rm W,Z}$	pesudo-observables defined at Z-peak	$F_{V,A}$, $\sin^2\theta^f_{eff}$ $\Gamma_{{\rm Z}\to f\bar{f}}$, Δr, etc.
Fermion masses	$m^{\rm OS}_{\rm e,\mu,\tau}$ $m^{\overline{\rm MS}}_{\rm d,u,s,c}(M_{\rm Z})$ $m^{\rm OS}_{\rm t}$	amplitude coefficients under pole scheme	R , S , and S'
Couplings	$\alpha(0)$ $\Delta\alpha \equiv 1-\alpha(0)/\alpha(M_{\rm Z}^2)$ $\alpha_s^{\overline{\rm MS}}(M_{\rm Z}^2)$, G_μ	(polarized) matrix element near or away Z-peak	M_{ij}

Table 1: The categories of two base classes in GRIFFIN.

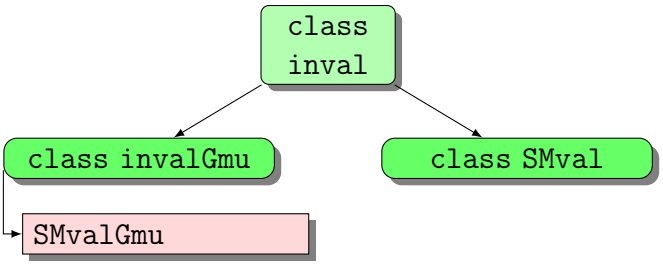

Figure 1: The hierarchy of classes for GRIFFIN's input. The base class `inval` is an abstract class that users need to define input parameters for a certain model (such as the SM or beyond) in its offspring. In GRIFFIN 1.1, we have only implemented the SM for the Z resonance (and muon-decay). Hence the derived classes define different EW input parameter schemes.

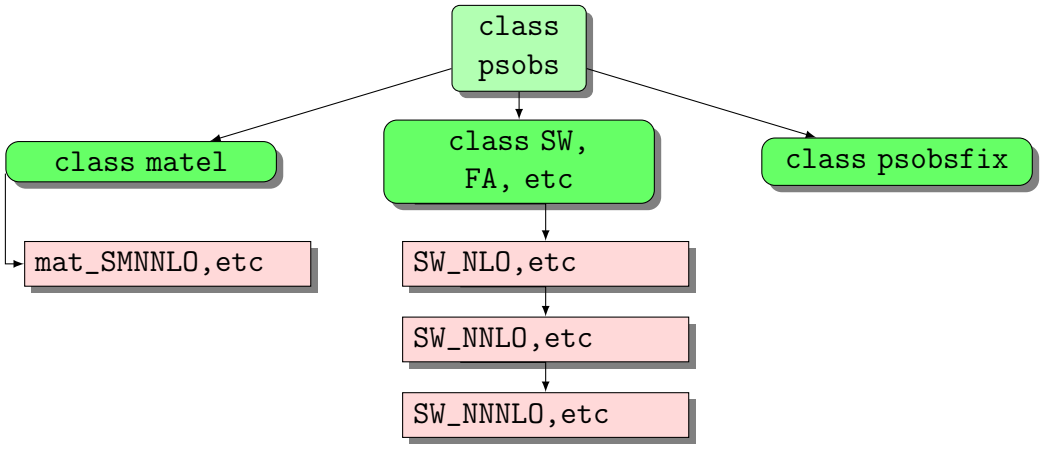

Figure 2: The hierarchy of classes for GRIFFIN's output. The base class `psobs` is an abstract class where several virtual member functions are defined. The user can in principle define new derived classes based off their own purpose. In GRIFFIN 1.1, three types of derived classes, and their sub-derived classes of higher order are defined accordingly.

2.1 Class `inval`

- **Location**: `classes.h`
- **Member Functions**: `set`, `get`, `compute`(virtual)

The base class `inval` serves as the basis for input parameters classes. The functionality of this class is not limited to any particular model. However, derived classes of `inval` can be defined to compute (model-specific) relations between parameters (see method `compute` below). `inval` has three constructors:

- `inval`(*size*): initialize the object with a data array for *size* number of input parameters.

Index	Symbol	Parameter	Index	Symbol	Parameter
0	`MW (MWc*)`	$M_{\rm W}$	12	`al`	$\alpha(0)$
1	`MZ (MZc*)`	$M_{\rm Z}$	13	`als`	$\alpha_{\rm s}^{\overline{\rm MS}}(M_{\rm Z})$
2	`MH`	$M_{\rm H}$	14	`Delal`	$\Delta\alpha \equiv 1-\alpha(0)/\alpha(M_{\rm Z}^2)$
3	`ME`	$m_{\rm e}^{\rm OS}$	15	`Delalhad`	$\Delta\alpha_{\rm had}$
4	`MM`	$m_{\mu}^{\rm OS}$	16	`Gmu`	G_{μ} (Fermi constant)
5	`ML`	$m_{\tau}^{\rm OS}$	17	`GamW (GWc*)`	$\Gamma_{\rm W}$
6	`MD`	$m_{\rm d}^{\overline{\rm MS}}(M_{\rm Z})$	18	`GamZ (GZc*)`	$\Gamma_{\rm Z}$
7	`MS`	$m_{\rm s}^{\overline{\rm MS}}(M_{\rm Z})$	20	`MW*`	$M_{\rm W}^{exp}$
8	`MB`	$m_{\rm b}^{\overline{\rm MS}}(M_{\rm Z})$	21	`MZ*`	$M_{\rm Z}^{exp}$
9	`MU`	$m_{\rm u}^{\overline{\rm MS}}(M_{\rm Z})$	22	`GamW*`	$\Gamma_{\rm W}^{\rm exp}$
10	`MC`	$m_{\rm c}^{\overline{\rm MS}}(M_{\rm Z})$	23	`GamZ*`	$\Gamma_{\rm Z}^{\rm exp}$
11	`MT`	$m_{\rm t}^{\rm OS}$			

Table 2: SM input parameters in GRIFFIN 1.1**. Here OS and $\overline{\rm MS}$ refer to the on-shell and $\overline{\rm MS}$ scheme, respectively. The asterik (*) indicates symbols defined in the classes `SMval` and `SMvalGmu`.**

- `inval()`: initialize the object with a data array of length given by the parameter `SIZE1` in `classes.h` (default 100).
- `inval(`*copyfrom*`)`: initialize the object by cloning another `inval` object *copyfrom*.

In addition, there are three important member functions passed down to derived classes:

- `set(`*idx*`,`*val*`)`: assign numerical value *val* to the input parameter specified by index *idx*. In GRIFFIN 1.1, the indices listed in Tab. 2 are reserved for SM parameters. Indices beyond 24 can, in principle, be used for BSM paremeters. An index number larger than the size of `inval`'s data array (see description of constructors above) produces an error message; so it is important to initialize any object with a sufficiently large size parameter.
- `get(`*idx*`)`: to retrieve the value that has been assigned to the index *idx*.
- `compute()`: compute parameter relations in a given model. The function is invoked automatically by `set` whenever an imput parameter is modified. In the base class `inval`, `compute` does not do anything. However, classes derived from `inval` can overload `compute` to, for example, compute the W-boson mass from $\{\alpha(0), M_{\rm Z}$ and $G_{\mu}\}$ (see `invalGmu` below) or to compute masses and widths in complex-pole mass scheme from user-provided masses and widths in the running-width scheme (see `SMval` below). Due to its polymorphic nature, it is thus defined as a virtual member function in `inval`.

2.2 Derived class `SMval`

- **Inherit from**: `inval`
- **Location**: `SMval.h`
- **Member Functions**: `compute`(virtual)

This is a derived class of `inval` providing the SM input parameters scheme $\{\alpha(0)\,, M_{\rm W}\,, M_{\rm Z}\}$ in the complex pole mass scheme, where the gauge boson masses and widths are determined from the masses and widths in the running-width scheme[3] according to

$$\overline{M} = M^{\rm exp}/\sqrt{1+(\Gamma^{\rm exp}/M^{\rm exp})^2}\,, \overline{\Gamma} = \Gamma^{\rm exp}/\sqrt{1+(\Gamma^{\rm exp}/M^{\rm exp})^2}. \tag{12}$$

To declare the input in the main function, one can write

Example 2.2.1 Setting input values, with conversion of the gauge-boson masses and widths from PDG value to complex-pole masses and widths

```
#include "SMval.h"
using namespace griffin;
int main()
{
  SMval myinput; //defining the input set as an object of class SMval
  myinput.set(MZ, 91.1876);
  myinput.set(GamZ, 2.4966);

  cout << myinput.get(MZc) << endl; //output the Z-boson mass in complex-
   pole mass scheme
}
```

where `MZc` is the Z-boson mass under complex-pole mass scheme.

2.3 Derived class `invalGmu`

- **Inherit from**: `inval`
- **Location**: `deltar.h, deltar.cc`
- **Dependencies**: `B0.cc, delrho.cc, li.cc, linex.cc`
- **Member Functions**: `compute`(virtual)

This is another input class where the W-boson mass is computed from EW input scheme $\{\alpha(0), M_{\rm Z}, G_\mu\}$. The polymorphic function `compute()` defined in the scope of this class can be found in `deltar.cc`, where the pesudo-observable Δr from muon-decay is computed, using the class `dr_SMNNLO` (see below).

[3]The running-width scheme is used by the PDG and most experimental collaborations.

2.4 Derived class `SMvalGmu`

- **Inherit from**: `invalGmu`
- **Location**: `SMvalG.h`
- **Dependencies**: `B0.cc, delrho.cc, linex.cc`
- **Member Functions**: `compute`(virtual).

In this input class, not only does it use the $\{\alpha(0), M_{\rm Z}, G_\mu\}$ scheme, but also convert masses/widths in the running-width scheme to complex-pole masses/widths. To use the input set from this class, one simply changes `SMval.h` and `SMval` into `SMvalG.h` and `SMvalG` in Example 2.2.1.

2.5 Class `psobs`

- **Location**: `classes.h`
- **Member Functions**: `setinput`(virtual), `result`(virtual), `errest`(virtual).

The abstract base class `psobs` serves as the output class that has various types of virtual member functions which will be defined differently in its offspring classes. Its constructor is

- `psobs(`*input*`)`, where *input* is an `inval` object that specifies the input parameters to be used for any output produced by `psobs` and its descendents.

The three virtual member functions are

- `result()`: the virtual member function that directly computes different pseudo-observables and form factors.
- `errest()`: the virtual member function that provides the theoretical uncertainty from missing higher orders for different pseudo-observables. It returns zero by default, which indicates that no error estimate is available. In derived classes it can be overloaded by a version that delivers an actual error estimate.
- `setinput(`*input*`)`: specify a new input parameter set given by *input*, which is an object of type `inval`. This will override the input parameter set passed through the constructor.

2.6 Derived classes `FA_SMLO, SW_SMLO, FV_SMLO`

- **Inherit from**: `psobs`
- **Location**: `classes.h, classes.cc`
- **Member Functions**: `result, setftype`

(a)

Index	Symbol	Particle
11	ELE	e
12	NUE	ν_e
13	MUO	μ
14	NUM	ν_μ
15	TAU	τ
16	NUT	ν_τ

Index	Symbol	Particle
1	DQU	d
2	UQU	u
3	SQU	s
4	CQU	c
5	BQU	b

(b)

Index	Symbol	Coupling
0	VEC	vector
1	AXV	axial-vector
2	SCA	scalar*
3	PSC	pseudoscalar*

*only relevant for $f{=}f'$

Table 3: Fermion types and vertex types defined in GRIFFIN 1.1**. The fermion indices follow the PDG particle numbering scheme (see section 45 of Ref. [2],** `https://pdg.lbl.gov/2022/reviews/rpp2022-rev-monte-carlo-numbering.pdf`**).**

These three classes are the base classes for the form factors $F^f_{V,A}$ and $\sin^2\theta^f_{\text{eff}}$, which tree-level results within the SM. Derived classes can be defined to include higher-order corrections (see below) or to define results in a BSM model. `FV_SMLO` internally uses `FA_SMLO` and `SW_SMLO` to compute its result according to eq. (11).

The classes `FA_SMLO, SW_SMLO, FV_SMLO` overload the `result()` function to implement the appropriate SM expressions. In addition to what is inherited from `psobs`, they have the following public functions:

- `FA_SMLO(`*type, input*`)`, `SW_SMLO(`*type, input*`)`, `FV_SMLO(`*type, input*`)`: the constructors require the following arguments:

  input: the usual `inval` object that specifies the model parameters;

  type: the fermion type f, see Tab. 3 (a).

- `setftype(`*type*`)`: changes the fermion type f to *type*.

2.7 Derived classes `FA_SMNLO, SW_SMNLO, FV_SMNLO`

- **Inherit from**: `FA_SMLO, SW_SMLO, FV_SMLO`, respectively.
- **Location**: `EWPOZ.h, EWPOZ.cc`.
- **Dependencies**: `classes.cc, B0.cc, C0.cc, D0.cc, ff.cc, li.cc`

- **Member Functions**: `result, res1f, res1b, errest`

The new or overloaded elements are:

- `res1f()`: NLO corrections with closed fermion loops.
- `res1b()`: NLO corrections without closed fermion loops.
- `result()`: sum of tree-level plus full NLO corrections.
- `errest()`: a simple estimate of the uncertainty from missing NNLO corrections, using the factors $\frac{g^2 N_f}{4\pi^2}$ and $2\frac{\alpha_s}{\pi}C_F$ for the relative size of one more EW and QCD loop, respectively.

2.8 Derived classes `FA_SMNNLO, SW_SMNNLO, FV_SMNNLO`

- **Inherit from**: `FA_SMNLO, SW_SMNLO, FV_SMNLO`, respectively.
- **Location**: `EWPOZ2.h, EWPOZ2.cc`.
- **Dependencies**: `classes.cc, EWPOZ.cc, B0.cc, C0.cc, D0.cc, ff.cc, li.cc, delrho.cc, linex.cc`.
- **Member Functions**: `result, errest`

  only in `FA_SMNNLO, SW_SMNNLO`:
  `res2ff, res2fb, res2bb, res2aas, res2aasnf, drho2a2, drho2aas, drho3a2as, drho3a3, drho3aas2, drho4aas3, res3fff, res3ffa2as`

These three classes give the SM predictions for $F^f_{V,A}$ and $\sin^2\theta^f_{\rm eff}$ at complete NNLO ($\mathcal{O}(\alpha^2)+\mathcal{O}(\alpha\alpha_s)$) plus some partial corrections beyond 2-loop order. For instance, these include correction to the EW ρ parameter defined as the ratio bewteen neutral current and charged current at zero momentum transfer [3]

$$\rho = \frac{J_{NC}(0)}{J_{CC}(0)}. \tag{13}$$

A correction to ρ will shift $\sin^2\theta^f_{\rm eff}$ and F^f_A by

$$\delta\sin^2\theta^f_{\rm eff} = \frac{M_{\rm W}^2}{M_{\rm Z}^2}\delta\rho\,, \qquad \delta F^f_A = \frac{\alpha\pi(1-2c_{\rm W}^2)}{4c_{\rm W}^2 s_{\rm W}^4}\delta\rho, \tag{14}$$

In practice, the ρ parameter is useful for capturing leading corrections proportional to some power of $\alpha_{\rm t} \equiv \frac{y_{\rm t}^2}{4\pi}$, where $y_{\rm t}$ is the top Yukawa coupling. Individual member functions have been defined in `FA_SMNNLO` and `SW_SMNNLO` for the following order-by-order contributions:

Corrections entering through $\delta\rho$:			
	`drho2aas`	$\mathcal{O}(\alpha_t\alpha_s)$	[4, 5]
	`drho2a2`	$\mathcal{O}(\alpha_t^2)$	[6–10]
*	`drho3aas2`	$\mathcal{O}(\alpha_t\alpha_s^2)$	[11, 12]
*	`drho3a2as`	$\mathcal{O}(\alpha_t^2\alpha_s)$	[13, 14]
*	`drho3a3`	$\mathcal{O}(\alpha_t^3)$	[13, 14]
*	`drho4aas3`	$\mathcal{O}(\alpha_t\alpha_s^3)$	[15–17]
Full corrections to F_A^f, $\sin^2\theta_{\rm eff}^f$:			
*	`res2ff`	$\mathcal{O}(\alpha_f^2)$	[18–20]
*	`res2fb`	$\mathcal{O}(\alpha_f\alpha_b)$	[18–21]
*	`res2bb`	$\mathcal{O}(\alpha_b^2)$	[22–26]
*	`res2aas`	$\mathcal{O}(\alpha\alpha_s)$	[27–29] (correction to internal gauge-boson self-energies)
*	`res2aasnf`	$\mathcal{O}(\alpha\alpha_s)$	[30–35] (non-factorizable final-state corrections for $f = q$)
*	`res3fff`	$\mathcal{O}(\alpha_f^3)$	[36]
*	`res3ffa2as`	$\mathcal{O}(\alpha_f^2\alpha_s)$	[37]

No resummation of $\delta\rho$ has been implemented. α_f (α_b) denote EW corrections with (without) closed fermion loops. Note that `res2aas` includes the contribution from `drho2aas`, *etc.*

`result()` returns the sum of all contributions indicated by an asterik (*) in the table above, plus tree-level and one-loop corrections. `errest()` returns the theory error estimates from Ref. [26]. A simple example for the use of these classes reads as follows:

Example 2.8.1 Generating numerical results of $F_{V,A}^f$ and $\sin^2\theta_{\rm eff}^f$ at NNLO in the SM:

```
#include "EWOPZ2.h"
#include "SMval.h"
using namespace griffin;
int main()
{
  SMval myinput; //defining the input set as an object of class SMval
  myinput.set(MZ, 91.1876);
  myinput.set(GamZ, 2.4966);
  ...           // more input parameters to be set up

  //defining objects from classes FA_SMNNLO and SW_SMNNLO:
  FA_SMNNLO FA2l(LEP, myinput);
  SW_SMNNLO SW2l(LEP, myinput);

  // rad. correction for FA and SW due to delta_rho at O(alpha_t*alpha_s^2):
  cout << FA2l.drho3aas2() << endl;
  cout << SW2l.drho3aas2() << endl;
```

```
  // output F_A^l and SW^l at NNLO + leading NNNLO:
  cout << "FA^lep" << FAl.result() << endl;
  cout << "SW^lep" << SWl.result() << endl;
}
```

If the user wished to obtain strictly NNLO results (without partial higher orders), this can be achieved by modifying `result()` according to

Example 2.8.2 Modification of `result()` in `FA_SMNNLO` and `SW_SMNNLO` so that it delivers corrections up to NNLO:

```
Cplx result(void)
{
  return(FA_SMNLO::result()+res2ff()+res2fb()
  +res2bb()+ res2aas()+res2aasnf()); // only alpha^2 and alpha*alpha_s
   contributions are included
}
```

2.9 Derived class `dr_SMNLO`

- **Inherit from**: `psobs`
- **Location**: `deltar.h, deltar.cc`.
- **Dependencies**: `B0.cc, delrho.cc, li.cc, linex.cc`
- **Member Functions**: `result, res1f, res1b`

This class computes the NLO corrections to Δr in

$$G_\mu = \frac{\pi\alpha}{\sqrt{2}M_{\rm W}^2(1-M_{\rm W}^2/M_{\rm Z}^2)}(1+\Delta r). \tag{15}$$

`result()` returns the full NLO corrections, while `res1f()` and `res1b()` provide the NLO with and without closed fermion loop, respectively.

2.10 Derived class `dr_SMNNLO`

- **Inherit from**: `dr_SMNLO`
- **Location**: `deltar.h, deltar.cc`.
- **Dependencies**: `B0.cc, delrho.cc, li.cc, linex.cc`

- **Member Functions**: same as class `SW_SMNNLO` (see section 2.8), plus `res3aas2`

The functions for order-by-order corrections to Δr are equivalent to those in classes `SW_SMNNLO` or `FA_SMNNLO`, but in addition the function `res3aas2()` provides the $\mathcal{O}(\alpha\alpha_s^2)$ corrections [38] beyond the leading $\delta\rho$ term.

Classes for computing Δr are useful for translating EW input schemes with G_μ (see sections 2.3, 2.4). To see how the M_W transfers between two input schemes, one can do the following in the `main()`

Example 2.10.1 Example showing the parametric shift of M_W:

```
#include "SMvalG.h"
using namespace griffin;
int main()
{
  SMvalGmu myinput;
  myinput.set(MZ, 91.1876);
  myinput.set(GamZ, 2.4966);
  ...           // more input parameters
  cout << myinput.get(MWc) << endl; // MW in complex-pole mass scheme + G_mu
    input scheme
  cout << myinput.get(MW) << endl;  // MW in running-width scheme + G_mu
   input scheme
}
```

2.11 Derived classes `matel`

- **Inherit from**: `psobs`
- **Location**: `classes.h`, `classes.cc`.
- **Member Functions**: `setftype`, `setform`, `setkinvar`, `result`, `coeffR`, `coeffS`, `coeffSp`, `resoffZ`

This class serves as the base class for the $f\bar{f} \to f'\bar{f}'$ matrix elements. Class `matel` delivers tree-level SM results, whereas derived classes include higher-order corrections. It also provides results for the coefficients of the complex-pole expansion, R_{ij}, S_{ij}, S'_{ij} in eq. (2). The SM prediction for R is not computed directly, but via the form factors $\sin^2\theta^f_{\rm eff}$ and F^f_A according to

$$R^{(0)}_{ij} = 4I^3_f I^3_{f'} \sqrt{F^f_A F^{f'}_A}\, \tilde{Q}^f_i \tilde{Q}^{f'}_j, \qquad \tilde{Q}^f_V = 1 - 4|Q_f| \sin^2\theta^f_{\rm eff}, \qquad \tilde{Q}^f_A = 1 \tag{16}$$

(see also eq. (5)).

The constructors for `matel` have the following form:

- `matel(`*intype, outtype, inform, outform, FAin, FAout, SWin, SWout, sval, costheta, input*`)`:
  Here *intype* and *outtype* correspond to the fermion flavors f and f', respectively, in the process $f\bar{f} \to f'\bar{f}'$ (see Tab. 3 (a)). Furthermore, *inform* and *outform* denote the initial-state and final-state coupling type, respectively, *i.e.* the indices i, j in eq. (2). They can take the values listed in Tab. 3 (b).

  The next four arguments refer to the form factors: $FAin = F_A^f$, $FAout = F_A^{f'}$, $SWin = \sin^2\theta_{\text{eff}}^f$, $SWout = \sin^2\theta_{\text{eff}}^{f'}$. They can provided either as objects (using the classes in sections 2.6–2.8 for predictions in the SM, or variants thereof for BSM models) or as floating-point numbers (which is useful for the purpose of performing fits).

  Finally, *sval* and *costheta* are the Mandelstam s variable and the cosine of the scattering angle, $\cos\theta$, respectively.

In addition, `matel` has the following public member functions:

- `setftype(`*intype, outtype*`)`: changes the fermion type f to *intype* and f' to *outtype*.
- `setform(`*inform, outform*`)`: changes the initial-state and final-state vertex type to *intype* and *outtype*, respectively.
- `setkinvar(`*sval, costheta*`)`: changes the values for s and $\cos\theta$.
- `coeffR(), coeffS(), coeffSp()`: return numerical results for the coefficients R, S, S' of the the complex-pole expansion, see eq. (2).
- `resoffZ()`: return numerical result for the off-resonance contribution M_{ij}^{offZ}, see eq. (3).

See section 3.2 for an example for how to generate numerical results for the matrix element.

Version change: In GRIFFIN 1.0, `resoffZ()` was defined as the remainder beyond the R, S, S' terms in the complex-pole expansion, whereas in the current version 1.1, it is the remainder beyond the leading pole term R only, see eq. (3). The combined result from `result()` is the same in versions 1.0 and 1.1.

2.12 Derived class `mat_SMNNLO`

- **Inherit from**: `matel`
- **Location**: `xscnnlo.h, xscnnlo.cc`.
- **Dependencies**: `classes.cc, B0.cc, C0.cc, D0.cc, ff.cc, li.cc`
- **Member Functions**: `result, coeffR, coeffS, coeffSp, resoffZ`

This class provides a description of the matrix element M_{ij} that is NNLO on the Z peak (within the complex-pole expansion) and NLO off-peak.

Note that at one order less, *i.e.* NLO in the Z peak and LO off-peak, the expression for the leading pole coefficient R_{ij} is identical to the LO expression in eq. (16). Of course, F_A^f and $\sin^2\theta_{\rm eff}^f$ change when going from LO and NLO, but since they are provided as inputs to `matel`, no separate class for M_{ij} at this order is needed. For completeness, `xscnnlo.h` provides an alias `mat_SMNLO` that are resolved simply as `matel`.

2.13 Derived class `mat_SMaas`

- **Inherit from**: `matel_SMNNLO`
- **Location**: `xscaas.h, xscaas.cc`.
- **Dependencies**: `classes.cc, xscnnlo.cc, linex.cc`
- **Member Functions**: `result, coeffR, resoffZ, resoffZaas`

This class provides a description of the matrix element M_{ij} that is the same accuracy on the Z peak as `mat_SMNNLO`, whereas it is NLO plus $\mathcal{O}(\alpha_f\alpha_{\rm s})$ off-peak. For this purpose, $\mathcal{O}(\alpha_f\alpha_{\rm s})$ corrections have been incorporated into the *imaginary part* of the leading pole coefficient R_{ij} [in `coeffR()`] and into the remainder $M_{ij}^{\rm offZ}$ [in `resoffZ()`]. `resoffZaas()` returns only the $\mathcal{O}(\alpha_f\alpha_{\rm s})$ contribution of the latter. The real part of R_{ij} receives $\mathcal{O}(\alpha_f\alpha_{\rm s})$ corrections via the form factors $\sin^2\theta_{\rm eff}^f$ and F_A^f.

2.14 Additional tools

The files `tools.h, tools.cc` provide functions to compute the partial and total width of the Z boson, including final-state QED/QCD corrections from Refs. [39–42].

- `partzwidth`(*fa, fv, type, input, scheme*): Compute partial width for $Z \to ff$ using the form factor objects provided, *fa* $= F_A^f$, *fv* $= F_V^f$. Here *type* denotes the fermion flavor f, *input* is an input parameter object of type `inval`, and *scheme* can take the following two values:

  scheme = `COMPPOLESCHEME` : the returned partial width is in the complex pole scheme, *i.e.* $\overline{\Gamma}$ in (12);

  scheme = `RUNWIDTHSCHEME` : the returned partial width is in the running-width scheme, *i.e.* $\Gamma^{\rm exp}$ in (12).

  Note that `partzwidth` will modify the *fa* and *fv* objects passed to it, by setting the fermion type and input.

- `zwidth`(*fa, fv, input, scheme*): Compute the total Z width, by summing `partzwidth` over all SM fermion types.

2.15 Implementation notes

The expressions for various higher-order contributions have been automatically generated from computer algebra tools in `Mathematica`, and they are provided in include files with file extension `.in`.

The results for $\alpha\alpha_s$, $\alpha_f\alpha_b$, α_b^2 and $\alpha_f^2\alpha_s$ corrections have been incorporated in the form of parameter grids, since the direct numerical evaluation of these contributions would be relatively slow. Interpolations of these grids (in files with extension `.grid`) are computed using the functions in `linex.cc`. The grids are valid within the following parameter ranges[4]:

$$M_H/M_Z = 0.274...2.47, \quad M_W/M_Z = 0.8795...0.8840, \quad m_t/M_Z = 1.70...2.14, \tag{17}$$

$$\sqrt{s}/m_t = 0.1...6.48. \tag{18}$$

With $M_Z = 91.1535$ GeV they correspond to the mass ranges

$$M_H = 25...225 \text{ GeV}, \quad M_W = 80.17...80.60 \text{ GeV}, \quad m_t = 155...195 \text{ GeV}, \tag{19}$$

$$\sqrt{s} = 20...1000 \text{ GeV}. \tag{20}$$

The ranges (18) and (20) are only relevant for the `mat_SMaas` class.

3 Installation and examples

3.1 Installation and usage

GRIFFIN is a collection of modules that can used by either directly including the relevant source files in a user project, or by first compiling GRIFFIN into a library. To build the GRIFFIN library, a C++11 compatible compiler and the `cmake` build system should be available. Then execute the following commands in the main GRIFFIN directory:

```
cmake .
cmake --build .
```

The resulting library is named `libgriffin.a` on Linux systems.

The download package includes a few example programs in the subdirectory `examples` that illustrate the usage of GRIFFIN v1.1:

`testdeltar.cc`	Illustration the computation of Δr (see sections 2.9 and 2.10) and the usage of the G_μ input schemes (see sect. 2.3 and 2.4).
`testtools.cc`	Evaluation of the Z (partial) width with final-state QED/QCD corrections (see section 2.14).
`testmatel.cc`	Demonstration of how to compute matrix elements and cross-section results for the process $e^+e^- \to f\bar{f}$ (also see next section).
`testbhabha.cc`	Demonstration of how to compute matrix elements and cross-section results for the process $e^+e^- \to e^+e^-$ (new in version 1.1).
`testaas.cc`	Demonstration of how to compute matrix elements and cross-sections for $e^+e^- \to f\bar{f}$ including $\mathcal{O}(\alpha_f\alpha_s)$ corrections (new in version 1.1).

[4]The contributions with fermion loops cover a larger Higgs mass range $M_H = 10...1000$ GeV.

The sample programs can be compiled with the provided `cmake` input files. To compile all examples, first compile the GRIFFIN library as explained above, and then execute the following commands:

```
cd examples
cmake .
cmake --build .
```

To compile just one example program, *e.g.* `testmatel`, replace the last line with

```
cmake --build . --target testmatel
```

Version change: In GRIFFIN 1.1, all classes and functions are encoded in the namespace `griffin`. To access any GRIFFIN elements in a user project one thus has to either add the prefix "`griffin::`" in front of a given function of class name or use the directive "`using namespace griffin`".

3.2 Sample test program and benchmark results

The following example computes SM predictions for the matrix element M_{VV} for $e^+e^- \to d\bar{d}$ to electroweak NNLO and leading higher-order precision on the Z peak and NLO precision off-peak:

```
#include <iostream>
using namespace std;

#include "EWPOZ2.h"
#include "xscnnlo.h"
#include "SMval.h"
using namespace griffin;

int main()
{
  SMval myinput;  // convert masses from PDG values to complex pole scheme
  myinput.set(al, 1/137.03599976);
  myinput.set(MZ, 91.1876);
  myinput.set(MW, 80.377);
  myinput.set(GamZ, 2.4952);
  myinput.set(GamW, 2.085);
  myinput.set(MH, 125.1);
  myinput.set(MT, 172.5);
  myinput.set(MB, 2.87);
  myinput.set(Delal, 0.059);
  myinput.set(als, 0.1179);
```

```
  cout << endl << "Complex-pole masses: MW=" << myinput.get(MWc) << ", MZ="
    << myinput.get(MZc) << endl << endl;

  // compute matrix element for ee->dd with vector coupling in initial
  // state and vector coupling in final state
  int ini = ELE, fin = DQU, iff = VEC, off = VEC;

  cout << "=== Matrix element for ee->dd (i=e, f=d) ===" << endl << endl;

  // compute vertex form factors:
  FA_SMNNLO FAi(ini, myinput), FAf(fin, myinput);
  SW_SMNNLO SWi(ini, myinput), SWf(fin, myinput);
  cout << "F_A^i (NNLO+) = " << FAi.result() << endl;
  cout << "F_A^f (NNLO+) = " << FAf.result() << endl;
  cout << "sineff^i (NNLO+) = " << SWi.result() << endl;
  cout << "sineff^f (NNLO+) = " << SWf.result() << endl;
  cout << endl;

  double cme,        // center-of-mass energy
         cost = 0.5; // scattering angle
  Cplx res1, res2;

  cout << "SM matrix element M_VV for cos(theta)=" << cost << ": " << endl;
  // compute matrix element for ee->dd using SM form factors:
  mat_SMNNLO M(ini, fin, iff, off, FAi, FAf, SWi, SWf, cme*cme, cost,
   myinput);
  cout << "sqrt(s)\t\ttot. result\t\toff-resonance contrib." << endl;
  for(cme = 10.; cme <= 190.; cme += 20.)
  {
    M.setkinvar(cme*cme, cost);
    res1 = M.result();
    res2 = M.resoffZ();
    cout << cme << " \t" << res1 << " \t" << res2 << endl;
  }
  cout << endl;

  return 0;
}
```

This code produces the following output, including a table of matrix element results for different center-of-mass energies. For illustration, the off-resonance contribution is separately

tabulated, and it can be seen to dominate for small center-of-mass energies due to the s-channel photon contribution.

```
Complex-pole masses: MW=80.35, MZ=91.1535

=== Matrix element for ee->dd (i=e, f=d) ===

F_A^i (NNLO+) = (0.034499,0)
F_A^f (NNLO+) = (0.0345443,0)
sineff^i (NNLO+) = (0.231172,0)
sineff^f (NNLO+) = (0.230985,0)

SM matrix element M_VV for cos(theta)=0.5:
sqrt(s)         tot. result             off-resonance contrib.
10      (0.000316739,-5.58082e-06)      (0.000316958,-5.61791e-06)
30      (3.53793e-05,-5.99317e-07)      (3.56216e-05,-6.39686e-07)
50      (1.25851e-05,-1.90789e-07)      (1.28944e-05,-2.39652e-07)
70      (6.07798e-06,-5.97311e-08)      (6.60641e-06,-1.28382e-07)
90      (-7.31188e-07,-3.55673e-06)     (4.0266e-06,-7.48569e-08)
110     (3.14635e-06,-1.62001e-07)      (2.68256e-06,-4.07618e-08)
130     (2.12596e-06,-7.90095e-08)      (1.91932e-06,-3.2315e-08)
150     (1.5668e-06,-5.34561e-08)       (1.44139e-06,-2.64893e-08)
170     (1.20884e-06,-3.97403e-08)      (1.12232e-06,-2.15845e-08)
190     (9.60973e-07,-3.33532e-08)      (8.96823e-07,-2.00842e-08)
```

This example is included in the sample program `testmatel`, which additionally also demonstrates the computation of a differential cross-section.

References

[1] L. Chen and A. Freitas, *GRIFFIN: A C++ library for electroweak radiative corrections in fermion scattering and decay processes*, *SciPost Phys. Codeb.* **2023** (2023) 18, [2211.16272].

[2] Particle Data Group collaboration, R. L. Workman et al., *Review of Particle Physics*, *PTEP* **2022** (2022) 083C01.

[3] M. Veltman, *Limit on Mass Differences in the Weinberg Model*, *Nucl. Phys. B* **123** (1977) 89–99.

[4] A. Djouadi and C. Verzegnassi, *Virtual very heavy top effects in LEP/SLC precision measurements*, *Phys. Lett.* **B195** (1987) 265–271.

[5] A. Djouadi, *$O(\alpha\alpha_s)$ vacuum polarization functions of the standard model gauge bosons*, *Nuovo Cim.* **A100** (1988) 357.

[6] J. J. van der Bij and F. Hoogeveen, *Two Loop Correction to Weak Interaction Parameters Due to a Heavy Fermion Doublet*, *Nucl. Phys. B* **283** (1987) 477–492.

[7] R. Barbieri, M. Beccaria, P. Ciafaloni, G. Curci and A. Vicere, *Radiative correction effects of a very heavy top*, *Phys. Lett. B* **288** (1992) 95–98, [hep-ph/9205238].

[8] R. Barbieri, M. Beccaria, P. Ciafaloni, G. Curci and A. Vicere, *Two loop heavy top effects in the Standard Model*, *Nucl. Phys. B* **409** (1993) 105–127.

[9] J. Fleischer, O. V. Tarasov and F. Jegerlehner, *Two loop heavy top corrections to the rho parameter: A Simple formula valid for arbitrary Higgs mass*, *Phys. Lett. B* **319** (1993) 249–256.

[10] J. Fleischer, O. V. Tarasov and F. Jegerlehner, *Two loop large top mass corrections to electroweak parameters: Analytic results valid for arbitrary Higgs mass*, *Phys. Rev. D* **51** (1995) 3820–3837.

[11] L. Avdeev, J. Fleischer, S. Mikhailov and O. Tarasov, $0(\alpha\alpha_s^2)$ *correction to the electroweak ρ parameter*, *Phys. Lett.* **B336** (1994) 560–566, [hep-ph/9406363].

[12] K. Chetyrkin, J. H. Kühn and M. Steinhauser, *Corrections of order $O(G_F M_t^2 \alpha_s^2)$ to the ρ parameter*, *Phys. Lett.* **B351** (1995) 331–338, doi:10.1016/0370–2693(95)00380–4, [hep-ph/9502291].

[13] J. J. van der Bij, K. G. Chetyrkin, M. Faisst, G. Jikia and T. Seidensticker, *Three loop leading top mass contributions to the ρ parameter*, *Phys. Lett.* **B498** (2001) 156–162, [hep-ph/0011373].

[14] M. Faisst, J. H. Kühn, T. Seidensticker and O. Veretin, *Three loop top quark contributions to the ρ parameter*, *Nucl. Phys.* **B665** (2003) 649–662, [hep-ph/0302275].

[15] Y. Schröder and M. Steinhauser, *Four-loop singlet contribution to the ρ parameter*, *Phys. Lett.* **B622** (2005) 124–130, [hep-ph/0504055].

[16] K. G. Chetyrkin, M. Faisst, J. H. Kühn, P. Maierhöfer and C. Sturm, *Four-loop QCD corrections to the ρ parameter*, *Phys. Rev. Lett.* **97** (2006) 102003, [hep-ph/0605201].

[17] R. Boughezal and M. Czakon, *Single scale tadpoles and $O(G_F m_t^2 \alpha_s^3)$ corrections to the ρ parameter*, *Nucl. Phys.* **B755** (2006) 221–238, [hep-ph/0606232].

[18] M. Awramik, M. Czakon, A. Freitas and G. Weiglein, *Complete two-loop electroweak fermionic corrections to $\sin^2\theta_{\rm eff}^{\rm lept}$ and indirect determination of the Higgs boson mass*, *Phys. Rev. Lett.* **93** (2004) 201805, [hep-ph/0407317].

[19] W. Hollik, U. Meier and S. Uccirati, *The effective electroweak mixing angle* $\sin^2\theta^{\rm eff}$ *with two-loop fermionic contributions*, *Nucl. Phys.* **B731** (2005) 213–224, [hep-ph/0507158].

[20] A. Freitas, *Higher-order electroweak corrections to the partial widths and branching ratios of the Z boson*, *JHEP* **04** (2014) 070, [1401.2447].

[21] M. Awramik, M. Czakon, A. Freitas and B. Kniehl, *Two-loop electroweak fermionic corrections to* $\sin^2\theta^{\rm b\bar{b}}_{\rm eff}$, *Nucl. Phys.* **B813** (2009) 174–187, [0811.1364].

[22] M. Awramik, M. Czakon and A. Freitas, *Bosonic corrections to the effective weak mixing angle at* $O(\alpha^2)$, *Phys. Lett.* **B642** (2006) 563–566, [hep-ph/0605339].

[23] W. Hollik, U. Meier and S. Uccirati, *The effective electroweak mixing angle* $\sin^2\theta^{\rm eff}$ *with two-loop bosonic contributions*, *Nucl. Phys.* **B765** (2007) 154–165, [hep-ph/0610312].

[24] I. Dubovyk, A. Freitas, J. Gluza, T. Riemann and J. Usovitsch, *The two-loop electroweak bosonic corrections to* $\sin^2\theta^{\rm b}_{\rm eff}$, *Phys. Lett.* **B762** (2016) 184–189, [1607.08375].

[25] I. Dubovyk, A. Freitas, J. Gluza, T. Riemann and J. Usovitsch, *Complete electroweak two-loop corrections to Z boson production and decay*, *Phys. Lett.* **B783** (2018) 86–94, [1804.10236].

[26] I. Dubovyk, A. Freitas, J. Gluza, T. Riemann and J. Usovitsch, *Electroweak pseudo-observables and Z-boson form factors at two-loop accuracy*, *JHEP* **08** (2019) 113, [1906.08815].

[27] B. A. Kniehl, *Two loop corrections to the vacuum polarizations in perturbative QCD*, *Nucl. Phys.* **B347** (1990) 86–104.

[28] B. A. Kniehl and A. Sirlin, *Dispersion relations for vacuum polarization functions in electroweak physics*, *Nucl. Phys.* **B371** (1992) 141–148.

[29] A. Djouadi and P. Gambino, *Electroweak gauge boson selfenergies: Complete QCD corrections*, *Phys. Rev.* **D49** (1994) 3499–3511, [hep-ph/9309298].

[30] A. Czarnecki and J. H. Kuhn, *Nonfactorizable QCD and electroweak corrections to the hadronic Z boson decay rate*, *Phys. Rev. Lett.* **77** (1996) 3955–3958, [hep-ph/9608366].

[31] R. Harlander, T. Seidensticker and M. Steinhauser, *Complete corrections of order* $\alpha\alpha_s$ *to the decay of the Z boson into bottom quarks*, *Phys. Lett.* **B426** (1998) 125–132, [hep-ph/9712228].

[32] J. Fleischer, O. Tarasov, F. Jegerlehner and P. Raczka, *Two loop* $O(\alpha_s G_\mu m_t^2)$ *corrections to the partial decay width of the* Z^0 *into* $b\bar{b}$ *final states in the large top mass limit*, *Phys. Lett.* **B293** (1992) 437–444.

[33] G. Buchalla and A. J. Buras, *QCD corrections to the $\bar{s}dZ$ vertex for arbitrary top quark mass*, *Nucl. Phys.* **B398** (1993) 285–300.

[34] G. Degrassi, *Current algebra approach to heavy top effects in $Z \to b + \bar{b}$*, *Nucl. Phys.* **B407** (1993) 271–289, [hep-ph/9302288].

[35] K. Chetyrkin, A. Kwiatkowski and M. Steinhauser, *Leading top mass corrections of order $O(\alpha\alpha_s m_t^2/m_W^2)$ to partial decay rate $\Gamma(Z \to b\bar{b})$*, *Mod. Phys. Lett.* **A8** (1993) 2785–2792.

[36] L. Chen and A. Freitas, *Leading fermionic three-loop corrections to electroweak precision observables*, *JHEP* **07** (2020) 210, [2002.05845].

[37] L. Chen and A. Freitas, *Mixed EW-QCD leading fermionic three-loop corrections at $\mathcal{O}(\alpha_s\alpha^2)$ to electroweak precision observables*, *JHEP* **03** (2021) 215, [2012.08605].

[38] K. G. Chetyrkin, J. H. Kuhn and M. Steinhauser, *QCD corrections from top quark to relations between electroweak parameters to order alpha-s**2*, *Phys. Rev. Lett.* **75** (1995) 3394–3397, [hep-ph/9504413].

[39] K. G. Chetyrkin, J. H. Kuhn and A. Kwiatkowski, *QCD corrections to the e^+e^- cross-section and the Z boson decay rate*, *Phys. Rept.* **277** (1996) 189–281, [hep-ph/9503396].

[40] P. A. Baikov, K. G. Chetyrkin and J. H. Kuhn, *Order alpha**4(s) QCD Corrections to Z and tau Decays*, *Phys. Rev. Lett.* **101** (2008) 012002, [0801.1821].

[41] P. A. Baikov, K. G. Chetyrkin, J. H. Kuhn and J. Rittinger, *Complete $\mathcal{O}(\alpha_s^4)$ QCD Corrections to Hadronic Z-Decays*, *Phys. Rev. Lett.* **108** (2012) 222003, [1201.5804].

[42] A. L. Kataev, *Higher order O (alpha**2) and O(alpha alpha-s) corrections to sigma total (e+ e- —> hadron and Z boson decay rate*, *Phys. Lett. B* **287** (1992) 209–212.