How fast does the `WallGo`?
A package for computing wall velocities in first-order phase transitions

Andreas Ekstedt[a,*], Oliver Gould[b,†], Joonas Hirvonen[b,c,‡], Benoit Laurent[d,§],
Lauri Niemi[c,¶], Philipp Schicho[e,‖], Jorinde van de Vis[f,††]

Abstract

`WallGo` is an open-source software designed to compute the bubble wall velocity in first-order cosmological phase transitions. Additionally, it evaluates the energy budget available for generating gravitational waves. The main part of `WallGo`, built in `Python`, determines the wall velocity by solving the scalar-field(s) equation of motion, the Boltzmann equations and energy-momentum conservation for the fluid velocity and temperature. `WallGo` also includes two auxiliary modules: `WallGoMatrix`, which computes matrix elements for out-of-equilibrium particles, and `WallGoCollision`, which performs higher-dimensional integrals for Boltzmann collision terms. Users can implement custom models by defining an effective potential and specifying a list of out-of-equilibrium particles and their interactions.

As the first public software to compute the wall velocity including out-of-equilibrium contributions, `WallGo` improves the precision of the computation compared to common assumptions in earlier computations. It utilises a spectral method for the deviation from equilibrium and collision terms that provides exponential convergence in basis polynomials, and supports multiple out-of-equilibrium particles, allowing for Boltzmann mixing terms. `WallGo` is tailored for non-runaway wall scenarios where leading-order coupling effects dominate friction.

While this work introduces the software and the underlying theory, a more detailed documentation can be found in `https://wallgo.readthedocs.io`.

[a]*Department of Physics and Astronomy, Uppsala University,*
P.O. Box 256, SE-751 05 Uppsala, Sweden

[b]*School of Physics and Astronomy, University of Nottingham,*
Nottingham NG7 2RD, United Kingdom

[c]*Department of Physics and Helsinki Institute of Physics, University of Helsinki, FI-00014, Finland*

[d]*McGill University, Department of Physics, 3600 University St., Montréal, QC H3A2T8 Canada*

[e]*Département de Physique Théorique, Université de Genève,*
24 quai Ernest Ansermet, CH-1211 Genève 4, Switzerland

[f]*Theoretical Physics Department, CERN,*
1 Esplanade des Particules, CH-1211 Geneva 23, Switzerland

[*]andreas.ekstedt@physics.uu.se

[†]oliver.gould@nottingham.ac.uk

[‡]joonas.hirvonen@nottingham.ac.uk

[§]benoit.laurent@mail.mcgill.ca

[¶]lauri.b.niemi@helsinki.fi

[‖]philipp.schicho@unige.ch

[††]jorinde.van.de.vis@cern.ch

Contents

Notation

Indices

    - a, b, c: particle species

    - i, j, k: polynomial basis elements

    - i: background scalar field components

    - I, J, L, M: spinor indices

    - α, β, γ: lattice points

    - μ, ν: Lorentz indices

    - $\lambda_1, \lambda_2, \lambda_3$: helicities

Program summary

License: GNU General Public License 3 (GPLv3)

Operating system: Any operating system; tested macOS 11–15, Ubuntu 18–24 and Windows 10–11.

Program title: `WallGo [v1.0.0]`

Program obtainable from: `https://pypi.org/project/WallGo`

Documentation link: `https://wallgo.readthedocs.io`

Developer's repository link: `https://github.com/Wall-Go/WallGo`

Programming languages: `Python`

Nature of problem: Computing terminal velocities of expanding bubble walls for first-order phase transitions, as coupled sets of scalar field, hydrodynamic and Boltzmann equations.

Solution method: The scalar-field equations are solved as a variational problem with an ansatz, and the Boltzmann equations are solved using a spectral decomposition.

Restrictions: `Python` version 3.10 or above.

Program title: `WallGoCollision [v1.0.0]`

Program obtainable from: `https://pypi.org/project/WallGoCollision`

Documentation link: `https://wallgocollision.readthedocs.io`

Developer's repository link: `https://github.com/Wall-Go/WallGoCollision`

Programming languages: `C++` and `Python`

Nature of problem: Computing collision integrals for Boltzmann equations in a spectral expansion.

Solution method: Adaptive Monte-Carlo integration.

Restrictions: `Python` version 3.10 or above. Source builds require a `C++17` compliant compiler and `CMake` 3.18 or newer.

Program title: `WallGoMatrix [v1.0.0]`

Program obtainable from:
`https://resources.wolframcloud.com/PacletRepository/resources/WallGo/WallGoMatrix`

Developer's repository link: `https://github.com/Wall-Go/WallGoMatrix`

Programming languages: `Mathematica`

External routines/libraries: `DRalgo` [1] and `GroupMath` [2].

Nature of problem: Computing 2-to-2 scattering matrix elements for arbitrary quantum field theories.

Solution method: Matrix elements are constructed by building coupling tensors for a given model, and contracting these with kinematic factors appropriately.

Restrictions: Tested on `Mathematica` versions 12, 13 and 14.

1. Introduction

Cosmological first-order phase transitions (FOPTs) are exciting possible processes in the early universe. They might have played a role in generating the asymmetry between matter and antimatter (e.g. in electroweak baryogenesis [3–7], or alternatives [8–10]), and they could also have sourced gravitational wave (GW) signals that can be observed with the next generation of GW detectors [11–14]. It is well-known that the two candidate processes of the Standard Model (SM) of particle physics, the electroweak phase transition [15–20] and the QCD phase transitions [21, 22], are in fact cross-overs. There are, however, many possibilities for extending the SM to render these phase transitions first order (see e.g. [13] and references therein for a number of examples). Moreover, a significant part of the energy budget of the universe is in dark matter and energy, and phase transitions could have also occurred in dark sectors, with possibly observable GW signatures [23–25].

The expansion velocity of bubbles generated during a FOPT, denoted v_w, is a crucial parameter for predicting GW signals. Variations in v_w can lead to changes of up to two orders of magnitude in predicted GW amplitudes [13, 26, 27], and the spectral shape of the GW spectrum is also highly sensitive to v_w [28–32]. Studies [33, 34] suggest that *if* LISA detects a GW signal from a FOPT, v_w may be the most accurately reconstructible parameter. Accurate predictions of GW spectra are essential to distinguish signals from FOPTs amid astrophysical sources in data from observatories like LISA [35]. Furthermore, the wall velocity can significantly impact predictions for the value of the matter-antimatter asymmetry [36–38], which can determine whether a model can successfully explain the observed asymmetry. In alternative mechanisms for baryogenesis, e.g. [9, 10, 39], and dark matter generation [40, 41] during phase transitions, the baryon and dark matter abundances are also expected to depend on v_w.

The wall velocity is determined as the speed at which the outward pressure, resulting from the pressure difference between the phases, is balanced by the friction and backreaction from the plasma. In practice, finding v_w (and the wall width) requires solving the scalar-field(s) equation of motion, the Boltzmann equations for the particles in the plasma, and the energy-momentum conservation equations for the fluid velocity and temperature profile. Early work on computing the wall velocity employed a so-called ballistic approximation [42, 43], which assumes that the mean free path of the particles is much larger than the wall width. The authors of [44, 45], were the first to self-consistently solve the scalar field and Boltzmann equations, including leading-log collision terms. For additional calculations of v_w using similar methods, see e.g., [46–49].

The approach of [44, 45] describes deviations from equilibrium in the distribution functions via the chemical potential μ, deviation of the temperature δT, and fluid velocity δv. By taking three moments of the Boltzmann equations, a set of equations for $\mu, \delta T$ and δv is obtained. However, as noted in [50, 51], the so-obtained equations contain a singularity around the sound speed. In [51], it was demonstrated that this singularity stems from the linearisation of the background solution. To improve the description of the out-of-equilibrium contribution, the authors introduce a generalised

fluid ansatz, and solve the Boltzmann equations for a larger set of moments. In contrast, [52] adopts an alternative approach using a spectral method to describe deviations from equilibrium, thereby eliminating the need for a linearisation of the background distribution. We adopt this latter approach in our work.

Due to the difficulty of computing the wall velocity, it is often guessed or treated as a free parameter. Estimates can be made by assuming local thermal equilibrium (LTE) [53–55], or by considering a large jump in degrees of freedom [56]. The LTE assumption has been shown to provide a reasonable approximation for the Standard Model coupled to a singlet [52], and should offer an upper bound on v_w. Further improvement in determining v_w requires accounting for out-of-equilibrium effects.

`WallGo` provides an **automated computation of the wall velocity for user-defined models**. The software package consists of a main part and two auxiliary modules:

- `WallGo` finds v_w by solving the equations of motion of the scalar fields, and the Boltzmann equations of the out-of-equilibrium particles.

- `WallGoMatrix` computes the $2 \rightarrow 2$ matrix elements for user-specified particles and interactions.

- `WallGoCollision` computes the collision integrals in the spectral basis.

`WallGo` allows readers to largely reduce the uncertainty related to the wall velocity in predictions of the GW signal and the baryon or dark matter abundance. Additionally, it will help to identify the largest sources of uncertainty in v_w, offering guidance for future studies aimed at further minimizing these uncertainties. A detailed examination of these uncertainty sources will be addressed in future work.

In addition to being the *first* publicly available numerical code for solving the wall velocity with out-of-equilibrium effects for user-defined models, `WallGo` also provides several options that improve the computation beyond commonly used approximations in the literature:

- *Matrix elements and collision terms*: `WallGo` computes matrix elements and collision terms for arbitrary particles and interactions, enabling analysis beyond the typical focus on only the out-of-equilibrium top quark and strong interactions.

- *User-defined potential*: The effective potential is fully customizable, allowing calculations beyond leading order. Higher-order corrections to the potential, shown to significantly affect aspects of phase transitions [57–62], will be explored in future work.

- *Mixing in collision terms and Boltzmann equations*: When multiple particles are out of equilibrium, `WallGo` incorporates mixing in the collision terms and Boltzmann equations (cf. section 3.4). This is in contrast to the conventional moment expansion where interaction rates replace collision terms [45].

- *Spectral method for convergence*: `WallGo` uses the spectral method introduced in [52], which parameterises the deviation from equilibrium on a basis of polynomials. For a sufficiently large basis of polynomials, the solution converges exponentially, which we expect to be a relevant improvement over the common three-moment Boltzmann equation method.

- *Computation of the gravitational wave energy budget*: `WallGo` can directly compute the efficiency factor used in the prediction of the GW power spectrum [13, 27, 32, 63], by solving the hydrodynamics equations with the fully model-dependent equation of state. This removes the need to map onto a simplified equation of state [26, 64, 65].

This version of `WallGo` has been developed to deal with weak to moderately strong phase transitions, where the leading-order pressure from out-of-equilibrium particles and the hydrodynamic backreaction are sufficient to compensate the driving force. For stronger phase transitions, this source of friction might not be sufficient to stop the wall from accelerating. The question whether the wall "runs away" in this scenario, or whether it is slowed down by next-to-leading-order contributions to the friction, has been a matter of active discussion in recent literature [66–72]. We leave the inclusion of these next-to-leading order contributions to the friction for a future version of `WallGo`.

This paper is organised as follows. Section 2 describes how to install `WallGo` and how to run a first simple model. In section 3, we detail the physics underlying the computation of the wall velocity. Section 4 overviews the different parts of the software package and their interactions with each other. In section 5, we describe convergence tests. We demonstrate some examples and compare to the literature in section 6, and conclude in section 7.

2.   Installation and running

The main `WallGo` `Python` package can be installed directly with pip, by running the following listing

L.1
```
pip install WallGo
```

from the command line, and then imported as any other `Python` package. A first example of how to use `WallGo` follows in section 2.1. A collection of other example models is found at `https://wallgo.readthedocs.io`.

The `WallGo` package requires collision integrals as input, loaded from files. Some pre-computed collision files can be found on the repository, in the folder `Models`. Beyond this, installing the `WallGoCollision` package enables the computation of new collision integrals, and is also available using pip,

L.2
```
pip install WallGoCollision
```

Example usage is also found within the collected example models at `https://wallgo.readthedocs.io`.

Finally, the `WallGoCollision` package requires 2-to-2 scattering matrix elements as input. The `WallGoMatrix Mathematica` package allows for computing these for generic models. To install this package, one must also install `GroupMath` [2] on which it depends. This can be done automatically by setting

```
WallGo`WallGoMatrix`$InstallGroupMath=True
```

in `Mathematica` before loading `WallGoMatrix`. The latter can be installed by running the following

```
PacletInstall["WallGo/WallGoMatrix"]
```

after which the package can be loaded with

```
<<WallGo`WallGoMatrix`
```

A number of matrix element calculations for example models can be found in the `examples` folder of the `WallGoMatrix` GitHub repository.

2.1. Defining a simple model

Defining a model in `WallGo` requires a few different ingredients: a scalar potential, a list of the particles in the model together with their properties, and the matrix elements for interactions between these particles. The matrix elements are used to compute the collision integrals in `WallGoCollision`. The collision integrals are then loaded into `WallGo` for the wall velocity computation.

We will now describe a simple example which demonstrates how to compute the wall velocity with `WallGo`. The relevant file can be found in `Models/Yukawa/yukawa.py`. Collision integrals are contained in `Models/Yukawa/CollisionOutput_N11`. These have been obtained by running the file `Models/Yukawa/yukawaWithCollisionGeneration.py`. Generating matrix elements and collisions will be further discussed in section 4.

Concretely, let us consider a simple model of a real scalar field ϕ coupled to a Dirac fermion ψ via a Yukawa coupling. Its interaction Lagrangian is given by

$$\mathscr{L} = -\frac{1}{2}\partial_\mu\phi\partial^\mu\phi - \sigma\phi - \frac{m^2}{2}\phi^2 - \frac{g}{3!}\phi^3 - \frac{\lambda}{4!}\phi^4 - i\bar{\psi}\slashed{\partial}\psi - m_f\bar{\psi}\psi - y\phi\bar{\psi}\psi \,. \tag{2.1}$$

In this case, the scalar field may undergo a phase transition, with the fermion field contributing to the friction for the bubble wall growth.

The first step to implement this model into `WallGo` is to define a specific model class, inheriting from the base class `WallGo.GenericModel`.

```python
import pathlib
import numpy as np
import WallGo
from WallGo import Fields, GenericModel, Particle

class YukawaModel(GenericModel):
    """
    The Yukawa model, inheriting from WallGo.GenericModel.
    """

    def __init__(self) -> None:
        """
        Initialize the Yukawa model.
        """
        self.modelParameters: dict[str, float] = {}

        # Initialize internal effective potential
        self.effectivePotential = EffectivePotentialYukawa(self)

        # Create a list of particles relevant for the Boltzmann equations
        self.defineParticles()

    # ~ GenericModel interface
    @property
    def fieldCount(self) -> int:
        """How many classical background fields"""
        return 1

    def getEffectivePotential(self) -> "EffectivePotentialYukawa":
        return self.effectivePotential

    # ~

    def defineParticles(self) -> None:
        """
        Define the out-of-equilibrium particles for the model.
        """
        self.clearParticles()

        # === left fermion ===
        # Vacuum mass squared
        def psiMsqVacuum(fields: Fields) -> Fields:
            return (
```

```
            self.modelParameters["mf"]
            + self.modelParameters["y"] * fields.getField(0)
        ) ** 2

    # Field-derivative of the vacuum mass squared
    def psiMsqDerivative(fields: Fields) -> Fields:
        return (
            2
            * self.modelParameters["y"]
            * (
                self.modelParameters["mf"]
                + self.modelParameters["y"] * fields.getField(0)
            )
        )

    psiL = Particle(
        "psiL",
        index=1,
        msqVacuum=psiMsqVacuum,
        msqDerivative=psiMsqDerivative,
        statistics="Fermion",
        totalDOFs=2,
    )
    psiR = Particle(
        "psiR",
        index=2,
        msqVacuum=psiMsqVacuum,
        msqDerivative=psiMsqDerivative,
        statistics="Fermion",
        totalDOFs=2,
    )
    self.addParticle(psiL)
    self.addParticle(psiR)
```

The scalar potential is used both for determining the free energy of homogeneous phases and for the shape and width of the bubble wall. In principle, the potentials determining these two phenomena are different, as the former is coarse grained all the way to infinite length scales, while the latter can only consistently be coarse grained on length scales shorter than the bubble wall width [73]. Nevertheless, at high temperatures and to leading order in powers of the coupling, these two potentials agree.

At high temperatures, the leading-order effective potential of our simple model is

$$V^{\text{eff}}(\phi, T) = -\frac{\pi^2}{20}T^4 + \sigma_{\text{eff}}\phi + \frac{1}{2}m_{\text{eff}}^2\phi^2 + \frac{1}{3!}g\phi^3 + \frac{1}{4!}\lambda\phi^4 \,, \tag{2.2}$$

where we have defined the effective tadpole coefficient and effective mass as

$$\sigma_{\text{eff}} = \sigma + \frac{1}{24}(g + 4y\,m_f)T^2 \,, \qquad\qquad m_{\text{eff}}^2 = m^2 + \frac{1}{24}(\lambda + 4y^2)T^2 \,. \tag{2.3}$$

9

The implementation in `WallGo` is as follows: one defines a class, here `EffectivePotentialYukawa` which inherits from the base class `WallGo.EffectivePotential`. This definition must contain a member function called `evaluate` which evaluates the potential as a function of the scalar fields and temperature.

```python
class EffectivePotentialYukawa(WallGo.EffectivePotential):
    """
    Effective potential for the Yukawa model.
    """

    def __init__(self, owningModel: YukawaModel) -> None:
        """
        Initialize the EffectivePotentialYukawa.
        """

        super().__init__()

        assert owningModel is not None, "Invalid model passed to Veff"

        self.owner = owningModel
        self.modelParameters = self.owner.modelParameters

    # ~ EffectivePotential interface
    fieldCount = 1
    """How many classical background fields"""

    effectivePotentialError = 1e-15
    """
    Relative accuracy at which the potential can be computed. Here the potential is
    polynomial so we can set it to the machine precision.
    """
    # ~

    def evaluate(self, fields: Fields, temperature: float) -> float | np.ndarray:
        """
        Evaluate the effective potential.
        """
        # getting the field from the list of fields (here just of length 1)
        fields = WallGo.Fields(fields)
        phi = fields.getField(0)

        # the constant term
        f0 = -np.pi**2 / 90 * (1 + 4 * 7 / 8) * temperature**4

        # coefficients of the temperature and field dependent terms
```

```python
        y = self.modelParameters["y"]
        mf = self.modelParameters["mf"]
        sigmaEff = (
            self.modelParameters["sigma"]
            + 1 / 24 * (self.modelParameters["gamma"] + 4 * y * mf) * temperature**2
        )
        msqEff = (
            self.modelParameters["msq"]
            + 1 / 24 * (self.modelParameters["lam"] + 4 * y**2) * temperature**2
        )

        potentialTotal = (
            f0
            + sigmaEff * phi
            + 1 / 2 * msqEff * phi**2
            + 1 / 6 * self.modelParameters["gamma"] * phi**3
            + 1 / 24 * self.modelParameters["lam"] * phi**4
        )

        return np.array(potentialTotal)
```

The `EffectivePotential` stores the model parameters for use in evaluating the potential. It is possible to override other member functions when defining `EffectivePotentialYukawa`, such as the initialisation function, or to add additional member functions and variables, though we have not done so in this simple example.

Once these two classes have been defined, we can now set up the `WallGoManager` and run `WallGo` to compute the bubble wall speed. What follows now, would be the content of the `main` function of the model file. First, we perform some initialisations.

L.8

```python
manager = WallGo.WallGoManager()

# Change the amount of grid points in the spatial coordinates
# for faster computations
manager.config.configGrid.spatialGridSize = 20
# Increase the number of iterations in the wall solving to
# ensure convergence
manager.config.configEOM.maxIterations = 25
# Decrease error tolerance for phase tracing to ensure stability
manager.config.configThermodynamics.phaseTracerTol = 1e-8

pathtoCollisions = pathlib.Path(__file__).resolve().parent / pathlib.Path(
    f"CollisionOutput_N11"
)
manager.setPathToCollisionData(pathtoCollisions)
```

```
    model = YukawaModel()
    manager.registerModel(model)
```

We modify the default value of the spatial grid, the maximum number of iterations in the solution of the equation of motion and the error tolerance of the phase tracer. We specify the location of the pre-computed collision files. Now that we have registered the model, we need to provide input parameters, and a number of settings for the computation of the wall velocity.

```
    inputParameters = {
        "sigma": 0.0,
        "msq": 1.0,
        "gamma": -1.2,
        "lam": 0.10,
        "y": 0.55,
        "mf": 0.30,
    }

    model.modelParameters.update(inputParameters)

    manager.setupThermodynamicsHydrodynamics(
        WallGo.PhaseInfo(
            temperature=8.0, # nucleation temperature
            phaseLocation1=WallGo.Fields([0.4]),
            phaseLocation2=WallGo.Fields([27.0]),
        ),
        WallGo.VeffDerivativeSettings(
            temperatureVariationScale=1.0,
            fieldValueVariationScale=[100.0],
        ),
    )
```

In this model, `inputParameters` directly contain the parameters appearing in the potential and the interactions. In a more realistic scenario, the `inputParameters` would correspond e.g. to physical particle masses and interaction strengths. In the `WallGo.PhaseInfo` object, we provide the nucleation temperature and the positions of the high-temperature and low-temperature phase. The positions of the phases do not need to be exact. `temperatureVariationScale` and `fieldValueVariationScale` are parameters that are used in the construction of an interpolated free energy and when computing the derivatives of the potential. A reasonable choice is the difference between the critical and nucleation temperature and the value of the vacuum expectation value (VEV), respectively.

Now we can compute the wall velocity. First, we return an estimate of the wall velocity in LTE, computed from hydrodynamics only. Then, the wall velocity is obtained by solving the scalar equation

of motion. The first computation ignores the contribution from the out-of-equilibrium fermions, and is therefore very close to vwLTE. Lastly, the wall velocity with out-of-equilibrium contributions is computed.

```python
# ---- Solve wall speed in Local Thermal Equilibrium (LTE) approximation
vwLTE = manager.wallSpeedLTE()
print(f"LTE wall speed: {vwLTE:.6f}")

solverSettings = WallGo.WallSolverSettings(
    bIncludeOffEquilibrium=False,
    # meanFreePathScale is determined here by the annihilation channels,
    # and scales inversely with y^4 or lam^2. This is why
    # meanFreePathScale has to be so large.
    meanFreePathScale=5000.0, # In units of 1/Tnucl
    wallThicknessGuess=10.0, # In units of 1/Tnucl
)

results = manager.solveWall(
            solverSettings
)

print(f"Wall velocity without out-of-equilibrium contributions {results.wallVelocity:.6f}")

solverSettings.bIncludeOffEquilibrium = True

results = manager.solveWall(
            solverSettings
)

print(f"Wall velocity with out-of-equilibrium contributions {results.wallVelocity:.6f}")
```

3. Theoretical background for the terminal wall velocity of an expanding bubble

This section provides an overview of the theoretical background relevant to FOPTs[1] and the bubble wall velocity.

3.1. Effective potential and thermodynamics

Cosmological phase transitions can occur when the free energy, or effective potential, transitions from a higher to a lower energy state at some temperature. In the Standard Model for example, the Higgs

---

[1]While, in principle, many such phase transitions could occur in the history of our universe, we will always focus on a single transition in this work.

field does not break the Electroweak symmetry at high temperatures, but it will transition to a state where the symmetry is broken as the Universe cools.

The temperature at which two minima of the free energy are degenerate is called the critical temperature, T_c. If the degenerate minima are separated by a barrier at the critical temperature, the phase transition will be of first order, and usually proceeds via the nucleation of bubbles. Around T_c, the nucleation rate is immensely exponentially suppressed, and barely any nucleation takes place. However, the exponential suppression shrinks quickly as the temperature decreases, allowing for the phase transition to take place. For the computation of the wall velocity, one is usually interested in the temperature at which non-trivial phase-transition dynamics take place, such as most of the bubble growth and the collisions of hydrodynamic shock fronts. A good estimate for this temperature is the percolation temperature, T_p, at which $\sim 1/e$ fraction of the Universe is in the metastable phase [74,75]. Below, we will refer to the temperature of interest as the nucleation temperature, T_n.[2] The computation of the temperature of interest is beyond the scope of this work and it is an input parameter for `WallGo`. We refer the reader to [77–82] for numerical tools and to [83–87] for the theoretical and computational framework for the high-temperature nucleation rate.

Let us now discuss the effective potential in more detail. We define the scalar fields as $\phi = (\phi_1, \phi_2, \cdots)^T$. We will denote the temperature-dependent effective potential as $V^{\text{eff}}(\phi, T)$, and it is given by

$$V^{\text{eff}}(\phi, T) = V_0(\phi) + V_{\text{higher order}}(\phi, T), \tag{3.1}$$

where V_0 denotes the zero-temperature, tree-level potential, and $V_{\text{higher order}}$ contains higher order corrections, e.g. the temperature-dependent loop corrections. We will denote the field values that minimise the potential at T_n as $v_{\text{HT}}(T_n)$ and $v_{\text{LT}}(T_n)$ for the high-temperature and low-temperature phases respectively.

In earlier computations of the wall velocity [45,52,88], $V_{\text{higher order}}$ was approximated by a one-loop effective potential augmented with (in some cases) thermally resumed masses. In recent years, it has become clear that the one-loop effective potential often does not give accurate predictions for phase transition parameters such as the nucleation temperature and the phase transition strength [57,58], and higher-loop corrections have to be included. The effective potential determines the critical and nucleation temperature, but also enters in the equation of motion of the scalar field, as we will see in section 3.2, and therefore directly affects the value of the wall velocity. As `WallGo` does not restrict the shape of the potential, we can now study the effect of these higher loop corrections on the wall velocity for the first time. This will be discussed in a separate publication.

We can describe the hot plasma in the early Universe in terms of the pressure, which is obtained

---

[2]Strictly speaking, T_n and T_p are not identical, as T_n corresponds to the temperature where the average number of bubbles per Hubble volume is exactly 1. The two quantities are usually very close [13], except in models with a large amount of supercooling [75,76], for which `WallGo` is anyway not applicable.

by evaluating the effective potential at its minima,

$$p_{\text{HT}}(T) = -V^{\text{eff}}(v_{\text{HT}}(T), T), \qquad\qquad p_{\text{LT}}(T) = -V^{\text{eff}}(v_{\text{LT}}(T), T). \qquad (3.2)$$

Other thermodynamic quantities such as the enthalpy density w, the entropy density s, the energy density e and the speed of sound c_s follow from the relations

$$w(T) = T\frac{\mathrm{d}p}{\mathrm{d}T}, \qquad s(T) = \frac{w(T)}{T}, \qquad e(T) = T\frac{\mathrm{d}p}{\mathrm{d}T} - p, \qquad c_s^2 = \frac{\mathrm{d}p/\mathrm{d}T}{\mathrm{d}e/\mathrm{d}T}. \qquad (3.3)$$

3.2. The scalar equation of motion and the thermal plasma

The (classical)[3] equation of motion (EOM) for scalar fields ϕ_i, coupled to the thermal plasma is:

$$\partial^2 \phi_i + \frac{\partial V^{\text{eff}}(\phi, T)}{\partial \phi_i} + \sum_a \frac{\partial m_a^2}{\partial \phi_i} \int_{\mathbf{p}} \frac{1}{2E} \delta f^a(p^\mu, \xi) = 0, \qquad (3.4)$$

where $\int_{\mathbf{p}} = \int \frac{\mathrm{d}^d p}{(2\pi)^d}$, $d = 3$, and the sum runs over all particle species in the plasma and δf^a denotes the deviation from the equilibrium distribution function. The equilibrium contributions of all plasma particles have been absorbed in V^{eff}, which equals the potential of eq. (3.1).

It is useful to choose a coordinate system that follows the expanding bubble. As such our δf^a only depends on the momentum p^μ and the distance from the wall

$$\xi = -\bar{u}_w^\mu x_\mu, \qquad (3.5)$$

where the 4-velocity $\bar{u}_w^\mu$ is perpendicular to the 4-velocity u_w^μ of the wall, *viz.*

$$\bar{u}_w^\mu = \gamma_w(v_w, 0, 0, 1), \qquad\qquad u_w^\mu = \gamma_w(1, 0, 0, v_w). \qquad (3.6)$$

The last term in eq. (3.4) is the friction caused by out-of-equilibrium particles. Since the friction term is proportional to the field-derivative of the mass, in a SM-like plasma, the dominant contribution comes from the top quark, the electroweak gauge bosons and possibly the scalar fields.

The temperature and fluid-velocity profiles must be solved simultaneously with the scalar-field equations of motion, as the equation of motion depends explicitly on the temperature profile. To this aim, we obtain two additional equations from energy-momentum, $T^{\mu\nu}$, conservation in the wall frame (here, the wall is assumed to be planar and moving in the z-direction)

$$T^{30} = w\gamma_{\text{pl}}^2 v_{\text{pl}} + T_{\text{out}}^{30} = c_1,$$
$$T^{33} = \frac{1}{2}(\partial_z \phi_i)^2 - V^{\text{eff}}(\phi, T) + w\gamma_{\text{pl}}^2 v_{\text{pl}}^2 + T_{\text{out}}^{33} = c_2, \qquad (3.7)$$

---

[3]Fundamentally all fields participating in the phase transition and the particles are quantum. Nevertheless, over sufficiently large distances and times $t, L \gg T^{-1}$, scalar fields behave classically [89–91]. This follows since low-energy $E \ll T$ modes are Bose-enhanced, and thus behave classically in accordance with the correspondence principle.

where v_{pl} denotes the local plasma velocity, and γ_{pl} the corresponding Lorentz factor. The out-of-equilibrium components of the energy-momentum tensor, T_{out}^{30} and T_{out}^{33}, are obtained from the following moments of the out-of-equilibrium particle distributions (see [52] for details)

$$\Delta_{mn}^a(\xi) = \int_{\mathbf{p}} \frac{1}{E} E_{\mathrm{pl}}^m p_{z,\mathrm{pl}}^n \delta f^a(p^\mu, \xi) \,, \tag{3.8}$$

where E_{pl} is the energy and $p_{z,\mathrm{pl}}$ the momentum in the z direction measured in the plasma frame. The boundary conditions, c_1 and c_2 are obtained by solving the hydrodynamic equations for the given T_{n} and v_w (cf. e.g. [26, 92–94]). These macroscopic hydrodynamic boundary conditions correspond to $\xi \to \pm\infty$ for the microscopic description of the bubble wall. In contrast, when finding the boundary conditions in the macroscopic context of hydrodynamics, they correspond to the temperature and fluid profile immediately behind and in front of the infinitely thin wall.

The previous discussion demonstrated how the δf^a need to be known to solve the equation of motion of the ϕ_i as well as the temperature and velocity profile. Let us assume that we have obtained a solution for δf^a and the corresponding T_{out}^{30}, T_{out}^{33}. It is most convenient to solve the equations for ϕ_i, T and v_{pl} in the rest frame of the bubble wall, where the profiles are simply a function of $\xi = z$.

A common approach is to describe the profiles of the fields that undergo the phase transition by a Tanh-ansatz:

$$\phi_i(z) = v_{\mathrm{HT},i} + \frac{v_{\mathrm{LT},i} - v_{\mathrm{HT},i}}{2} \left[1 - \tanh\left(\frac{z}{L_i} + \delta_i \right) \right] \,, \tag{3.9}$$

where L_i denotes the wall width, δ_i the offset (the center of the profiles does not need to coincide). The Tanh-ansatz does not solve the scalar EOM eq. (3.4) exactly, but in most known cases the impact of this approximation on the value of the wall velocity is small; see e.g. [45, 95]. `WallGo` uses the Tanh-ansatz, with the intention to make it optional in a future version.

There exist several procedures to determine the parameters L_i and δ_i that give the best solution to the EOM. We chose here to minimise the action that gives rise to the EOM (3.4)

$$S = \int \mathrm{d}z \left[\frac{1}{2} \sum_i (\partial_z \phi_i)^2 + V^{\mathrm{eff}}\left(\boldsymbol{\phi}, \bar{T} \right) - V^{\mathrm{eff}}\left(\bar{\boldsymbol{\phi}}, \bar{T} \right) + \sum_a m_a^2 \bar{\Delta}_{00}^a \right] \,. \tag{3.10}$$

The functions T and Δ will, in general, depend on the scalar fields. However, to recover the appropriate EOMs during the minimisation procedure, these need to be independent of ϕ_i. Therefore, we replace them by $\bar{T}$ and $\bar{\Delta}_{00}^a$, which are the same functions computed using a fixed scalar field profile $\bar{\phi}_i$. To find a solution with the correct T and Δ_{00}^a, the minimisation procedure can be repeated iteratively, each time with updated $\bar{T}$ and $\bar{\Delta}_{00}^a$ computed with the previous estimation of ϕ_i until convergence is attained. Note, that the $-V^{\mathrm{eff}}\left(\bar{\boldsymbol{\phi}}, \bar{T} \right)$ term does not have an effect on the minimisation as it is just a constant. It is there to make the integral over $z \in (-\infty, \infty)$ convergent.

One can show that minimising S with respect to δ_i and L_i is equivalent to solving the moment

equations

$$P_i = \int \mathrm{d}z \frac{\partial \phi_i}{\partial \delta_i} (\mathrm{EOM}) = 0 \,, \tag{3.11}$$

$$G_i = \int \mathrm{d}z \frac{\partial \phi_i}{\partial L_i} (\mathrm{EOM}) = 0 \,. \tag{3.12}$$

The first one can be interpreted as the pressure on the ϕ_i wall, as it controls the position of the ϕ_i wall with respect to the other walls. The second is the gradient of pressure in the wall, and controls the wall thickness.

From this representation in terms of moment equations, it becomes clear that the minimisation procedure is not sufficient to ensure that the total pressure on the wall $P_{\mathrm{tot}} = \sum_i P_i$ vanishes, since we need to enforce $\delta_1 = 0$ to fix the center of the wall. There is therefore no corresponding $P_1 = 0$ equation. However, once the action is minimised with respect to L_i and $\delta_{i>1}$, P_{tot} becomes purely a function of v_w and one can finally solve

$$P_{\mathrm{tot}}(v_w) = 0 \,, \tag{3.13}$$

where it is understood that the L_i and $\delta_{i>1}$ are chosen as to minimise S.

Finally, even for vanishing δf^a, the wall still feels a backreaction force from the plasma that gets heated [53, 54, 96, 97]. For deflagration and hybrid solutions (wall velocity smaller than the so-called Jouguet velocity v_J), this backreaction force is an increasing function of v_w, but for detonation solutions, it decreases with v_w, see e.g. [52, 54]. In many cases, this hydrodynamic backreaction effect is already sufficient to obtain a static deflagration or hybrid solution.[4] For the xSM, it was even shown that the local thermal equilibrium (LTE) approximation gives a reasonable estimate of the wall velocity [52]. However, as seen for the Standard Model with a light Higgs mass in section 6.2 and the Inert Doublet Model in section 6.3, the wall velocity is significantly overestimated in LTE.

3.3. Boltzmann equations for the plasma particles

As argued before, we need to know the distribution functions of the plasma particles to solve the scalar-field equation of motion. As the out-of-equilibrium friction is proportional to the derivative of the particle mass, the particles with the greatest shift in their masses are expected to give the dominant contribution to the friction. Nevertheless, light quarks and gluons will still affect the friction indirectly. For example, out-of-equilibrium gluons help top quarks to equilibrate, as the gluon has a much larger cross-section and can act as a catalyst for the quarks.

---

[4]Simulations of [98] suggest that this static solution does not always get reached in a dynamical simulation, and the wall would in fact run away without additional friction effects.

We will write the distribution function as[5]

$$f^a(\mathbf{p}, \xi) = f^a_{\text{eq}}(\mathbf{p}, \xi) + \delta f^a(\mathbf{p}, \xi)\,, \qquad f^a_{\text{eq}} = \left.\frac{1}{\exp\left[p_\mu u^\mu_{\text{pl}}(\xi)/T(\xi)\right] \pm 1}\right|_{E_a = \mathbf{p}^2 + m_a^2}\,, \tag{3.14}$$

with the plus sign for fermions, and the minus sign for bosons. For now, we will assume that the temperature T and fluid profile u^μ_{pl}, as well as the scalar field profile are all known (in practice the Boltzmann equation and the scalar field, temperature and fluid equations are solved iteratively). We will assume that deviations from equilibrium are small $|\delta f^a| \ll f^a_{\text{eq}}$, which will greatly simplify the computation (the validity of this assumption can be checked in `WallGo`). While such nonlinear contributions were also considered in [99], their effect on the wall velocity was small for the benchmarks considered there.

The evolution of the distribution function in the wall frame is given by the Boltzmann equation

$$\left(p^\mu \partial_\mu + \frac{1}{2}\nabla m_a^2 \cdot \nabla_{\mathbf{p}}\right) f^a(\mathbf{p}, x^\mu) = -\mathcal{C}_a[\mathbf{f}]\,, \tag{3.15}$$

where the second term denotes the classical force term that the particles experience due to the mass change caused by the passing wall. The collision term $\mathcal{C}_a$ of particle a describes the interactions between the particles in the plasma and ensures that the distributions relax to equilibrium far away from the wall.[6] Here, $\mathbf{f}$ without a superscript refers to the full set of distribution functions. The dependence of the collision term on the particle distribution functions makes this system very difficult to solve in practice. In turn, the assumption that the δf^a are small allows for linearising the Boltzmann equations which greatly simplifies the problem. After the linearisation, the Boltzmann equation in terms of ξ becomes

$$\left(-p_\mu \bar{u}^\mu_w \partial_\xi - \frac{1}{2}\partial_\xi(m_a^2)\bar{u}^\mu_w \partial_{p^\mu}\right)\delta f^a = -\mathcal{C}^{\text{lin}}_{ab}[\delta f^b] + \mathcal{S}_a\,, \tag{3.16}$$

where the source term $\mathcal{S}_a$ contains the contributions from the equilibrium distribution

$$\mathcal{S}_a = \left(p_\mu \bar{u}^\mu_w + \frac{1}{2}\partial_\xi(m_a^2)\bar{u}^\mu_w \partial_{p^\mu}\right)f^a_{\text{eq}}\,, \tag{3.17}$$

and the definition of the collision term $\mathcal{C}^{\text{lin}}_{ab}[\delta f^b]$ can be found in eq. (C.2). The linearised Boltzmann equation contains mixing in the collision term. Hence, particle b in eq. (3.16) is not necessarily equal to particle a, and we imply a sum over index b. This mixing effect is usually not considered in computations of v_w.

To render the Boltzmann equations in a numerically solvable form, we follow the procedure developed in [52]. This procedure differs from the seminal work of [45] in the description of the distribution

---

[5]Particles such as the Higgs, quarks, and transverse gluons develop gauge-invariant poles at large momenta $\mathbf{p}^2 \sim T^2$; these can be interpreted as thermal masses. In defining $f_{a,\text{eq}}$, we have implicitly included all such thermal masses to one-loop. See [91] for a discussion on avoiding double-counting.

[6]Far away from the wall, the masses are constant in time, and the equilibrium distribution solves the Boltzmann equation exactly.

functions which uses the so-called fluid ansatz and takes three moments of the Boltzmann equations to obtain kinetic equations for the quantities that parameterise the deviation of equilibrium: the chemical potential, the temperature, and the fluid velocity.[7] In [52] however, the distribution functions are expanded on the orthogonal basis of Chebyshev polynomials, T_i, and read

$$\delta f^a(\chi, \rho_z, \rho_\parallel) = \sum_{i=2}^{M} \sum_{j=2}^{N} \sum_{k=1}^{N-1} \delta f_{ijk}^a \bar{T}_i(\chi) \bar{T}_j(\rho_z) \tilde{T}_k(\rho_\parallel), \tag{3.18}$$

where the compact coordinate χ corresponds to the distance from the wall ξ, ρ_z to the momentum perpendicular to the wall p_z, and $\rho_\parallel$ to the momentum parallel to the wall $p_\parallel$, mapped to the interval $[-1, 1]$. See the end of this subsection and eq. (3.29) below for details). The number of basis polynomials in the spatial direction is M and in the momentum direction is N. Utilizing the so-called restricted Chebyshev polynomials

$$\bar{T}_i(x) = \begin{cases} T_i(x) - T_0(x), & i \text{ even}, \\ T_i(x) - T_1(x), & i \text{ odd}, \end{cases} \tag{3.19}$$

$$\tilde{T}_i(x) = T_i(x) - T_0(x), \tag{3.20}$$

ensures that the δf^a vanish for $\xi, p_z \to \pm\infty$ and $p_\parallel \to \infty$. The convergence of the approximation by expanding in polynomials is exponential in the number of basis polynomials (this is demonstrated in section 5.1). Thus, one can approximate the δf^a with any desired accuracy.

The virtue of this parameterisation is that the Boltzmann equation reduces to an algebraic equation of the coefficients δf_{ijk}^a, $viz.$

$$\sum_{i,j,k} \left\{ \partial_\xi \chi \left[\mathcal{P}_w \partial_\chi - \frac{\gamma_w}{2} \partial_\chi (m^2)(\partial_{p_z} \rho_z) \partial_{\rho_z} \right] \bar{T}_i(\chi) \bar{T}_j(\rho_z) \tilde{T}_k(\rho_\parallel) \delta f_{ijk}^a \right.$$
$$\left. + \bar{T}_i(\chi) \mathcal{C}_{ab}^{\text{lin}} \left[\bar{T}_j(\rho_z) \tilde{T}_k(\rho_\parallel) \right] \delta f_{ijk}^b \right\} = \mathcal{S}_a(\chi, \rho_z, \rho_\parallel). \tag{3.21}$$

The algebraic equation has $(M-1)(N-1)^2$ coefficients, δf_{ijk}^a, from the parametrisation in eq. (3.18). They can be uniquely fixed by demanding that the algebraic equation holds on a discrete grid of points, $(\chi^{(\alpha)}, \rho_z^{(\beta)}, \rho_\parallel^{(\gamma)})$:

$$\chi^{(\alpha)} = -\cos\left(\frac{\pi\alpha}{M}\right), \qquad\qquad \alpha = 1, \cdots, M-1, \tag{3.22}$$

$$\rho_z^{(\beta)} = -\cos\left(\frac{\pi\beta}{N}\right), \qquad\qquad \beta = 1, \cdots, N-1, \tag{3.23}$$

$$\rho_\parallel^{(\gamma)} = -\cos\left(\frac{\pi\gamma}{N-1}\right), \qquad\qquad \gamma = 0, \cdots, N-2. \tag{3.24}$$

---

[7]This approach was generalised to a larger number of moments in [51].

This precise choice for the grid points ensures the exponential convergence of the spectral method [52, 100]. See fig. 3 below for a numerical example.

With the choice of the grid, $(\chi^{(\alpha)}, \rho_z^{(\beta)}, \rho_\parallel^{(\gamma)})$, the algebraic equation simply becomes a (dense) matrix equation,[8]

$$\left(\mathcal{L}[\alpha, \beta, \gamma; i, j, k]\delta_{ab} + \bar{T}_i(\chi^{(\alpha)})\mathcal{C}_{ab}[\beta, \gamma; j, k] \right) \delta f_{ijk}^b = \mathcal{S}_a[\alpha, \beta, \gamma] \,, \tag{3.25}$$

in the indices $\{\alpha, \beta, \gamma\}$ and $\{i, j, k\}$, and where repeated indices are summed. Here, we have defined the Liouville operator, the collision operator and the source as

$$\mathcal{L}[\alpha, \beta, \gamma; i, j, k] \equiv \partial_\xi \chi^{(\alpha)} \left[\mathcal{P}_w^{(\alpha,\beta,\gamma)} \partial_\chi - \frac{\gamma_w}{2} \partial_\chi (m^2)^{(\alpha)} (\partial_{p_z} \rho_z^{(\beta)}) \partial_{\rho_z} \right] \bar{T}_i(\chi^{(\alpha)}) \bar{T}_j(\rho_z^{(\beta)}) \tilde{T}_k(\rho_\parallel^{(\gamma)}) \,, \tag{3.26}$$

$$\mathcal{C}_{ab}[\alpha, \beta; j, k] \equiv \mathcal{C}_{ab}^{\text{lin}} \left[\bar{T}_j(\rho_z^{(\alpha)}) \tilde{T}_k(\rho_\parallel^{(\beta)}) \right] \,, \tag{3.27}$$

$$\mathcal{S}_a[\alpha, \beta, \gamma] \equiv \mathcal{S}_a\left(\chi^{(\alpha)}, \rho_z^{(\beta)}, \rho_\parallel^{(\gamma)} \right) \,. \tag{3.28}$$

The derivatives of the basis polynomials arising can be re-expressed in terms of linear combinations of basis polynomials.

Now that we have the Boltzmann equation in a numerically solvable form, let us discuss the mapping between compact and physical coordinates in `WallGo`. For the momentum directions, these are the same as in [52],

$$\rho_z(p_z) = \tanh\left(\frac{p_z}{2T} \right) \,,$$
$$\rho_\parallel(p_\parallel) = 1 - 2 \exp\left(-\frac{p_\parallel}{T} \right) \,. \tag{3.29}$$

This ensures that the solution vanishes exponentially as $|\mathbf{p}| \to \infty$, with a decay length of T.

In the spatial direction, `WallGo` uses a more sophisticated mapping than [52] to better model the different scales involved in the solution. The solution of the Boltzmann equation should decay exponentially with decay lengths $l_\pm$ when $\xi \to \pm\infty$, with $l_- \neq l_+$ in general (we refer to these regions as the *tails*). Furthermore, the source term is only nonzero within the bubble wall, $|\xi| \lesssim L_{\text{wall}}$, where L_{wall} is the width of the whole wall. The mapping is therefore constructed in such a way that these three scales are properly resolved.

In practice, the scales $l_\pm$ are estimated from the wall velocity and the parameter `meanFreePathScale`, which must be provided by the user; see section 5.2 for details on how to choose this parameter properly. Then, `WallGo` estimates the parameter $L_{\text{Grid}} \sim L_{\text{wall}}$ from the field profiles $\phi_i(\xi)$. The mapping also depends on the parameters `smoothing` and `ratioPointsWall`, which control how smooth the transition between the different regions is and the approximate ratio of points used to resolve the wall in the interval $\xi \in [-L_{\text{Grid}}, L_{\text{Grid}}]$; see section 5.3 for more details.

---

[8]The ordering of the indices here is the same as in `WallGo`. Note, that the index ordering in `WallGoCollision` is transposed, i.e. $\mathcal{C}_{ab}[\alpha, \beta; j, k] \to \mathcal{C}_{ab}[j, k; \alpha, \beta]$.

3.4. Collision terms

In the Boltzmann equation for a given particle species labelled by a, the collision term describes the rate at which particles a with given momentum P_1 are lost and created due to elastic and inelastic scattering processes. In principle, there exist collision processes which take an initial a particle and $N_{\rm in} - 1$ other particles into a final state of $N_{\rm out}$ particles, i.e. a process $N_{\rm in} \to N_{\rm out}$. However, at leading logarithmic order, only $2 \to 2$ scatterings contribute [101–103]; see section 3.4.1.

The linearised collision terms are linear functionals acting on a set of distribution functions. In principle, these distribution functions carry all the quantum numbers of the corresponding particles, but in cases where certain quantum numbers are irrelevant to the scattering, they can safely be averaged over, e.g. for helicity in QCD. We label the distribution functions by an index $a, b, c, ..$, each denoting a group of degrees of freedom, potentially averaged some quantum numbers. Then, the linearised collision term appearing in eq. (3.21) is given by

$$\mathcal{C}_{ab}^{\rm lin}[\bar{T}_j(\rho_z)\tilde{T}_k(\rho_\parallel)] = \frac{1}{4} \sum_{cde} \int_{\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4} \frac{1}{2E_2 2E_3 2E_4}(2\pi)^4\delta^4(P_1 + P_2 - P_3 - P_4) \tag{3.30}$$

$$\times |M_{ac\to de}(P_1, P_2; P_3, P_4)|^2 f^a f^c f^d f^e (\delta_{ab}F_a^c + \delta_{cb}F_c^a - \delta_{db}F_d^e - \delta_{eb}F_e^d)\bar{T}_j(\rho_z)\tilde{T}_k(\rho_\parallel),$$

where ρ_z and $\rho_\parallel$ are the momenta of the b-particle. We have also introduced the notation

$$F_b^a = \frac{e^{E_a/T}}{(f^b)^2}. \tag{3.31}$$

By denoting $P_a = (E_a, \mathbf{p}_a)$, the particles $\{a, c, d, e\}$ carry four-momenta $\{P_1, P_2, P_3, P_4\}$ in that order. $|M_{ac\to de}(P_1, P_2; P_3, P_4)|^2$ are (squared) matrix-elements averaged over the degrees of freedom specified by index a and summed over all other degrees of freedom specified by c, d, e,

$$|M_{ac\to de}(P_1, P_2; P_3, P_4)|^2 \equiv \frac{1}{N_a} \sum_{a_i\in a} \sum_{c_i\in c} \sum_{d_i\in d} \sum_{e_i\in e} |\mathcal{T}_{a_i c_i \to d_i e_i}(P_1, P_2; P_3, P_4)|^2, \tag{3.32}$$

where $\mathcal{T}$ is a normal scattering amplitude. The factor of $\frac{1}{N_a}$ arises from the definition of f^a as the occupancy of a-particles, averaged over the set of unimportant quantum numbers labelled by a_i. The choice of quantum numbers to average over depends on the physical model and situation considered. Considering gluons, for example, here a_i could run over its possible helicities and colours, in which case $N_a = 16$. Instead considering a left-handed top-quark, averaging over colour and particle and antiparticle gives $N_a = 6$.

The factor of $\frac{1}{4}$ on the right hand side of eq. (3.30) is the product of two factors of $\frac{1}{2}$: one from the usual relativistic invariant momentum integration factor $\frac{1}{2E_a}$, and the other which is a symmetry factor if d and e are identical, or a factor which compensates the double-counting in the sum over species if d and e are different [101, 103].

The challenge is now to evaluate the 9-dimensional momentum integral for all $n_p^2(N-1)^4$ components of the collision tensor on the grid; see eq. (3.27). The number of out-of-equilibrium particles is denoted

21

as n_p. Fortunately, the task becomes easier because half of the components of $\mathcal{C}_{ab}^{\rm lin}[\bar{T}_j(\rho_z)\tilde{T}_k(\rho_\parallel)]$ are redundant,

$$\mathcal{C}_{ab}^{\rm lin}[\bar{T}_j(-\rho_z)\tilde{T}_k(\rho_\parallel)] = (-1)^j\mathcal{C}_{ab}^{\rm lin}[\bar{T}_j(\rho_z)\tilde{T}_k(\rho_\parallel)]\,, \tag{3.33}$$

which follows from the properties of the Chebyshev polynomials. Moreover $\mathcal{C}_{ab}^{\rm lin}[\bar{T}_j(0)\tilde{T}_k(\rho_\parallel)] = 0$ for odd j, or when $\rho_z = \rho_\parallel = 0$. In the absence of vacuum masses, the momenta can be rescaled as $P_i \to P_i/T$, such that the temperature only appears as a pre-factor in front of the collision term:

$$\mathcal{C}_{ab}^{\rm lin}[\delta f^b] = T^2\hat{\mathcal{C}}_{ab}^{\rm lin}[\delta f^b]\,. \tag{3.34}$$

We will use this rescaling in `WallGo` and leave the inclusion of vacuum masses to future work.

Out of the nine integration dimensions, four are trivial due to the momentum-conserving δ-function. In appendix C, the final form of the collision integral is derived. The remaining integrals are performed using a Monte Carlo method, see appendix C.

3.4.1. The leading-logarithmic approximation

In the integration of collision terms, kinematic enhancements can increase the parametric size of certain terms over the naive expectation based on counting powers of couplings. In particular, the dominant processes involve $2 \to 2$ scatterings of small angle θ, for which the cross-section behaves as $d\sigma \sim \frac{d\theta}{\theta}$. Thermal screening softens the divergence at small θ, and leads to a multiplicative $O(|\log g|)$ logarithmic enhancement dependent on the screening length [101–103].

State-of-the-art wall velocity computations use the leading-log approximation when calculating the collision rates [52]. Such an approximation correctly accounts for all terms which are logarithmically enhanced due to the small angle scatterings, and may truncate or otherwise distort terms which are not so enhanced. We adopt one particular, common implementation of the leading-logarithmic approximation, wherein masses of the external particles are neglected, and internal lines are regulated by asymptotic thermal masses.[9] This approximation was found to lead to an overestimate of the quark diffusion constant by 50% [102], consistent with the expected $O(|\log g|^{-1})$ relative errors.

Asymptotic thermal masses result from soft resummation on the lightcone. Such resummation becomes necessary when not all momentum components are taken as soft ($P_\mu \sim gT$) but rather $P^2 \sim (gT)^2$ [103, 104] and therefore close to the lightcone. This also happens in a hard regime where $P \sim T$ which is the case for the hard momenta assumed in the matrix elements. The corresponding resummation near the lightcone gives rise to a modified dispersion relation $\omega^2 = k^2 + m_\infty^2$, where we denote m_∞ as the corresponding asymptotic mass. For scalars, asymptotic and (static) thermal masses coincide but are genuinely different for vector fields and fermions and at LO are

$$m_{V,\infty}^2 = \frac{1}{2}m_{\rm D}^2\,, \qquad\qquad m_{q,\infty}^2 = 2m_q^2\,, \tag{3.35}$$

---

[9]In some of our model files, we deviate from this prescription to reproduce results obtained in the literature.

where m_{D} is the corresponding Debye mass and m_q the standard fermionic thermal mass. See [105] for the asymptotic masses of vector fields at LO and NLO for generic models. If not stated otherwise, we employ asymptotic masses throughout our analyses.

Going beyond leading logarithmic accuracy introduces a number of new phenomena and computational difficulties [102, 103]. Collision integrals for naively forbidden $1 \to 2$ scatterings must be included, momentum dependence and scalar field background dependence must be resummed in the particle self-energies, and gauge and fermion fields experience non-local Landau damping. Including these effects is beyond the scope of the current work, but we foresee an investigation of effects beyond leading logarithmic approximation in an upcoming publication.

4. What happens in the code

`WallGo` consists of three parts:

- `WallGoMatrix`, a `Mathematica` code that determines the matrix elements,

- `WallGoCollision`, a `C++` code (with `Python` wrapper) that computes the collision integrals,

- `WallGo`, the `Python` code that that solves the Boltzmann equations and equations of motion of the scalar field(s).

Detailed documentation of the code can be found in the online documentation, `https://wallgo.readthedocs.io`. In this section, we will merely give a compact overview of the different parts of the code and how they depend on each other.

4.1. WallGoMatrix

`WallGoMatrix` is a `Wolfram Mathematica` tool that generates matrix elements needed for out-of-equilibrium computations in the `WallGo` framework. In `WallGo`, users can supply their own matrix elements or generate them using `WallGoMatrix`. The model generation in `WallGoMatrix` is based on `DRalgo` [1]. The procedure to generate matrix elements is:

(i) **Loading packages:** `WallGoMatrix` requires the files `WallGoMatrix.m`, `matrixElements.m`, and `modelCreation.m` along with its dependency on `GroupMath`. Load the package by setting the directory and using the command `<<WallGoMatrix'`.

(ii) **Defining the model:** Define gauge groups (e.g., `SU3` and `SU2`) and particle representations. Fermion and scalar representations are specified using their Dynkin coefficients.

(iii) **Spontaneous symmetry breaking:** The tool supports spontaneous symmetry breaking by defining the VEV of scalars. Use the command `SymmetryBreaking[vev]` to display an indexed
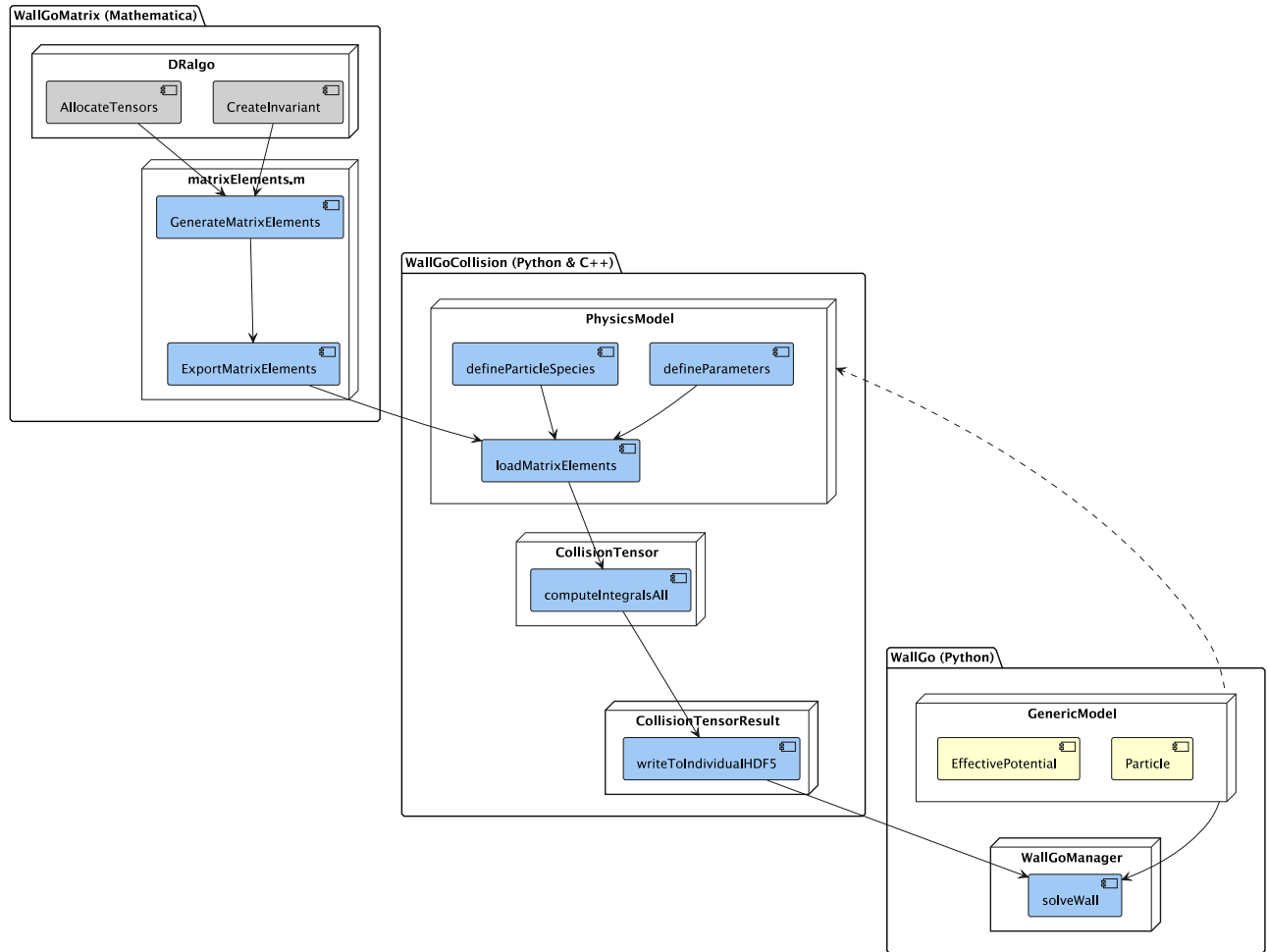
Figure 1: Diagram showing the main parts and functions required to compute the bubble wall velocity for a given particle physics model. The main `WallGo` package solves for the wall velocity with input from the `WallGoCollision` package, which in turn has input from the `WallGoMatrix` package.

       list of particles according to their symmetry-breaking-induced masses. This command is needed if couplings are VEV-dependent or if one wants to use the `CreateParticle` command.

(iv) **Specifying particles:** Particles that can be taken in- or out-of-equilibrium are specified by creating their representations and grouping them into distributions using e.g. the `CreateParticle` command. For example, left- and right-handed quarks, gluons, and scalar Higgs particles can be defined this way. See listing L.39 or the example `2scalars.m` for scenarios that bypass the `CreateParticle` command.

(v) **Generating matrix elements:** Matrix elements for specific particles are generated with the command `ExportMatrixElements`. Options for normalisation, truncation at leading logarithmic

24

order, and output formats (e.g., `.json`, `.txt`) are available through the `OptionPattern`.

The corresponding main functions of `WallGoMatrix` are documented in appendix A. Appendix B details the internal computation of the matrix elements. Here, we exemplify a typical matrix element file which can be used to compute the collisions—for QCD given in the example `qcd.m`. In this scenario, only the top quark is out-of-equilibrium, and only strong interactions are included in the matrix elements.

The `WallGoMatrix` default format is `.json` and consists of two main keys pointing to arrays

L.11

```
{
  "particles": array,
  "matrixElements": array
}
```

The `particles` array contains a list of particle objects, where each particle is defined by its index and name

L.12

```
"particles":[
  {
    "index": number,
    "name": string
  }
]
```

While the naming does not need to match the naming in the `WallGo` model file, the indexing should. For QCD this corresponds to

L.13

```
"particles":[
  {
    "index":0,
    "name":"Top"
  },
  {
    "index":1,
    "name":"Gluon"
  },
  {
    "index":2,
    "name":"LightParticle"
  }
]
```

25

where the `LightParticle` contains all light quarks.

The `matrixElements` array contains objects that define the interactions between external particles, the matrix elements. Each object has the following properties:

L.14

```
"matrixElements":[
  {
    "externalParticles": array[number],
    "parameters": array[string]
    "expression": string
  }
]
```

The order in the `externalParticles` array corresponds to the particles a, c, d, e in eq. (3.30). Strictly speaking, only the out-of-equilibrium particles need to appear on the first index. Matrix elements with in-equilibrium particles on the first index will simply be ignored. For the example of QCD, [0,1,0,1], denotes scattering of a top quark with a gluon. Concretely, the output corresponds to

L.15

```
"matrixElements":[
  {
    "externalParticles":[0, 0, 0, 0],
    "parameters":["gs","mg2"],
    "expression":"(16*gs^4*(_s^2 + _t^2))\/(3*(mg2 - _u)^2) + (16*gs^4*(_s^2 + _u^2))\/(3*(mg2 -
    _t)^2)"
  },
  {
    "externalParticles":[0, 1, 0, 1],
    "parameters":["gs","mg2","mq2"],
    "expression":"(-64*gs^4*_s*_u)\/(9*(mq2 - _u)^2) + (16*gs^4*(_s^2 + _u^2))\/(mg2 - _t)^2"
  },
  {
  ...
  },
  {
    "externalParticles":[1, 1, 1, 1],
    "parameters":["gs","mg2"],
    "expression":"(18*gs^4*(_s - _t)^2)\/(mg2 - _u)^2 + (18*gs^4*(_s - _u)^2)\/(mg2 - _t)^2"
  }
]
```

The strong coupling constant, `gs`, is the only relevant coupling and must match the corresponding name in `WallGo`. If other couplings or parameters are involved, they must be specified for each matrix element. The Mandelstam variables are represented by `_s`, `_t`, `_u`, while `mq2`, `mg2` denote the masses of quarks and gluons in the propagators. In the leading-log approximation, these masses can be treated

as thermal masses.

4.2. WallGoCollision

Collision integrations are typically by far the most computationally intensive part of the WallGo wall velocity pipeline. WallGoCollision performs the integrations and related operations in native C++ for maximal performance and versatility. WallGoCollision compiles to a Python extension module for seamless interoperation with the main WallGo package, but can also be used as a pure C++ library.

The purpose of WallGoCollision is to compute elements of the collision tensor (3.27). Integrals on different grid points are fully independent and parallelise trivially. WallGoCollision supports OpenMP for parallel evaluation with shared memory, and has been tested on up to 96 cores.

To compute collision integrals with WallGoCollision, the user has to

(i) Create a PhysicsModel object containing your particle and model parameter definitions. This should include all particles and parameters that appear in your collision matrix elements, including any light species that can be approximated as remaining in equilibrium but still appear as external particles in collision processes involving out-of-equilibrium particles.

(ii) Load in matrix elements to the model in a symbolic form. The Mandelstam variables have to be denoted as _s, _t, _u. The assumed physics conventions are as in (3.32). The output files of WallGoMatrix can be used directly as long as the model definition is compatible.

(iii) Use the model to create a CollisionTensor object and pass it the size N of your momentum grid. CollisionTensor holds the integrals in an unevaluated but otherwise ready form.

(iv) Call computeIntegralsAll() from your CollisionTensor object to perform the integrations. Once finished, the results can be stored in binary .hdf5 format and loaded into the WallGo Boltzmann solver.

Unless otherwise specified, all classes described in this section can be found in the WallGoCollision Python module, and in the wallgo namespace in native C++ application programming interface (API).

Model definition in step (i) is done by filling in a ModelDefinition helper object and passing it to the PhysicsModel constructor. Parameters must be defined as (name, value) pairs; for example, to define symbol "g" with initial value 0.42: modelDefinition.defineParameter("g", 0.42). Any appearance of the symbol g in matrix elements will then be replaced with value 0.42 during collision integration. Dimensionful parameters must always be given in units of the temperature. In particular, particle masses appearing in propagators of matrix elements are also treated as "model parameters" and must be given as dimensionless, floating point numbers.

A particle species is defined by passing a ParticleDescription object to your ModelDefinition instance. A ParticleDescription consists of a unique string name, unique integer identifier, matching the index of the matrix element file, type specifier (boson or fermion) and a flag for keeping the

27

particle species in thermal equilibrium. The last property can be used to reduce the number of collision integrations for models containing particle species for which deviations from equilibrium are negligible. The default behavior is to treat all particles as ultrarelativistic ($E_i = |p_i|$ during integration). This allows for heavy performance optimisations in collision integration. To go beyond the ultrarelativistic approximation, you can specify a mass function that computes the mass-square of this particle species from given model parameters, again in units of the temperature.

`WallGoCollision` performs the integrations using the Vegas algorithm [106] which is an adaptive Monte Carlo integrator.[10] Configuration settings for the integrator are available in the class `IntegrationOptions`, including the upper limit to use for momentum integration and error tolerances for the Monte Carlo method. They can be passed to a `CollisionTensor` object by calling its `setIntegrationOptions()` function.

Examples of collision generation can be found in the `WallGo Models` folder. In all these examples, the collision generation and the computation of v_w are combined, though they can also be run independently.

Limitations The release version of `WallGoCollision` comes with the following limitations regarding the physics content:

- Only $2 \rightarrow 2$ collision processes are supported.

- All momentum dependence in the matrix elements must be expressed in terms of the Mandelstam variables s, t and u.

- All physics parameters that appear in the matrix elements must be constant, floating point valued numbers. Dimensionful parameters must be given in units of the temperature. Concretely, `WallGoCollision` works in units of $T = 1$. The limitation is thus that parameters that vary along the momentum or position grid are not supported.

The first two limitations are important for going beyond the leading logarithm approximation. The relevance of the third limitation is model dependent.

4.3. WallGo

Now that the collision terms have been computed, `WallGo` can be used to compute the wall velocity. In this subsection, we will discuss certain details of this computation. The computation is performed by a number of classes, such as `Thermodynamics`, `EOM`, `BoltzmannSolver`, etc. The user does not need to keep track of all these classes, as this is done by the `WallGoManager` class, which initialises them in the appropriate order. In what follows, we will refer to the instance of the `WallGoManager` as `manager`.

We will now give some more details about the computation, mainly focussing on the steps the user needs to perform for computing the wall velocity. A more detailed documentation can be found

---

[10]Internally we use the GSL [107] implementation of Vegas.

in `https://wallgo.readthedocs.io`, which will also be up to date with the most recent release of the code. For illustration, we also refer the users to the examples in the `Models` folder. Note, that most of the examples rely on the file `Models/wallGoExampleBase.py`, which gives a template for the computation with a specific model.

Configuration To compute v_w, `WallGo` needs a number of model-independent configuration settings, such as the size of the grid on which the momentum and position are discretised, and error tolerances. These are stored in a `Config` object in the `WallGoManager` class, which is initialised with reasonable default values with `WallGo.Config()`. This class contains an instance of the smaller dataclasses `ConfigGrid`, `ConfigEOM`, `ConfigHydrodynamics`, `ConfigThermodynamics` and `ConfigBoltzmannSolver`, which contain settings for the corresponding class. Each individual setting can be accessed with, for example, `config.configGrid.momentumGridSize`. Users can also read in a custom configuration file by running `config.loadConfigFromFile(<Path to configuration file>)`. See the `Models/ManySinglets/manySingletsConfig.ini` file for a concrete example of a configuration file that can be loaded by `Config`. It also contains a list of all the parameters stored in the `Config` class. The user's custom configuration file should have the same format as this example (note that if some settings are absent from the file their value will not be updated compared to their default value), which can be found in the source code and the online documentation. `WallGo` throws an error and crashes if the path is not correct. Let us also stress that the number of Chebyshev polynomials `momentumGridSize` *must always be odd*.

Setting up the `WallGoManager` To prepare the `manager` to compute the wall velocity, the user must call `manager.setupThermodynamicsHydrodynamics()`,[11] which creates and initialises the objects `Thermodynamics` and `Hydrodynamics` required for the calculation. This function takes as parameter a `PhaseInfo` object (to be described in the next paragraph) and a `VeffDerivativeScales` object. This data class contains the two model-dependent quantities

- `temperatureVariationScale`: the temperature scale over which the potential changes by $\mathcal{O}(1)$ (often $T_c - T_n$ is a good estimate). This number is used to estimate the step size in the phase tracer and when taking the temperature derivatives of the potential. If it is chosen too large, `WallGo` could crash as the phase tracer tries to probe a region where the phase no longer exists. It might also falsely return a run-away wall. If it is chosen too small, the phase tracing will take longer than necessary. In both cases, the temperature derivatives will become less accurate.

- `fieldValueVariationScale`: the field scale over which the potential changes by $\mathcal{O}(1)$. This can be given as an array with the length of the number of fields, or as a single float. Usually, the value of the VEV is a good estimate. Choosing a too large or too small value will result in

---

[11]The object `manager.config` must contain the desired settings *before* calling this function. The settings can be changed individually or by loading a configuration file (see discussion in the previous paragraph).

inaccuracies in the derivatives, though the effect should be minimal dependence if the order of mangitude is correct.

These parameters are used by `WallGo` to ensure that it uses an optimal step size when computing the derivatives of the effective potential by finite differences.

Phase information `WallGo` needs to know between which two phases the phase transition takes place, and at what temperature. This input is provided via a `WallGo.PhaseInput` object, which is a data class, holding the nucleation temperature, and the (approximate) values of the VEVs of all fields in the two phases. The high-temperature (outside) phase is listed before the low-temperature (inside) phase. Once `manager.setupThermodynamicsHydrodynamics()` has been called, the precise position of the phases is then determined. It is verified that the phases are not identical, and that the potential energy of the low-temperature phase is smaller than the potential energy of the high-temperature phase. If any of these conditions fails, an error will be thrown and the computation cannot continue. Note, that it is the responsibility of the user to choose an appropriate nucleation temperature, as `WallGo` does not perform additional checks.

  `WallGo` now traces the positions of the fields in the minima for the two phases as a function of the temperature. This serves two purposes. First, it determines for both phases the temperature range for which they exist, obtaining a maximum and minimum temperature for both. Second, it constructs an interpolated function of the value of the effective potential along the way, which speeds up the computation. To limit the time spent on this step, the phases are not traced over their entire range of existence. The minimum and maximum temperature are determined by the configuration file. If a too small maximum temperature is chosen, the wall velocity might not be found (`WallGo` will print a warning if the choice of maximum temperature restricts the range of v_w).

Registration of the model and the effective potential The user needs to define a `model`, and register it with the `WallGoManager` by running `manager.registerModel(model)`. The model inherits from the abstract base class `GenericModel`. It needs to have the following properties:

- `outOfEquilibriumParticles`: a list of all the out-of-equilibrium particles that enter the friction and Boltzmann equations. See section 2.1 for a concrete example of an object of the `Particle` class. The particles are labeled by a string identifier, e.g. `"top"`. The `index` should correspond to the `index` in the corresponding matrix elements file. `msqVacuum` denotes the mass squared without thermal corrections and `msqDerivative` its field-derivative. `statistics` should be equal to `"Boson"` or `"Fermion"`. `totalDOFs` is the total number of degrees of freedom. E.g. tor a top quark with only SU(3) interactions this would be 12, but if we distinguish left-handed and right-handed top quarks they would both have 6 DOFs. The particles are added to the model with the `GenericModel` member function `addParticle`.

- `fieldCount`: property function that returns the number of (scalar) fields participating in the phase transition.

- `getEffectivePotential`: a member function which returns the effective potential for the field(s) undergoing the phase transition (see below).

Typically, users will add additional functions and properties to the model, such as `modelParameters` and a function converting input parameters to model parameters.

The `effectivePotential` is also constructed by the user. It should have the same `fieldCount` as the model and it should have a member function `evaluate`, which gives the value of the potential as a function of the field(s) and temperature. Usually, the potential uses the same `modelParameters` as the `model`, but this is implemented by the user. The `effectivePotential` should also contain an estimate of the relative error of the potential in `effectivePotentialError`. Note, that it is important to include field-independent contributions to the effective potential (e.g. the T^4 contribution). These terms are essential for a correct description of the hydrodynamics of the plasma. Initialisation of the potential class is done inside of the model.

Some functions that are commonly used in the construction of the effective potential are collected in `src/PotentialTools/EffectivePotentialResum`, such as the Coleman-Weinberg potential and the one-loop thermal functions $J_{b,f}$. See e.g. the Inert Doublet and xSM model files for examples. The xSM model file also demonstrates how one can load a custom table for the $J_{b,f}$ (we do not advise this in principle, since `WallGo` has its own interpolation tables, but we can imagine that this is useful for comparing to other implementations).

Computation of the wall velocity The wall velocity and wall parameters can now be computed. For deflagration or hybrid solutions, one calls `manager.solveWall(WallSolverSettings)`, where `WallSolverSettings` is a data class containing the parameters

- `bIncludeOffEquilibrium`: boolean which determines whether the out-of-equilibrium contributions should be included (True) or not (False),

- `meanFreePathScale`: an estimate of the *longest* mean free path of out-of-equilibrium plasma particles, given in units of $1/T_n$. This number is used to estimate the extent of the solution for δf (see section 5.2 for more details). The default value is 50, which is roughly equal to $1/\alpha_s^2$, with α_s the strong coupling constant. In models having weaker interactions, this value likely has to be adjusted.

- `wallThicknessGuess`: an initial guess of the wall thickness, given in units of $1/T_n$. This number is model-dependent, but should be larger than 1 in order for a gradient expansion for the bubble wall to be appropriate. The default value is 5.

The function `solveWall` relies on the fact that the pressure is an increasing function of v_w when $v_w < v_J$. This ensures that any deflagration or hybrid solution, if it exists, will be unique and can be bracketed between 0 and v_J.[12] This property allows for the use of fast and robust root-finding algorithms such as Brent's method. It is first verified that the pressure at the minimum velocity is negative, and positive at the maximum velocity. If the minimum pressure is positive, this indicates a problem in the effective potential, as the low-temperature phase should have lower potential energy than the high-temperature phase. `WallGo` will return a `WallGoResults` object with `solutionType =` `ESolutionType.ERROR`. If the maximum pressure is negative, the friction is not large enough to stop the wall from accelerating and `WallGo` will return `solutionType = ESolutionType.RUNAWAY` as this typically means that the wall would continue accelerating until reaching runaway speeds. If the pressures do have the correct sign, the wall parameters are determined by finding the solution which yields zero pressure on the wall. For every attempted v_w, the pressure is computed by solving the Boltzmann equation and minimising the action (3.10) with a relative error tolerance for the pressure defined in `WallGo.config`. The maximum number of iterations for finding the pressure for a particular value of v_w is also specified in `WallGo.config`.

The `solveWall` function returns a `WallGoResults` object, containing parameters such as the wall velocity and width(s), but also the solution to the Boltzmann equation and the hydrodynamic boundary conditions. The results for wall velocity, widths and offsets can be assessed via `results.wallVelocity`, `results.wallWidths` and `results.wallOffsets`, respectively. `results.wallVelocityError` gives an estimate of the error of v_w, resulting from the finite grid size in the Boltzmann solver. `WallGo` will throw a warning if the truncation error is estimated to be large. Whenever this happens, the user could increase the momentum or position grid size or adjust `meanFreePathScale`.

To find the wall velocity for a detonation, $v_w > v_J$, one calls `manager.solveWallDetonation()`. For detonations, the pressure is generally not a monotonous function of v_w, and one can therefore find several solutions with zero pressure. In practice, only the roots with increasing pressure with respect to v_w are stable, so the function will only return these roots in a list of `WallGoResults` objects. Even if it is technically possible to have several stable solutions for the same model, most cases will either have zero or one solution. To avoid loosing time looking for a second root that usually does not exist, the user can pass the parameter `onlySmallest=True`, which will force the solver to stop the calculation after finding the first root with the smallest v_w.

Three situations can prevent `manager.solveWallDetonation()` from finding a solution—each returning a list containing a single `WallGoResults` object with attribute

- `solutionType = ESolutionType.RUNAWAY`. If the pressure is consistently negative, the friction from the plasma is insufficient to stop the wall, leading to a runaway solution. Note, a deflagration or hybrid solution may exist but will not be returned here.

---

[12]The allowed velocity range is determined by `Hydrodynamics`. In principle it is given by $v_w \in [0, v_J]$, but for strong phase transitions a minimum velocity might exist, and for a limited range of existence of the thermodynamic phases, the maximum velocity might be smaller than v_J.

- `solutionType = SolutionType.DEFLAGRATION`. If the pressure is consistently positive, friction is too strong to allow for a detonation or runaway solution. Here, the returned object indicates that only deflagration or hybrid solutions are possible.

- `solutionType = ESolutionType.DEFLAGRATION_OR_RUNAWAY`. If the pressure is positive at $v_w = v_J$ and negative at $v_w = 1$, and if there is no stable solution (there is however an unstable one), the outcome is uncertain and would require a time-dependent analysis.[13]

A third option for the computation of v_w is `WallGoManager.wallSpeedLTE`, which returns an approximation of the wall velocity, given by local thermal equilibrium, using hydrodynamics only. This function should give approximately the same result as `manager.solveWall(bIncludeOffEq = False)`, but is much faster. Note, that this function will never return a detonation solution.

The kinetic energy fraction One of the goals of `WallGo` is to improve the prediction of GW signals generated in FOPTs. An essential parameter in this prediction is the energy budget K [13, 27, 32, 63] for GWs from sound waves. This quantity describes the ratio of kinetic energy in the fluid to the energy density of the symmetric phase. It can easily be obtained from the `Hydrodynamics` module of `WallGo`, which returns the related efficiency factor κ via `Hydrodynamics.efficiencyFactor(vw)`. The energy budget is then obtained from

$$K = \frac{3\alpha_n w_n}{4e_n}\kappa \,, \tag{4.1}$$

where the subscript n denotes that the quantity is evaluated at the nucleation temperature. α_n denotes the strength of the phase transition, and we use the definition of [64, 65].

Units `WallGo` works with natural units, but otherwise does not enforce units. The user can thus choose units that are appropriate for their physics application, by defining the effective potential and giving input such as the nucleation temperature, mass parameters and field expectation values in the units of choice (e.g. GeV, TeV). The only constraint is that one has to use the same units throughout the whole calculation.

Some functions will ask for quantities given in particular units. If that is the case, it will be clearly stated in the documentation. For example, the `WallSolverSettings` data class, which is used as input by `manager.setupWallSolver()`, requires the `meanFreePathScale` and `wallThicknessGuess` to be given in units of $1/T_n$.

---

[13]The positive pressure at $v_w = v_J$ should in principle stop the wall from accelerating and force it to be a deflagration or hybrid solution. But if for some reason the wall is able to overcome this pressure barrier (which can happen in a time-dependent analysis) and reach the region with negative pressure, the wall will become a runaway solution.

Multithreading To increase performance, `WallGo` can be run in parallel to analyse multiple models at once. This can be especially helpful when doing a scan of the parameter space where many parameter points must be sampled. However, many `numpy` and `scipy` functions used by `WallGo` use multiple threads by default, which saturates the CPU and makes running `WallGo` in parallel very slow. Furthermore, these multithreaded functions must create and destroy threads frequently, which creates a lot of overhead and renders it quite inefficient. Therefore, we strongly suggest the user to turn off `numpy`'s multithreading. This can be done by running the following before importing `numpy`, `scipy` or `WallGo`:

L.16

```
import os
os.environ['OPENBLAS_NUM_THREADS'] = '1'
```

when using `OpenBLAS`, or

L.17

```
os.environ['MKL_NUM_THREADS'] = '1'
```

for MKL (whether `OpenBLAS` or `MKL` is used can be found by running `numpy.__config__.show()`). However, some integrated development environments (IDEs) use their own multithreading settings, in which case the previous command might not work.

5. Convergence tests

As with any numerical solver, usage of `WallGo` depends on a number of meta-parameters, which determine the accuracy of the results. These enter both in the calculation of the collision integrals, and in the solution of the coupled hydrodynamics, scalar field and Boltzmann equations.

In what follows, we test the dependence of the bubble wall speed on all the most relevant meta-parameters. Unless otherwise stated, we plot the results of these tests for one benchmark point in the Standard Model with singlet scalar, explained below in section 6.1. Our default benchmark point (BM1) has

$$m_s = 120 \text{ GeV}, \qquad \lambda_{hs} = 0.9, \qquad \lambda_s = 1, \qquad \text{(BM1)}$$

where m_s is the singlet mass, λ_{hs} the portal coupling and λ_s the self coupling. The specific implementation can be found in `Models/SingletStandardModel_Z2/singletStandardModelZ2.py`.

5.1. Testing convergence in the number of basis polynomials

Perhaps the most crucial meta-parameters are the integer basis sizes M, the spatial basis size or `spatialGridSize`, and N, the momentum basis size or `momentumGridSize`. These must be sufficiently
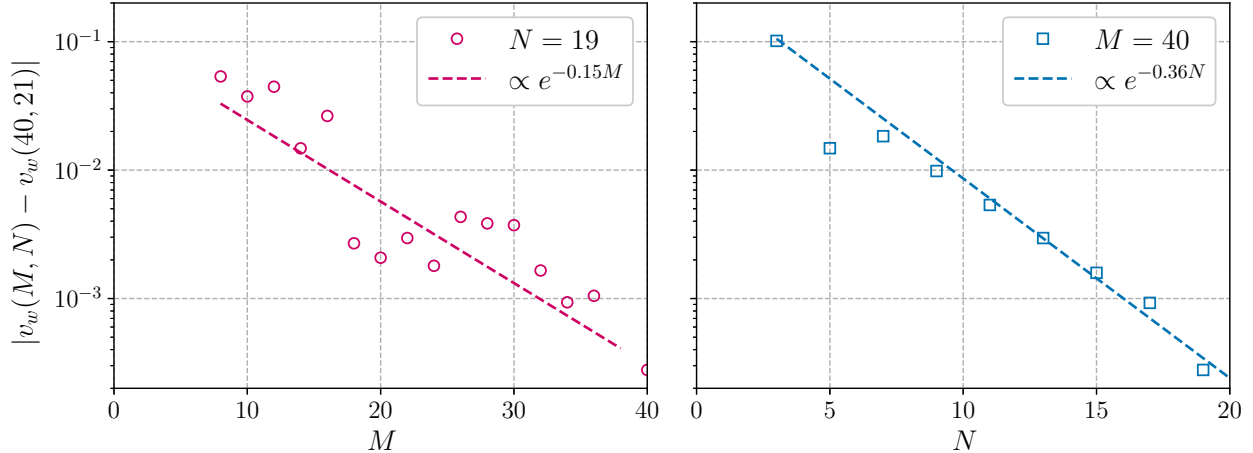
Figure 2: Convergence for the bubble wall speed, at our parameter point (BM1) in the xSM, as the numbers of basis polynomials in the spatial (M) and momentum (N) dimensions are increased. The rate of convergence is expected to be exponential for sufficiently large N and M. The dot-dashed line highlights the result for largest N and M. All points with $N > 20$ were computed on a cluster.

large to resolve the length and momentum scales present in the system. For sufficiently large M and N, the spectral method that we adopt leads to exponential convergence to the continuum limit.

We estimate truncation error of the spectral decomposition by the magnitude of the last coefficient in the Chebyshev basis. This follows Boyd's Rule of Thumb [100], and should give the correct order of magnitude as long as the convergence in M and N is exponential rather than power-like. For one benchmark point in the xSM, the deviation of v_w as a function of N and M is shown in figure 2. At this benchmark point, the errors due to finite M and N are reduced to a few percent for $M \gtrsim 20$ and $N \gtrsim 10$. Fits of the form $\delta v_w(M) = a\, e^{-bM}$ and $\delta v_w(N) = c\, e^{-dN}$ yield reasonably good agreement with $b \approx 0.15$ and $d \approx 0.36$ respectively, as can be seen in figure 3. However, these values are not universal.

For additional insight into the approach to the continuum limit, in `WallGo` we also solve the Boltzmann equation using a finite difference method, in addition to the spectral method. The finite difference method is accurate up to $O(M^{-2})$ and $O(N^{-2})$ and hence its asymptotic convergence is much slower than the spectral method. It is therefore expected that, for sufficiently large M and N, differences between the two methods should be larger than the intrinsic truncation error of the spectral method. `WallGo` therefore throws a warning if this is not satisfied, and requests a larger basis set.

35

Figure 3: Exponential fits to the rate of convergence for the bubble wall speed, at our parameter point (BM1) in the xSM, as the numbers of basis polynomials in the spatial (M) and momentum (N) dimensions increase. In computing these data, the error tolerance for the wall speed calculation was set to 10^{-3}, establishing a floor for the exponential convergence of the data.

5.2.   Dependence on `meanFreePathScale`

An important parameter that the user has to select is `meanFreePathScale`. It enters in the `Grid` object via the definition of the mapping between the physical spatial coordinates z and the compact ones $\chi(z)$, cf. section 3.3. An optimal choice would be similar to the asymptotic decay length of the Boltzmann equation's solution when evaluated at a slow wall velocity. In other words, if the solution behaves like $\delta f(v_w \to 0, z \to \pm\infty) \sim \exp(\mp z/l)$, then the optimal choice for `meanFreePathScale` is l. With the chosen $z \to \chi(z)$ mapping and for the optimal choice of the `meenFreePathScale`, the solution expressed in the χ coordinates will be a straight line close to the boundaries at $\chi = \pm 1$. This ensures that, in the χ coordinates in which the Boltzmann equation is solved, the solution is as smooth as possible which makes it easier to resolve.

In principle, the asymptotic decay length l of the solution is completely determined by the collision operator $\mathcal{C}[f]$. Therefore, the optimal `meanFreePathScale` should only depend on the matrix elements. Furthermore, if only strong interactions are included in the matrix elements, a simple estimate gives us $l \sim \frac{1}{\alpha_\mathrm{S}^2 T_\mathrm{n}} \sim 70/T_\mathrm{n}$.[14]

A simple way to test whether the chosen value of `meanFreePathScale` is adequate is to plot the solution of the Boltzmann equation in the χ coordinates. This can be done from a `WallGoResults`

---

[14]If other interactions are included, α_S^2 can simply be replaced by the relevant couplings. For having the Standard Model Ws out of equilibrium, the relevant couplings would be $\alpha_w \alpha_\mathrm{S}$. This corresponds to slower decay than decay from the strong interaction, and is thus relevant to l.

Figure 4: Left: Value of the function $\Delta_{00}(\chi)$ for the top quark obtained by solving the Boltzmann equation with $M = 50$ and $v_w = 0.1$. Three different values of `meanFreePathScale` were used (shown in the legend). Right: Variation of the wall velocity obtained by varying `meanFreePathScale` with four different values of M.

object with

L.18

```
WallGoResults.Deltas.Delta00.coefficients[a]
```

which returns the values of the function $\Delta_{00}^a(\chi^{(\alpha)})$ on the $\chi^{(\alpha)}$ grid coordinates. The latter can be accessed via

L.19

```
WallGoResults.Deltas.Delta00.grid.chiValues
```

The left side of figure 4 shows an example of a solution for the xSM (BM1) where only the top quark was considered to be out-of-equilibrium, and only including the strong interaction in the matrix elements. Three solutions are shown, computed with three different values of `meanFreePathScale`. Note that all three curves are meant to represent the same physical solution on z; only the mapping $z \to \chi(z)$ is different.

From this figure, it is quite apparent that the value `meanFreePathScale` $= 500/T_{\rm n}$ is too large to efficiently represent the solution. The problem is that the solver samples points too far from the wall where the solution is nearly vanishing. This means that all the points in the intervals $\chi \in [-1, -0.6]$ and $[0.6, 1]$ are essentially wasted. It would therefore be possible to get a similar accuracy with a smaller value of M (and thus a faster calculation) with a better choice of `meanFreePathScale`. On the other hand, the value `meanFreePathScale` $= 5/T_{\rm n}$ is clearly too small, which causes numerical

Figure 5: Dependence of v_w on different values of `smoothing` (left) and `ratioPointsWall` (right) in the xSM. We used `ratioPointsWall` $= 0.5$ in the left plot, `smoothing` $= 0.1$ in the right plot, and `meanFreePathScale` $= 50/T_\mathrm{n}$ for both.

instabilities, appearing as large oscillations in the solution. Here, the solver does not explore the space far away from the wall, so it cannot resolve the solution's tails. This causes the solution's derivative to become infinite at $\chi = \pm 1$, ultimately creating these numerical instabilities. Finally, the value of `meanFreePathScale` $= 50/T_\mathrm{n}$ seems much more adequate (note that it is quite close to our previous estimate of $70/T_\mathrm{n}$) as it leads to a smooth solution free from wild oscillations. Furthermore, the solution only vanishes at $\chi = \pm 1$ so each point sampled by the solver is actually increasing the solution's accuracy.

To understand to what extent the computed wall velocity depends on `meanFreePathScale`, we vary it over three orders of magnitude resulting in less than a 2% change in v_w; see figure 4 (right) for (BM1). Even in cases of significant numerical instabilities—such as in the $5/T_\mathrm{n}$ curve of figure 4—the oscillations tend to cancel out, keeping the effect on v_w minimal. However, if these instabilities become excessive, the solution would eventually diverge completely. Therefore, we strongly recommend avoiding these instabilities by choosing an appropriate `meanFreePathScale`. Additionally, as expected, increasing M reduces the variation in v_w since the larger number of points helps resolve the solution and makes up for a non-optimal value of `meanFreePathScale`. Also, all the curves roughly agree on the wall velocity when `meanFreePathScale` is between $20/T_\mathrm{n}$ and $100/T_\mathrm{n}$, indicating that the optimal value for this model should be in that interval.

5.3. Dependence on other `Grid` parameters

We investigate here the effect of the remaining `Grid` parameters, `smoothing` and `ratioPointsWall`, on the wall velocity in (BM1). The current `Grid3Scales` (which inherits from `Grid`) implementation used by `WallGo` divides the spatial direction into three distinct regions. The tails of the solution, which are located in the intervals $\xi \in (-\infty, -L_{\text{Grid}}]$ and $\xi \in [L_{\text{Grid}}, \infty)$ are mapped to the compact intervals $\chi \in [-1, -\text{ratioPointWall}]$ and $\chi \in [\text{ratioPointsWall}, 1]$, respectively ($L_{\text{Grid}}$ is the wall thickness in the `Grid` object). The density of points in the tails is set to decay exponentially when $\xi \to \pm\infty$, with a decay length set by `meanFreePathScale`. The density of points in the wall (which is mapped from $\xi \in [-L_{\text{Grid}}, L_{\text{Grid}}]$ to $\chi \in [-\text{ratioPointsWall}, \text{ratioPointsWall}]$) is set to be roughly constant. The `smoothing` parameter controls the transition from one region to the other. In the `smoothing` $\to 0$ limit, the mapping's first derivative becomes discontinuous at $\chi = \pm\text{ratioPointsWall}$. Increasing `smoothing` makes the transitions smoother between the three regions, which removes the discontinuity introduced by the mapping, but makes the distinction between the regions less clear.

The effect of these two parameters on the wall velocity is shown in figure 5. Varying `smoothing` by four orders of magnitude only changes the wall velocity by a maximum of 3%, which shows that the results are not very sensitive to this parameter. One can still observe that best convergence is attained with `smoothing` ≈ 0.1. However, we note that, when `meanFreePathScale` is much larger than the wall width, it can be beneficial to increase `smoothing` up to ~ 1 to smooth out the large variation of scales between the different regions. In the right panel of this figure, one can observe that the impact of `ratioPointsWall` on the wall velocity is also small, although best convergence is attained when at least half of the points are used to resolve the wall.

5.4. Dependence on `fieldValueVariationScale` and `temperatureVariationScale`

To numerically evaluate derivatives, `WallGo` needs an estimate of the relevant temperature and field scale. A poor choice of these parameters might lead to inaccuracies in the interpolated `FreeEnergy` object, which can result in problems while solving the wall velocity. The solver might e.g. mistakenly return a runaway solution, or not be able to find the local thermal equilibrium solution. Typically, `temperatureVariationScale` (TVS) can roughly be estimated by the difference of the critical and nucleation temperatures, and `fieldValueVariationScale` (FVVS) can be estimated by the VEVs of the field.

In figure 6 we demonstrate the dependence of v_w on these parameters for the Standard Model with a light Higgs (see section 6.2). For the five different Higgs masses we consider, the difference between the critical temperature and nucleation temperature is of the order of 0.2 GeV, and the VEV is of order 60 GeV. In the left panel, we keep the FVVS fixed at 50 GeV, and we vary the TVS from 0.5 GeV to 5.0 GeV. We see that the value of v_w depends only very weakly on the value of the TVS. For a TVS
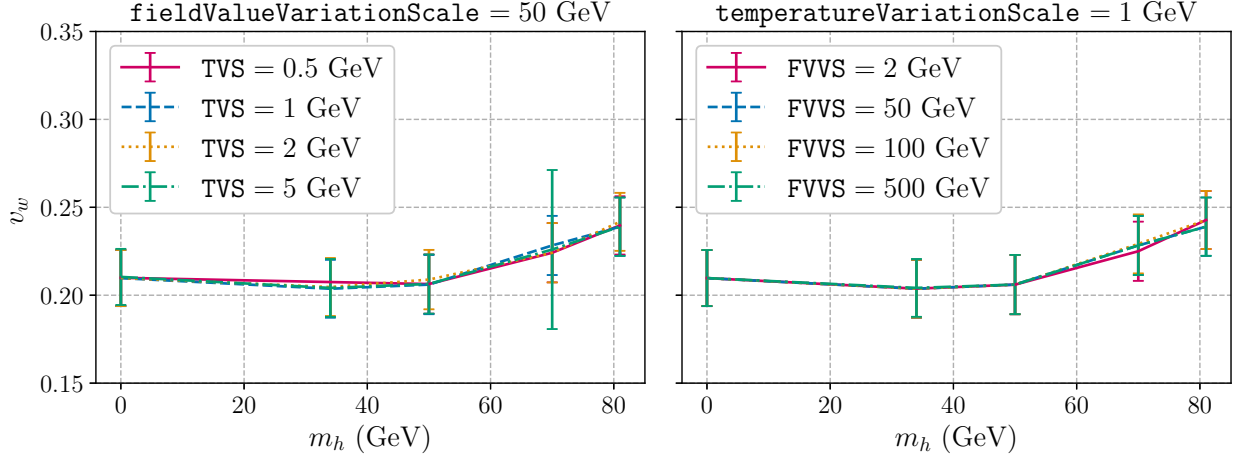
Figure 6: Dependence of v_w on different values of `temperatureVariationScale` (TVS, left) and `fieldValueVariationScale` (FVVS, right) for the Standard Model with a light Higgs (cf. section 6.2).

of 0.5 GeV however, the wall velocity is not recovered for $m_h = 34$ GeV.[15] For a `TVS` of 5.0 GeV, `WallGo` prints a warning:

```
Warning:  the temperature step size seems too large.
          Try decreasing temperatureVariationScale.
```

Moreover, the obtained v_w for $m_h = 70$ GeV has a large error bar. The latter is a result of `WallGo` not recovering the LTE value of v_w, which enters in the truncation error estimate. From this graph we thus conclude that the optimal choice of `TVS` is ~ 1 GeV, slightly larger than the difference between the critical and nucleation temperature for this model.

The right panel of figure 6 shows the dependence on the `fieldValueVariationScale`, for a fixed `TVS` of 1.0 GeV. We see that we can vary the scale over a much larger range than the `TVS`; the results between 2.0 and 500 GeV are almost identical, and the differences are much smaller than the truncation error. If we choose a smaller value for the `FVVS`, the phase tracer throws an error. We thus conclude that the VEV is an appropriate choice for the `FVVS`, but that the solution is not very sensitive to it.

---

[15]For this particular point, the correct wall velocity would be recovered by decreasing the tolerance of the phase tracer. We have however chosen to keep it fixed at 10^{-6} for the graph.

6. Examples beyond the most simple one

6.1. Standard model with singlet scalar

A simple model that renders the electroweak phase transition first-order and is still allowed by experimental data is the singlet scalar extension (xSM). It is obtained by augmenting the SM with a new scalar field s, which is not charged under the SM gauge group. To simplify the analysis, one can impose the singlet field to have an additional $\mathbb{Z}_2$ symmetry, in which case the effective potential reads

$$V^{\text{eff}}(\Phi, s, T) = \mu_h^2 \Phi^\dagger \Phi + \lambda_h (\Phi^\dagger \Phi)^2 + \frac{1}{2}\mu_s^2 s^2 + \frac{1}{4}\lambda_s s^4 + \frac{1}{2}\lambda_{hs}(\Phi^\dagger \Phi)s^2$$
$$+ V_{\text{CW}}(\Phi, s) + V_T(\Phi, s, T), \tag{6.1}$$

where V_{CW} is the Coleman-Weinberg potential, V_T the thermal potential, and Φ denotes the Higgs doublet. This model only depends on three new parameters: the singlet mass in the broken phase $m_s^2 = -\lambda_{hs}\mu_h^2/(2\lambda_h) + \mu_s^2$, its self-coupling λ_s, and its coupling with the Higgs λ_{hs}.

We follow here the methodology of [52] and choose a renormalisation scheme where V_{CW} does not change the scalar fields' VEVs and masses. We then have

$$V_{\text{CW}} = \sum_a \pm \frac{n_a}{64\pi^2}\left\{m_a^4(\Phi, s)\left[\log\left(\frac{m_a^2(\Phi, s)}{\bar{m}_a^2}\right) - \frac{3}{2}\right] + 2m_a^2(\Phi, s)\bar{m}_a^2\right\}, \tag{6.2}$$

where the upper sign is for bosons and lower one for fermions, the sum is over all the massive particles (which we take to be the top quark, the W and Z bosons, the Higgs, the singlet and the Goldstone bosons), n_a is their number of degrees of freedom, $m_a(\Phi, s)$ their field-dependent mass and $\bar{m}_a$ their mass in the broken phase at $T = 0$. Finally, the thermal potential is given by the one-loop contribution

$$V_T = \frac{T^4}{2\pi^2}\sum_a \pm n_a \int_0^\infty dy\, y^2 \log\left[1 \mp \exp\left(-\sqrt{y^2 + m_a^2(H, s)/T^2}\right)\right] - \frac{\tilde{g}\pi^2 T^4}{90}, \tag{6.3}$$

where the second term is the thermal contribution from the massless degrees of freedom, with $\tilde{g} = 83.25$.

To study the stability of `WallGo` against a wide range of models, we performed a parameter scan of the xSM. We used the same data points and nucleation temperatures that were computed in [52], which were obtained with $\lambda_s = 1$, $m_s \in [62.5, 160]$ GeV and $\lambda_{hs} \in [0.5, 1.4]$. To compute the wall velocity, we only considered the top quark to be out-of-equilibrium.

We show the result of this scan in figure 7, which shows the wall velocity computed by `WallGoManager.solveWall()` on the left side (deflagration and hybrid solutions) and `WallGoManger.solveWallDetonation()` on the right side (detonation solutions). In both cases, the corresponding solution did not necessarily exist for every model. If the net pressure on the wall is always negative in the relevant velocity interval (shown in red in the figure), the friction with the plasma would not be strong enough to stop the wall from accelerating and the wall velocity would end up exceeding v_J for deflagration/hybrid solutions and becoming ultrarelativistic for detonations. In detonation solutions, there is also a
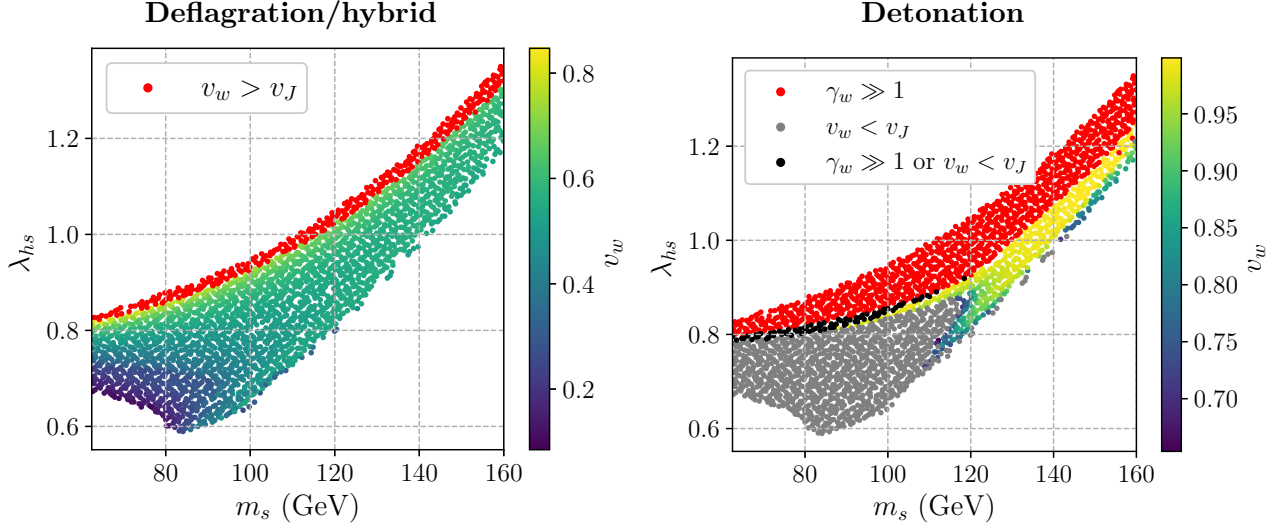
Figure 7: Scan of the xSM parameter space where the singlet mass m_s and the coupling λ_{hs} are varied with constant $\lambda_s = 1$. The left plot shows deflagration and hybrid solutions and the right plot shows detonations. The red, black and grey points do not have any solution of the corresponding type. The legend shows what their wall velocity will be based on the sign of the pressure on the wall.

possibility of having a positive pressure everywhere[16] (shown in grey) which indicates that the friction is too strong to allow solutions with $v_w > v_J$ and that these models can only have deflagration or hybrid solutions. Detonations can also have models with positive pressure at $v_w = v_J$ and negative pressure at $v_w = 1$ with no stable solution (shown in black). A pure static analysis would seem to indicate that these models would become deflagration/hybrid solutions since they would not be able to accelerate over the positive pressure at $v_w = v_J$, but time-dependent effects could allow them to overcome that positive pressure barrier and end up in the region with negative pressure, in which case it would accelerate to an ultrarelativistic speed. It is unfortunately not possible to conclude with certainty which one of these outcomes would turn out to be true with the static analysis made by `WallGo`.

It can also be seen that certain models exhibit two solutions: one deflagration or hybrid and one detonation solution. In fact, every model with a detonation solution also has a deflagration or hybrid one. Again, `WallGo` is not able to determine which one would be realised in nature because it relies on a static analysis. On one hand, one could argue that because the wall starts still and then accelerates, it would be stopped at the first static solution it encounters, which would be the deflagration/hybrid. On the other hand, these subjouguet solutions sometimes rely on the presence of a shock wave to increase the friction on the wall. If the shock wave does not have enough time to form before reaching the

---

[16]This cannot happen for deflagration solutions because the pressure is always negative at $v_w = 0$, otherwise tunneling would be impossible and the phase transition would not happen.

corresponding velocity, the wall could overshoot this first solution and reach the detonation solution (see [98], but note that no out-of-equilibrium effects are included in this study).

Finally, note that, although it is in principle possible to have several detonation solutions for the same model, we found no instance of that happening in this scan.

6.2. Standard Model with light Higgs

In [45, 46, 108], the wall velocity was computed for the Standard Model with a light Higgs mass. The computations all rely on a moment expansion of δf, and use three moments. We will compare our results with those obtained by Moore and Prokopec in [45] and Konstandin, Nardini and Rues in [46], and from now we will refer to these results as MP and KNR respectively. For a sufficiently light Higgs, the electroweak phase transition becomes first order without addition of new particles. The wall velocity becomes a function of the Higgs mass only. Lattice computations show that the phase transition becomes a cross-over for $m_h \gtrsim 72$ GeV [16, 18, 109, 110], but in the perturbative description of [45, 46, 108], the phase transition remains (weakly) first order even for larger Higgs masses as well, and the maximum value we consider is $m_h = 80$ GeV.

The temperature-dependent effective potential used by MP (and also [108]) is given in eq. (7) of [45][17]

$$V^{\text{eff}}(h, T) = D(T^2 - T_0^2)h^2 - CT^2 h^2 \log\left(\frac{h}{T}\right) - ETh^3 + \frac{\lambda_T}{4}h^4, \tag{6.4}$$

where h denotes the (background) Higgs field. Note that the term proportional to C is absent in the potential of KNR. The coefficients are given by (note that MP's expression for E contains a typo)

$$\lambda_T = \frac{m_h^2}{2v_0^2} - \frac{3}{16\pi^2 v_0^4}\left(2m_w^4 \log\left(\frac{m_w^2}{a_b T^2}\right) + m_z^4 \log\left(\frac{m_z^2}{a_b T^2}\right) - 4m_t^4 \log\left(\frac{m_t^2}{a_f T^2}\right)\right), \tag{6.5}$$

$$D = \frac{1}{8v_0^2}\left(2m_w^2 + m_z^2 + 2m_t^2\right), \tag{6.6}$$

$$C = \frac{1}{16\pi^2}(1.42g_w^4 + 4.8g_w^2\lambda_T - 6\lambda_T^2), \tag{6.7}$$

$$E = \frac{1}{12\pi}\left(4\frac{m_w^3}{v_0^3} + 2\frac{m_z^3}{v_0^3} + (3 + 3^{3/2})\lambda_T^{3/2}\right), \tag{6.8}$$

$$B = \frac{3}{64\pi^2 v_0^4}\left(2m_w^4 + m_z^4 - 4m_t^4\right), \tag{6.9}$$

$$T_0 = \sqrt{\frac{1}{4D}\left(m_h^2 - 8Bv_0^2\right)}. \tag{6.10}$$

The term proportional to $\lambda_T^{3/2}$ in E is absent in KNR's potential, and E gets multiplied by an overall

---

[17]Eq. (7) of MP contains one additional term, proportional to A_F. This term is set to zero in the benchmark points that we compare to. It is also set to zero by KNR.

Figure 8: Wall velocity as function of a light the Higgs mass, m_h, in the Standard model. The results are obtained by `WallGo` using the potentials of MP and KNR. The results from MP and KNR are displayed for comparison.

factor 3/2. The input parameters can be found in [108]

$$g_w = \frac{2M_w}{v_0}, \qquad m_w = 80.4\,\text{GeV}, \qquad m_z = 91.2\,\text{GeV}, \qquad m_t = 174.0\,\text{GeV},$$

$$v_0 = 246.0\,\text{GeV}, \qquad a_b = 49.78019250, \qquad a_f = 3.111262032. \tag{6.11}$$

We follow the procedure described by MP and KNR to find the nucleation temperature (this happens outside of `WallGo`, we make use of [81]) as a function of m_h. Note that MP and [108] use $S_3/T = 97$ as nucleation criterion, whereas KNR use $S_3/T = 140$.

We compute v_w with out-of-equilibrium contributions from tops and W- and Z-bosons (the latter two are treated as a single species, with the mass of the W-boson). We do not allow for out-of-equilibrium Higgs particles. This approximation is also made in MP and KNR. We obtain the matrix elements from `WallGoMatrix`, where we consider the same couplings as MP and KNR. We perform the computation at $N = 11$, $M = 20$ for the two potentials of MP and KNR. Our results and the results from MP, KNR (we take the results with Tanh ansatz from [45]) are summarised in figure 8.

Before comparing to our results, let us discuss the differences between the results of MP and KNR. The wall velocity reported by KNR (blue dashed line) is significantly smaller than the values obtained by MP (magenta solid line). KNR gives two reasons for the difference. First, the potentials and the nucleation criterion are not identical; we find that the value of the phase transition strength α_n of KNR is a factor $0.61 - 0.95$ smaller than the α_n of MP, which could partially explain the smaller velocity. Second, the treatment of the shock front is different: in MP the fluid profile in front of the wall is approximated by a linearisation of the fluid equations. KNR solve the full fluid equation (as does `WallGo`) and argues that they therefore correctly capture the backreaction effect from heating [53, 54, 97], whereas [45] do not.

44

The results obtained with `WallGo`, for $N = 11$, $M = 20$ are smaller than the ones obtained with the moment expansion, for both choices of potential (solid green for the potential of MP, pink dashed for the one of KNR). Part of the difference with MP could be explained by the different treatment of the hydrodynamics. In addition, MP and KNR use the same matrix elements, but some errors in those were pointed out in [101], and our matrix elements agree with [101]. We include mixing between the different out-of-equilibrium species in the Boltzmann equations, which MP and KNR do not. The method to extract the wall velocity is also slightly different: MP and KNR minimise moments of the equation of motion, whereas we minimise the action to obtain the wall parameters.

A last prominent difference between our approach and the MP and KNR approach is the different method used to solve the Boltzmann equations - we expand δf in Chebyshev polynomials rather than taking a moment expansion. For comparison, we demonstrate our result for $N = 5$, $M = 20$ for the MP potential in dotted light blue, and we see that our value for v_w becomes significantly larger (but still smaller than the result of MP). This begs the question whether the computation of v_w obtained with just two moments has really converged. In [51], the corrections from higher moments to the friction was computed. The distribution from particles out-of-equilibrium increased with the number of moments, suggesting that v_w decreases with an increasing number of moments. Unfortunately, the dependence of v_w on the number of moments was not computed. It is therefore yet an open question whether the two methods converge towards the same value of v_w, which we leave to future work.

For comparison, figure 8 also shows the local thermal equilibrium result for the potential of MP in dotted yellow. As expected, v_w is larger than with out-of-equilibrium effects included. Unlike for the xSM, where it was demonstrated in [52] that LTE often gave a reasonable estimate for the full v_w, the LTE result does not give a very good prediction of v_w, as the two values are different by about a factor 3.

6.3. Inert doublet model

The wall velocity was computed for three benchmark points of the Inert Doublet Model in [49], and we will now compare to the results obtained with `WallGo`. The corresponding model file can be found in `Models/InertDoubletModel/inertDoubletModel.py`. The Inert Doublet Model [111–113] is a special case of the Two Higgs Doublet Model, where the new doublet has the same SM gauge charges as the SM Higgs field. In addition, it has a global $\mathbb{Z}_2$-symmetry, which prevents tree-level couplings to the Standard Model fermions. The model can provide a dark matter candidate [113–115], and features a first-order phase transition in part of its parameter space [116–124].

To make a faithful comparison of v_w, we will follow closely the implementation of the effective potential used in [49]. Note that the computation of [124] suggests that an accurate computation might require the inclusion of higher-order corrections in the effective potential, but we leave these to future work. The scalar potential at zero-temperature is given by (note that our normalisation of $\lambda_{1,2}$

BM	$\bar{m}_H$ [GeV]	$\bar{m}_A, \bar{m}_{H^\pm}$ [GeV]	λ_L	T_c [GeV]	T_n [GeV]	v_w [49]	v_w [WallGo]
A	62.66	300	0.0015	118.3	117.1	0.165	0.191 ± 0.024
B	65.00	300	0.0015	118.6	117.5	0.164	0.180 ± 0.025
C	63.00	295	0.0015	119.4	118.4	0.164	0.182 ± 0.024

Table 1: Benchmark (BM) input parameters used in [49] and results for v_w in [49] and WallGo.

differs from [49])

$$V_0 = \mu_1^2 \Phi^\dagger \Phi + \mu_2^2 \chi^\dagger \chi + \lambda_1 (\Phi^\dagger \Phi)^2 + \lambda_2 (\chi^\dagger \chi)^2$$
$$+ \lambda_3 \Phi^\dagger \Phi \chi^\dagger \chi + \lambda_4 \Phi^\dagger \chi \chi^\dagger \Phi + \left[\frac{\lambda_5}{2} (\Phi^\dagger \chi)^2 + \text{H.c.} \right], \tag{6.12}$$

where Φ denotes the SM Higgs boson, and χ the inert doublet

$$\chi = \begin{pmatrix} H^+ \\ \frac{1}{\sqrt{2}}(H + iA) \end{pmatrix}. \tag{6.13}$$

We have four new degrees of freedom: the CP-even and CP-odd scalars H and A respectively, and the charged scalars $H^\pm$. Their zero-temperature masses correspond to

$$\bar{m}_H^2 = \mu_2^2 + \frac{1}{2}(\lambda_3 + \lambda_4 + \lambda_5)v^2, \tag{6.14}$$

$$\bar{m}_A^2 = \mu_2^2 + \frac{1}{2}(\lambda_3 + \lambda_4 - \lambda_5)v^2, \tag{6.15}$$

$$\bar{m}_{H^\pm}^2 = \mu_2^2 + \frac{1}{2}\lambda_3 v^2, \tag{6.16}$$

where v denotes the zero-temperature Higgs VEV. In the following, we will choose $\lambda_4 = \lambda_5$, such that $\bar{m}_A^2 = \bar{m}_{H^\pm}^2$. The combination appearing in the mass $\bar{m}_H^2$ is denoted as $\frac{1}{2}(\lambda_3 + \lambda_4 + \lambda_5) \equiv \lambda_L$.

We consider a one-step phase transition where the Higgs-field obtains a VEV, $\Phi = 1/\sqrt{2}(0, h)^T$. The effective potential for h is given by

$$V^{\text{eff}}(\phi, T) = V_0(\phi) + V_{\text{CW}}(h) + V_T(h, T). \tag{6.17}$$

For further details on the implementation, see [49] or the `inertDoubletModel.py` file. We have confirmed that we obtain the same critical temperature as [49] with our implemented potential. We take the value of the nucleation temperature from [49]. The parameters of the three benchmark points are given in table 1. For all benchmark points, $\lambda_2 = 0.1$. Note that the running of the couplings is not considered.

[49] considers the out-of-equilibrium contributions from the top quark, the W- and Z-boson and the A and $H^\pm$ scalars. The W- and Z-boson are treated as a single species (with the mass of the W), and similarly the $H^\pm$ and A are treated as a single species with the mass of the A. The

Boltzmann equations are solved with a moment expansion as in [45]. The wall is approximated with a tanh-ansatz. The hydrodynamics equations are solved without linearisation, but the fluid equation of motion is approximated by the pure radiation-contribution and temperature-independent vacuum energy difference only (bag equation of state).

We use `WallGo` to compute the wall velocity for the same out-of-equilibrium particles at a `momentumGridSize = 11` and `spatialGridSize = 30`. The matrix elements are determined with `WallGoMatrix`. We include the same external particles and couplings as in [49], (i.e. we set $\lambda_{1,2,4,5}, g_Y = 0$). We end up with a larger set of diagrams than [49], since certain diagrams were neglected there, for example interactions between A and W. As position-dependent VEVs are currently not supported in `WallGo`, we fix the VEV to be equal to half of $v_{\mathrm{LT}}(T_\mathrm{n})$ of benchmark point A.

The last two columns of table 1 compare the obtained values of v_w. In all cases, the result of `WallGo` is slightly larger than the result of [49], but overall there is a reasonable agreement, with deviations smaller than 20%. The observed differences are likely caused by the different approaches used in the Boltzmann equations (taking moments versus the spectral method), the slightly different treatment of the hydrodynamics of the plasma, and the differences in the matrix elements and corresponding collision terms. As we included a larger set of matrix elements, this could also explain why we observe a larger value of v_w, as the particles can get slightly closer to their equilibrium distribution, reducing the corresponding friction.

7. Conclusions

We present `WallGo`, a `Python` package for calculating the wall velocity, v_w, in first-order cosmological phase transitions. `WallGo` includes two auxiliary packages, `WallGoMatrix` and `WallGoCollision`. Together, they perform the entire computation from calculating the matrix elements for in- and out-of-equilibrium particles, to computing the collision integrals for the Boltzmann equations, to solving the coupled hydrodynamics, scalar-field equation of motion and Boltzmann equations for the out-of-equilibrium particles. The model, along with the set of out-of-equilibrium particles, is fully user-defined. Since the wall velocity significantly affects the predictions for GW spectra and the baryon or dark matter abundance, `WallGo` has the potential to enhance future predictions of observables related to cosmological first-order phase transitions.

In this article, we have presented a quick start guide, theoretical background and a succinct documentation of the code. A more extensive documentation of the code can be found in `https://wallgo.readthedocs.io`. The online documentation will also stay up-to-date with future versions of `WallGo`. By varying the number of basis polynomials we demonstrate that the spectral expansion indeed converges exponentially in section 5. In addition, we show that the wall velocity has only a mild sensitivity to the other configuration parameters. We also provide examples of tests that the user can perform to determine whether their obtained value of v_w has converged. In section 6, we demonstrate computations of v_w for several BSM models. In section 6.1, we use `WallGo` to study the

xSM, demonstrating that `WallGo` can be used to scan over a parameter space. The results agree with those obtained earlier in [52].

For the Standard Model with a light Higgs in section 6.2, and the Inert Doublet Model (IDM) in section 6.3, we compare our results to the results presented in [45, 46] and [49] respectively. In the case of the Standard Model, we observe that the obtained wall velocities differ significantly from the values obtained in [45, 46] with our results being almost a factor 2 smaller. For the IDM, we only deviate from the results of [49] by less than 20%.

A number of factors could explain the significant differences with [45, 46] and the milder differences with [49]: all three references expand the δf_a in three moments, which is known to yield a singular solution at c_s [50, 51], due to the linearisation of the background solution. We use the spectral expansion, which allows us to converge exponentially quickly towards the exact solution for δf_a. Moreover, we do not need to linearise the background solution. Unlike [45, 46, 49], we have included mixing in the collision terms in the Boltzmann equations. There are also some errors in the matrix elements used in [45, 46], pointed out by [101]. We also use a slightly larger set of matrix elements than [49]. Lastly, there are differences with [45] in the treatment of the hydrodynamics outside of the bubble wall. As pointed out by [46], that treatment incorrectly captures the backreaction force from heating. Finding the main source of the discrepancy will require a more in-depth comparison, which will be the topic of future work.

For future versions of `WallGo`, we foresee several improvements, such as dropping the Tanh-ansatz, including matrix elements and collision integrals beyond leading logarithmic order, allowing for field-dependent couplings and masses, and the implementation of a consistent treatment of soft gauge modes. These improvements involve both technical and theoretical challenges.

Having automated the full computation of the wall velocity, from matrix elements to Boltzmann equations, `WallGo` has the potential to greatly improve predictions of FOPT observables. Rather than being the final verdict in the computation of v_w, `WallGo` opens a new chapter of precision computations for FOPTs. The automation of the computation will allow us to investigate and address the main sources of uncertainty which will be a topic of upcoming work after `pip install`ing `WallGo`.

Acknowledgements

A.   Using the `WallGoMatrix` matrix-element generator

In this section, we detail the usage of matrix-element generator `WallGoMatrix`. Calculations of out-of-equilibrium contributions require a multitude of matrix elements. These can either be supplied to `WallGo` directly, or alternatively generated automatically with `WallGoMatrix` which is available via the repositories:
`https://github.com/Wall-Go/WallGoMatrix`,
`https://resources.wolframcloud.com/PacletRepository/resources/WallGo/WallGoMatrix`.

A.1.   Loading the required packages

The main routines for matrix element generation are collected in the files: `WallGoMatrix.m` and `matrixElements.m`. The model generation of `WallGoMatrix` utilises `DRalgo` [1] and `GroupMath` [2]. We ask the user to cite these references when making use of `WallGoMatrix`. Before loading the package it either needs to be installed as in listing L.4 or by placing the `WallGoMatrix` directory inside the `Wolfram` user base directory

```
<UserBaseDirectory>/Applications/WallGoMatrix
```

where the latter can be found under `$UserBaseDirectory` in `Mathematica`. Thereafter, `WallGoMatrix` is loaded via

```
WallGo`WallGoMatrix`$InstallGroupMath=True;
<<WallGo`WallGoMatrix`
```

where the flag `WallGo`WallGoMatrix`$InstallGroupMath` automatically installs `GroupMath` in the user base directory.

A.2. Defining the model

We first need to define the model, in the same way as in `DRalgo`. For the Standard Model without hypercharge group[18] this is done by writing

```
Group={"SU3","SU2"};
CouplingName={gs,gw};
```

The user needs to specify all particles that should be included in the computation of v_w; cf. discussion in sections 2 and 4. Just as in `DRalgo`, this is done by giving the Dynkin coefficients for each representation. These coefficients allow for handling a general model, but they might be unfamiliar. `GroupMath` provides a map to a more familiar notation; for SU(3) one can obtain all representations up to size 30 [2]

```
su3Reps = RepsUpToDimN[SU3,30];
Grid[Prepend[{#,RepName[SU3,#]}&/@ su3Reps,{"Dynkin coefficients","Name"}],
    Frame->All,FrameStyle->LightGray]
```

where a typical representation is the adjoint one with Dynkin index `{1,1}` and representation $R = \mathbf{8}$. As a consequence, the (adjoint) gauge representation of SU(3) × SU(2) is

```
RepAdjoint={{1,1},{2}};
```

where the notation is `RepAdjoint` $= \{\{\text{SU3}_{\text{dynkin}}\}, \{\text{SU2}_{\text{dynkin}}\}\}$. For fermions and scalars the same notation as for the vector particles is used.

---

[18]We follow the common practice in the wall velocity literature, of ignoring the U(1) contributions. However, this assumption could also be dropped.

As an example, we focus on a simplified fermion sector. One generation of quarks consists of one left-handed SU2-doublet quark and two right-handed SU2-singlet quarks. These are specified as

```
Rep1={{{1,0},{1}},"L"};
Rep2={{{1,0},{0}},"R"};
Rep3={{{1,0},{0}},"R"};
RepFermion1Gen={Rep1,Rep2,Rep3};
```

All three quark generations can now be created by stacking together three copies of RepFermion1Gen,

```
RepFermion3Gen={RepFermion1Gen,RepFermion1Gen,RepFermion1Gen}//Flatten[#,1]&;
```

The Higgs field is similarly specified by

```
HiggsDoublet={{{0,0},{1}},"C"};
RepScalar={HiggsDoublet};
```

The last entry indicates whether the scalar representation is complex ("C") or real ("R"). In the example of the SU(2) triplet, choosing a real ("R") or complex ("C") scalar representation would result for the scalar to have three or six degrees of freedom, respectively.

WallGoMatrix takes ordered lists of representations as an input. Generically a fermionic or scalar Rep will take the form

```
Rep={Rep[r₁],Rep[r₂],Rep[r₃],...,Rep[rₙ]};
```

The ordering of the indices $\{r_1, \ldots, r_n\} \in \overline{s} \vee \overline{f} \vee \overline{v}$ in which representations are stacked for scalars ($\overline{s} = \{s_1, \ldots, s_m\}$), fermions ($\overline{f} = \{f_1, \ldots, f_n\}$), and vectors ($\overline{v} = \{v_1, \ldots, v_o\}$) respectively, will be fixed throughout the computation and generation of the matrix elements in the following section; see e.g. section A.4. In our example of three quark families this corresponds to

```
RepFermion3Gen={Rep1,Rep2,Rep3,Rep1,Rep2,Rep3,Rep1,Rep2,Rep3};
```

Once all particles have been defined, the model can be loaded by writing

```
{gvvv,gvff,gvss,λ1,λ3,λ4,μij,μIJ,μIJC,Ysff,YsffC}=
    AllocateTensors[Group,RepAdjoint,CouplingName,RepFermion3Gen,RepScalar];
```

To distinguish fermions by their masses one has to define Yukawa couplings. An example is the

left-handed quark representation that splits into one left-handed top and bottom quark, respectively. The corresponding Yukawa couplings are definable as

```
InputInv={{1,1,2},{False,False,True}};
YukawaDoublet=CreateInvariantYukawa[Group,RepScalar,RepFermion3Gen,InputInv]//Simplify;
Ysff=-GradYukawa[yt*YukawaDoublet[[1]]];
```

The corresponding input invariant is of the form

```
InputInv={{s1,f1,f2},{s1c,f1c,f2c}};
```

where `s1` $\in \overline{\mathbf{s}}$ and `f1, f2` $\in \overline{\mathbf{f}}$ are indices that refer to the definition of scalar or fermionic representation in `RepScalar` and `RepFermion3Gen`, respectively. Here, `{s1,f1,f2} = {1,1,2}` corresponds to

$$\text{Higgs doublet} \times \text{Left-handed top-quark} \times \text{Right-handed top-quark} \sim \overline{Q}_{\mathrm{L}} \Phi^{\dagger} t_{\mathrm{R}} \, , \qquad (\text{A.1})$$

and the attribute `{s1c,f1c,f2c} = {False,False,True}` indicates which of the representations are unconjugated (`True`) and conjugated (`False`). By instead setting `f2 = 3`, the right-handed top quark is swapped for the right-handed bottom quark. For the definition of scalar couplings see [1].

Finally, the model is initiated via

```
ImportModel[Group,gvvv,gvff,gvss,λ1,λ3,λ4,μij,μIJ,μIJC,Ysff,YsffC];
```

A.3. Implementing spontaneous symmetry-breaking

After symmetry-breaking, interaction eigenstates and mass eigenstates are generally not the same. While one can still use the interaction eigenstates from the particle definitions of the previous section, one can also use the mass eigenstates directly. To this end, `WallGoMatrix` groups each degree of freedom of a particle depending on the corresponding mass. We now discuss how to make use of this feature.

To group particles depending on the symmetry-breaking induced masses, first a vacuum-expectation value must be defined. In the case of the doublet in the fundamental representation of the SU(2) of listing L.27, this amounts to

```
vev={0,v,0,0};
```

The `vev` here has size `Length[gvss[[1]]]`, where `gvss` is the `DRalgo` internal vector-scalar trilinear coupling tensor. The components belonging to each representation can be found by

```
PrintFieldRepPositions["Vector"]
PrintFieldRepPositions["Fermion"]
PrintFieldRepPositions["Scalar"]
```

which gives `{1;;4}` for a fundamental SU(2) doublet. For a complex representation (like ours), the first two indices specify the real-components, and the last two the complex components. Schematically

$$\Phi = \begin{pmatrix} \texttt{vev1} + i\texttt{vev3} \\ \texttt{vev2} + i\texttt{vev4} \end{pmatrix}, \tag{A.2}$$

or correspondingly `vev={vev1,vev2,vev3,vev4}`. By writing `vev={0,v,0,0}`, we specify that the Higgs VEV is of the form $\Phi = (0, \texttt{v})^T$.

After specifying the `vev`, the representations can be separated via their masses by writing

```
SymmetryBreaking[vev];
```

and subsequently referenced by the `CreateParticle` command. The information from the output of `SymmetryBreaking` can be used to deal field-dependent masses to the particles during the matrix element generation in section A.5. However, in the context of the `WallGoMatrix` examples, only thermal masses are used and therefore calling `SymmetryBreaking` does not affect the matrix elements.

The command becomes effective, when turning on VEV-dependent couplings. By default, VEV-dependence is turned off but can be invoked by setting the following option inside `SymmetryBreaking`

```
VevDependentCouplings->True
```

A.4. Specifying out-of-equilibrium particles

We now specify representations of particles which, during collision integration, can be taken in- or out-of-equilibrium. For these particles, a complete set of matrix elements with them appearing on the external legs will be provided.

As an example, we create particles for the left-handed top quark, the right-handed top-quark, the gluons, the W-bosons, and the scalar doublet, by creating a separate distribution for each of these. The to-be equilibrium-particles can be grouped in individual distributions or, if they are considered to be of the same light mass, into a single collective distribution. One such example is grouping together all quarks except for the top. If also leptons would be included, it is sensible to create a separate distribution for all left-handed leptons and all right-handed leptons.

By calling `SymmetryBreaking[vev]`, a list of representations and their corresponding masses is generated only for representations with at least one massive degree of freedom

```
SymmetryBreaking[vev];
Gauge rep 2 splits into particles with mass squared:
{2,1}:   (gw²v²)/4
Fermion rep 1 splits into particles with mass squared:
{1,1}:   (v yt)/√2
{1,2}:   0
Fermion rep 2 splits into particles with mass squared:
{2,1}:   (v yt)/√2
```

where, for the quarks, we chose to put the left-handed quark doublet on position 1 of `RepFermion3Gen` and the right-handed top quark on position 2; cf. listing L.25. In the above listing, the bracketed integer indices, `{r,m}`, indicate the particle with mass $M_{\mathtt{r}}[\mathtt{m}]$ of the corresponding representation $\mathtt{r} \in \{\overline{\mathtt{s}}, \overline{\mathtt{f}}, \overline{\mathtt{v}}\}$. An example is $M_{\mathtt{r}=1}[\mathtt{m}=1] = m_{\mathtt{ReptL}} = (\mathtt{v}\ \mathtt{yt})/\sqrt{2}$. An entire representation, without distinguishing mass states, would correspond to writing `{r}`.

Using this notation, several particles can be grouped together into one distribution using the explicit list of indices from listing L.35

```
{{i₁,...,iₗ},"R",M,"particleName"};
```

where $\mathtt{i}_i \in \mathtt{PrintFieldRepPositions}[\text{"Field"}]$, with $\mathtt{Field} \in \{\mathtt{Vector}, \mathtt{Fermion}, \mathtt{Scalar}\}$. Alternatively, after having called `SymmetryBreaking`, one can also use the command

```
CreateParticle[{{r₁,m₁},...,{rₙ,mₙ},"R",M,"particleName"];
```

Above, n is the number of particles grouped into the distribution, $\mathtt{R} = \{\mathtt{F}, \mathtt{V}, \mathtt{S}\}$ is the corresponding particle species, namely Fermion, Vectors, and Scalars, M the particle mass squared, and `"particleName"` the particle name. Specifically, the distribution for the massive left-handed and right-handed quarks, namely the top quark, can be specified by writing

```
(*left-handed top-quark*)
ReptL=CreateParticle[{{1,1}},"F",mt2,"TopLeft"];
(*right-handed top-quark*)
ReptR=CreateParticle[{{2,1}},"F",mt2,"TopRight"]
```

where `mt2` is the squared top-quark mass. Distributions for the gauge-bosons and scalars can similarly be created by writing

```
(*Vector bosons*)
RepGluon=CreateParticle[{1},"V",mg2,"Gluon"];
```

```
RepW=CreateParticle[{2},"V",mW2,"W"];

(*Higgs*)
RepH=CreateParticle[{1},"S",mH2,"Higgs"];
```

As seen above, in the definition of `CreateParticle`, several particles can be grouped together and dealt with in the same distribution.

To generate the matrix elements in section A.5, all particles of the theory defined in section A.2 need to be declared as with `CreateParticle[]`. Therefore, we group the remaining light quarks into a single representation

<div align="right">L.43</div>

```
(*Light quarks*)
LightQuarks=CreateParticle[{{1,2},3,4,5,6,7,8,9},"F",mq2,"LightQuarks"];
```

with mass squared `mq2`. By separately stacking together out-of-equilibrium representations via `ReptL`, `ReptR`, `RepGluon`, `RepW`, and `RepH` as well as equilibrium representations via `LightQuarks`, we create the two lists of particles that are relevant for collisions in the model, *viz.*

<div align="right">L.44</div>

```
ParticleList={ReptL,ReptR,RepGluon,RepW,RepH};
LightParticleList={LightQuarks};
```

The ordering of the particles can be arbitrary but this numeric ordering (starting from 0) will be subsequently used in all output of the matrix elements. This is the same `index` as in the collision model from section 4.2. Particles of the `LightParticleList` can never be ingoing in collisions and they will thus never appear on the first index of the generated matrix element file. The representations listed in `ParticleList` can still be assumed to be out-of- or in-equilibrium when using them during the collision integration in `WallGoCollision`. If a particle appears on the first index of the matrix element but it is in-equilibrium, this matrix element will simply be skipped in collision generation.

A.5. Generating the matrix elements

In calculations of the collision operators, we will encounter logarithmic divergences for small-angle scattering. These are softened by thermal masses. In this tutorial, the top quarks was given the mass `mt2`, the remaining fermions the mass `mq2`, the gluons the mass `mg2`, the W-bosons `mW2`, and the scalars `ms2`. The matrix elements are then calculated by writing

<div align="right">L.45</div>

```
MatrixElements=ExportMatrixElements[<filePath>,ParticleList,LightParticleList,OptionPattern];
```

where its default options are given as

```
OptionPattern={
    Replacements->{},
    NormalizeWithDOF->True,
    TruncateAtLeadingLog->True,
    Verbose->False,
    Format->"none"};
```

By default, the generated matrix elements are truncated at leading logarithmic order.

The indices in the elements M[a,c,d,e] of MatrixElements (cf. eq. (3.30)) correspond to the position indices of the respective particles inside the total list {ParticleList,LightParticleList}. Each element contains all (squared) matrix elements that contribute with those specific particles. The matrix-elements are also normalised by the degrees of freedom of the incoming particle, that is by the degrees of freedom of particle1. By modifying OptionPattern, it is possible to omit this normalisation factor by setting

```
NormalizeWithDOF->False
```

While there is no default output format of WallGoMatrix, the default input format of WallGo is .json; see section 4.1. The following formats are implemented:

```
Format->{"json","txt","hdf5"}
```

In some cases, one might generate certain matrix elements that one does not want to include in the collisions. For example, to include the lepton exchange diagram in W-scattering, the leptons have to be included in the (light) particle representations, and this will generate matrix elements with leptons on the second, third and fourth index. If we do not want to include these in the collision computation (e.g. because of computational costs), we simply do not include the corresponding particles in the WallGoCollision model. We then only need to provide an input for the lepton asymptotic thermal mass.

B. General matrix elements

WallGo and specifically WallGoMatrix, employ the leading-logarithmic approximation. To this end, we take a given (vacuum) matrix-element, and regulate terms that diverge logarithmically via adding the corresponding (thermal, or rather asymptotic, cf. section 3.4.1) mass. Since we only need vacuum matrix-elements, we can generate these efficiently. A useful way to do this for general models, is to use general coupling tensors (see [125–129]) to handle the group algebra; and spinor-helicity methods

to handle the Lorentz algebra.[19] This appendix illustrates the approach by focussing on three specific scattering processes.

Along these calculations, we first treat all particles as outgoing. All possible particle flows then follow upon changing the sign of the relevant momenta. Henceforth, we employ the notation (cf. [131])

$$s_{ij} = -(p_i + p_j)^2 \,, \tag{B.1}$$

and label particles as in section A as V for vector bosons, F for fermions, and S for scalars. The coupling tensors for the different interactions appearing in the calculations below are given by

$$S_i S_j V^a : \quad g_{ij}^a \,, \tag{B.2}$$

$$F_I F_J V^a : \quad g_{IJ}^a \,, \tag{B.3}$$

$$F_I F_J S_i : \quad Y_{iIJ} \,, \tag{B.4}$$

$$S_i S_j S_k : \quad \lambda_{ijk} \,, \tag{B.5}$$

where I, J, L, M describe spinor indices, a, b, c, d vector indices, and i, j, k, l scalar indices.

B.1. Matrix elements for $F_1 F_2 \to F_3 F_4$

We first focus on fermion scattering with a vector or scalar boson as propagator. The contributing diagrams to the s-, t-, and u-channels of the vector- and scalar-exchange are

$$-\mathcal{T}_{\lambda_1 \lambda_2 \lambda_3 \lambda_4} \supset \tag{B.6}$$

$$= \underbrace{g_J^{a\,I} g_L^{a\,M}}_{\equiv G_s} A_{\lambda_1\lambda_2\lambda_3\lambda_4}^s + \underbrace{g_L^{a\,I} g_M^{a\,J}}_{\equiv G_t} A_{\lambda_1\lambda_2\lambda_3\lambda_4}^t + \underbrace{g_M^{a\,I} g_L^{a\,J}}_{\equiv G_u} A_{\lambda_1\lambda_2\lambda_3\lambda_4}^u \,, \tag{B.7}$$

$$-\mathcal{T}_{\lambda_1 \lambda_2 \lambda_3 \lambda_4} \supset \tag{B.8}$$

where wiggly lines denote vector bosons, directed lines fermions, and dashed lines scalars. We now focus only on the u- and t-channels of the vector exchange. The contributing scalar-exchange diagrams of this scattering process are implemented in `WallGoMatrix`.

Here, the $\lambda_1, \ldots, \lambda_4$ specify the helicities of the particles. The coupling-tensor $g_L^{a\,I} = g_{13,L}^{a\,I}$ describes the interactions between two fermions and a vector boson, namely fermion 1 with spinor index I, fermion 3 with spinor index L, and the vector boson with label a. We treat all particles as Weyl;

---

[19]Using Feynman diagrams for expressions with chiral particles is not very practical and the spinor-helicity formalism has the added advantage that gauge-invariance is manifest [130].

which means that an upper spinor index always must contract with a lower one in the end—the hermitian conjugate flips upper and lower signs.

Furthermore, $A^t_{\lambda_1 \cdots \lambda_4}$ denotes the Lorentz-structure of the t-channel diagram, and $A^u_{\lambda_1 \cdots \lambda_4}$ of the u-channel one. Since $[p|\gamma^\mu|p] = \langle p|\gamma^\mu|p\rangle = 0$, the amplitudes are only non-zero if two helicities at a fermion vertex are of opposite sign, leaving six helicity signatures, namely

$$A^{t,u}_{++--}, \qquad A^{t,u}_{+-+-}, \qquad A^{t,u}_{+--+}, \qquad \left[A^{t,u}_{--++}, \qquad A^{t,u}_{-+-+}, \qquad A^{t,u}_{-++-} \right]. \tag{B.9}$$

Via complex-conjugation, the last three bracketed amplitudes can be related to the first three ones which reduces the total number of helicity signatures to three.

For the signature A_{+--+}, the u-channel diagram vanishes as two positive-helicity lines meet, hence we are left with the t-channel diagram. The signature A_{+-+-} only has a u-channel, and A_{++--} has both t and u-channel contributions. The contributions are,

$$A^t_{+--+} = + \langle 3|\gamma^\mu|1] \, [4|\gamma_\mu|2\rangle \, /s_{13} = -2\,[1\,4]\,\langle 2\,3\rangle \, /s_{13}\,, \tag{B.10}$$

$$A^u_{+-+-} = - \langle 4|\gamma^\mu|1] \, [3|\gamma_\mu|2\rangle \, /s_{14} = +2\,[1\,3]\,\langle 2\,4\rangle \, /s_{14}\,, \tag{B.11}$$

$$A^t_{++--} = + \langle 3|\gamma^\mu|1] \, [2|\gamma_\mu|4\rangle \, /s_{13} = +2\,[1\,2]\,\langle 3\,4\rangle \, /s_{13}\,, \tag{B.12}$$

$$A^u_{++--} = - \langle 4|\gamma^\mu|1] \, [2|\gamma_\mu|3\rangle \, /s_{14} = +2\,[1\,2]\,\langle 3\,4\rangle \, /s_{14}\,, \tag{B.13}$$

where in the second step of eq. (B.10), we used the Fierz identity $\langle 1|\gamma^\mu|2] \, \langle 3|\gamma_\mu|4] = 2\,\langle 1\,3\rangle\,[2\,4]$, $[k|\gamma^\mu|p\rangle = \langle p|\gamma^\mu|k]$, and the anti-symmetry $\langle p\,q\rangle = -\langle q\,p\rangle$. Using that $\langle n\,m\rangle^* = [m\,n]$ and $s_{mn} = \langle m\,n\rangle\,[n\,m]$, we find

$$\left| A^t_{+--+} \right|^2 = 4s_{14}^2/s_{13}^2\,, \qquad \left| A^u_{+-+-} \right|^2 = 4s_{13}^2/s_{14}^2\,, \qquad A^u_{++--}\left[A^t_{++--} \right]^\dagger = 4s_{12}^2/s_{13}/s_{14}\,,$$

$$\left| A^t_{++--} \right|^2 = 4s_{12}^2/s_{13}^2\,, \qquad \left| A^u_{++--} \right|^2 = 4s_{12}^2/s_{14}^2\,, \tag{B.14}$$

including the contributions from complex conjugation.

The remaining step is to multiply with the complex conjugate and sum over helicities, particles and anti-particles, such that

$$\sum |\mathcal{T}|^2 = 8(G_t G_t^\dagger)\frac{s_{14}^2 + s_{12}^2}{s_{13}^2} + 8(G_u G_u^\dagger)\frac{s_{13}^2 + s_{12}^2}{s_{14}^2} + 8(G_t G_u^\dagger + G_u G_t^\dagger)\frac{s_{12}^2}{s_{13}s_{14}}\,, \tag{B.15}$$

where the G_t and G_u are defined in eq. (B.7).

We can now get any amplitude of this type by adapting the signs of some particles as incoming, e.g.

$$F_1 F_2 \to F_3 F_4 : \qquad p_1, p_2 \to -p_1, -p_2\,, \qquad s_{13}, s_{14}, s_{12} \to t, u, s\,, \tag{B.16}$$

$$F_1 \bar{F}_2 \to F_3 \bar{F}_4 : \qquad p_1, p_4 \to -p_1, -p_4\,, \qquad s_{13}, s_{14}, s_{12} \to t, s, u\,. \tag{B.17}$$

B.2. Matrix elements for $S_1 S_2 \to F_1 F_2$

As before, we initially take all particles as outgoing. The momenta p_1, p_2 correspond to the scalars S_1, S_2, and p_3, p_4 to the fermions F_1, F_2. The corresponding diagrams constitute a s-, t-, and u-channel, viz.

$$-\mathcal{T}_{00\lambda_3\lambda_4} = \quad \cdots \quad + \quad \cdots \quad + \quad \cdots \quad + \quad \cdots \tag{B.18}$$

$$= \lambda_{ijk} Y_M^{k\,L} \mathcal{A}_{00\lambda_3\lambda_4}^s + g_{ij}^a g_M^{a\,L} A_{00\lambda_3\lambda_4}^s + Y_i^{IM} Y_{jIL} A_{00\lambda_3\lambda_4}^t + Y_i^{IM} Y_{jIL} A_{00\lambda_3\lambda_4}^u \,, \tag{B.19}$$

where Y_{iIJ} are the Yukawa coupling tensors and λ_{ijk} the cubic scalar coupling tensor. We henceforth only focus on the vector- and fermion-exchange contributions from eq. (B.19). Since the helicities of the fermions have to be opposing, we consider $\mathcal{T}_{00+-}$. We can write down the corresponding channels using $\langle p|\mathbb{P}\,|q] = \langle p\,P\rangle\,[P\,q]$ and $p_n\,|n] = 0$, such that

$$A_{00+-}^s = -\frac{\langle 4|1-2|3]}{s_{12}} = +2\frac{\langle 1\,4\rangle\,[1\,3]}{s_{12}}\,,$$

$$A_{00+-}^t = -\frac{\langle 4|1-3|3]}{s_{13}} = +\frac{\langle 1\,4\rangle\,[1\,3]}{s_{13}}\,,$$

$$A_{00+-}^u = +\frac{\langle 4|1-4|3]}{s_{14}} = -\frac{\langle 1\,4\rangle\,[1\,3]}{s_{14}}\,. \tag{B.20}$$

For simplicity, we only display quadratic contributions at leading logarithmic order and suppress mixed contributions. The sum of the (possibly) logarithmically divergent matrix elements is

$$\left|\mathcal{T}_{00+-}\right|^2 \supset 4\mathrm{Tr}\left[\left(g_{ij}^a g_M^{a\,L}\right)^2\right]\frac{s_{13}s_{14}}{s_{12}^2} + \mathrm{Tr}\left[\left(Y_i^{IM}Y_{jIL}\right)^2\right]\frac{s_{13}s_{14}}{s_{13}^2} + \mathrm{Tr}\left[\left(Y_i^{IM}Y_{jIL}\right)^2\right]\frac{s_{13}s_{14}}{s_{14}^2}\,. \tag{B.21}$$

The $\mathcal{T}_{00-+}$ process is the same as the above.

B.3. Matrix elements for $F_1 F_2 \to SV$

The corresponding diagrams that contribute to this process are

$$-\mathcal{T}_{\lambda_1\lambda_2 0\lambda_4} = \quad \cdots \quad + \quad \cdots \quad + \quad \cdots \tag{B.22}$$

$$= Y_{iJ}^I g_{ik}^d A_{\lambda_1\lambda_2 0\lambda_4}^s + Y_k^{MI} g_{MJ}^d A_{\lambda_1\lambda_2 0\lambda_4}^t + Y_k^{MJ} g_{MI}^d A_{\lambda_1\lambda_2 0\lambda_4}^u \,. \tag{B.23}$$

The only non-zero contribution to this process originates from fermions with identical helicity opposed to the helicity of the vector spin-1 particle. By first considering $\mathcal{T}_{++0-}$, we list the corresponding

channel contributions

$$A^t_{++0-} = \frac{\sqrt{2}}{[3\,4]} \frac{[2\,3]\,\langle 4|3|1]}{s_{13}}\,, \tag{B.24}$$

$$A^u_{++0-} = \frac{\sqrt{2}}{[4\,3]} \frac{[2\,4]\,\langle 3|4|1]}{s_{14}}\,, \tag{B.25}$$

where we used the external line rule for outgoing spin-1 massless vectors by inserting their polarization vectors in the spinor helicity notation [131], namely $\not{\epsilon}_-(p;q) = \frac{\sqrt{2}}{[q\,p]}\,(|p\rangle\,[q| + |q]\,\langle p|)$ and $\not{\epsilon}_+(p;q) = \frac{\sqrt{2}}{\langle q\,p\rangle}\,(|p]\,\langle q| + |q\rangle\,[p|)$. The matrix element squared gives the following terms

$$\begin{aligned}
|\mathcal{T}_{++0-}|^2 &= 2\mathrm{Tr}\left[\left(Y_k^{MI}g_{MJ}^d\right)^2\right]\frac{s_{14}s_{13}}{s_{13}^2} + 2\mathrm{Tr}\left[\left(Y_k^{MJ}g_{MI}^d\right)^2\right]\frac{s_{13}s_{14}}{s_{14}^2} \\
&\quad - 2\mathrm{Tr}\left[\left(Y_k^{MI}g_{MJ}^d\right)\left(Y_k^{MJ}g_{MI}^d\right)^\dagger + \left(Y_k^{MJ}g_{MI}^d\right)\left(Y_k^{MI}g_{MJ}^d\right)^\dagger\right]\,,
\end{aligned} \tag{B.26}$$

where the last term contains the contributions from mixed u- and t-channels.

C. Reduction of the collision integral

The integral (3.30) is nine dimensional, but four of the integrals are trivial due to the momentum-conserving δ-function. This leaves 5 integrals for numerical evaluation. Here we discuss a convenient parametrisation for use with numerical integrators.[20] For generality, we will keep particle masses explicit, so that $E_i = \sqrt{p_i^2 + m_i^2}$. The momentum P_1 is a known input, fixed e.g. by giving the $\rho_z, \rho_{||}$ momentum components on the grid.

We begin by doing the integral over p_4. For this, write

$$\int \frac{\mathrm{d}^3 p_4}{2E_4}(\cdots) = \int \mathrm{d}^4 P_4\,\delta(P_4^2 - m_4^2)\theta(E_4)(\cdots)\,, \tag{C.1}$$

and to make use of the newly introduced δ we define a function $g(p_3) \equiv P_4^2 - m_4^2$. Then

$$\begin{aligned}
\mathcal{C}^{\mathrm{lin}}_{ab}[\delta f^b] &= \frac{1}{4}\sum_{cde}\int \frac{\mathrm{d}^3 p_2\,\mathrm{d}^3 p_3\,\mathrm{d}^3 p_4}{(2\pi)^5 2E_2 2E_3 2E_4}\delta^4(P_1 + P_2 - P_3 - P_4)|M|^2\mathcal{P}^{\mathrm{lin}}[\delta f^b] \\
&= \frac{1}{4}\sum_{cde}\int \frac{\mathrm{d}^3 p_2\,\mathrm{d}^3 p_3}{(2\pi)^5 2E_2 2E_3}\delta(g(p_3))\theta(E_4)|M|^2\mathcal{P}^{\mathrm{lin}}[\delta f^b]\Big|_{P_4 = P_1 + P_2 - P_3}\,,
\end{aligned} \tag{C.2}$$

where $\mathcal{P}^{\mathrm{lin}}[\delta f^b]$ is given by

$$\mathcal{P}^{\mathrm{lin}}[\delta f^b] = f^a f^c f^d f^e \left(\delta_{ab}F_a^c + \delta_{cb}F_c^a - \delta_{db}F_d^e - \delta_{eb}F_e^d\right)\delta f^b\,, \tag{C.3}$$

---

[20]As discussed in e.g. [132, 133], it is possible to perform one more integral analytically, leaving 4 numerical integrals. However, this requires a nontrivial rotation of the coordinate system in our case, as we work in a frame where P_1 is not aligned along any of the x, y, z axes. We will thus not discuss the $9 \rightarrow 4$ reduction here.

and we have defined $F_b^a \equiv e^{E_a/T}/f_b^2$.

Next we go to spherical coordinates:

$$\mathbf{p}_2 = p_2(\sin\theta_2\cos\phi_2, \sin\theta_2\sin\phi_2, \cos\theta_2),$$
$$\mathbf{p}_3 = p_3(\sin\theta_3\cos\phi_3, \sin\theta_3\sin\phi_3, \cos\theta_3),\tag{C.4}$$

where $\theta_i \in [0, \pi]$ and $\phi_i \in [0, 2\pi]$. The integral over p_3 is done with the remaining δ function:

$$\mathcal{C}_{ab}^{\text{lin}}[\delta f^b] = \frac{1}{4}\sum_{cde}\int_0^\infty \frac{p_2^2 p_3^2 \mathrm{d}p_2 \mathrm{d}p_3}{(2\pi)^5 2E_2 2E_3}\int_0^{2\pi}\mathrm{d}\phi_2\mathrm{d}\phi_3\int_{-1}^1\mathrm{d}\cos\theta_2\,\mathrm{d}\cos\theta_3\,\delta(g(p_3))\theta(E_4)|M|^2\mathcal{P}^{\text{lin}}[\delta f^b]$$

$$= \frac{1}{4}\sum_{cde}\sum_i\int_0^\infty\frac{p_2^2 p_3^2\mathrm{d}p_2}{(2\pi)^5 2E_2 2E_3}\left|\frac{1}{g'(p_3^{(i)})}\right|$$

$$\times\int_0^{2\pi}\mathrm{d}\phi_2\mathrm{d}\phi_3\int_{-1}^1\mathrm{d}\cos\theta_2\,\mathrm{d}\cos\theta_3\,\theta(E_4)|M|^2\mathcal{P}^{\text{lin}}[\delta f^b]\Big|_{p_3=p_3^{(i)}}.\tag{C.5}$$

Here the i-sum is over non-negative roots $p_3^{(i)}$ of $g(p_3) = 0$ and p_3 has been fixed by the δ-function.

To find these roots, note that P_4 is already fixed in terms of the other momenta:

$$g(p_3) = P_4^2 - m_4^2 = (P_1 + P_2 - P_3)^2 - m_4^2$$

$$= Q + 2(P_1 \cdot P_2) + 2\mathbf{p}_3 \cdot (\mathbf{p}_1 + \mathbf{p}_2) - 2\sqrt{p_3^2 + m_3^2}(E_1 + E_2),\tag{C.6}$$

where $Q = m_1^2 + m_2^2 + m_3^2 - m_4^2$. The square root comes from energy $E_3(p_3)$. We now define $\hat{\mathbf{p}}_3 = \mathbf{p}_3/p_3$ and after some algebra, we get a quadratic equation for p_3:

$$\left(\delta^2 - \epsilon^2\right)p_3^2 + 2\kappa\delta p_3 + \kappa^2 = 0,\tag{C.7}$$

with

$$\delta = 2\hat{\mathbf{p}}_3\cdot(\mathbf{p}_1 + \mathbf{p}_2),$$
$$\epsilon = 2(E_1 + E_2),$$
$$\kappa = 2P_1 \cdot P_2 + Q.\tag{C.8}$$

Solving this gives two roots $p_3^{(i)}$ of which we only need the non-negative ones. The derivative is given by

$$g'(p_3) = \delta - \epsilon\frac{p_3}{E_3}.\tag{C.9}$$

Using this parameterisation it is now straightforward to integrate eq. (C.5) using e.g. a numerical Monte Carlo method.

This expression for the root simplifies in the ultrarelativistic limit. Setting $E_i = p_i$ one gets that the only non-negative solution to $g(p_3) = 0$ is

$$p_3 = \frac{p_1 p_2 - \mathbf{p}_1 \cdot \mathbf{p}_2}{p_1 + p_2 - \hat{\mathbf{p}}_3 \cdot (\mathbf{p}_1 + \mathbf{p}_2)}\qquad\text{(ultrarelativistic limit)}\tag{C.10}$$

and $g'(p_3) = -2(p_1 + p_2 - \hat{\mathbf{p}}_3 \cdot (\mathbf{p}_1 + \mathbf{p}_2))$.

References

[1] A. Ekstedt, P. Schicho, and T. V. I. Tenkanen, *DRalgo: A package for effective field theory approach for thermal phase transitions,* Comput. Phys. Commun. **288** (2023) 108725 [`2205.08815`].

[2] R. M. Fonseca, *GroupMath: A Mathematica package for group theory calculations,* Comput. Phys. Commun. **267** (2021) 108085 [`2011.01764`].

[3] V. A. Kuzmin, V. A. Rubakov, and M. E. Shaposhnikov, *On the Anomalous Electroweak Baryon Number Nonconservation in the Early Universe,* Phys. Lett. B **155** (1985) 36.

[4] M. E. Shaposhnikov, *Possible Appearance of the Baryon Asymmetry of the Universe in an Electroweak Theory,* JETP Lett. **44** (1986) 465.

[5] M. E. Shaposhnikov, *Baryon Asymmetry of the Universe in Standard Electroweak Theory,* Nucl. Phys. B **287** (1987) 757.

[6] A. G. Cohen, D. B. Kaplan, and A. E. Nelson, *WEAK SCALE BARYOGENESIS,* Phys. Lett. B **245** (1990) 561.

[7] A. G. Cohen, D. B. Kaplan, and A. E. Nelson, *Progress in electroweak baryogenesis,* Ann. Rev. Nucl. Part. Sci. **43** (1993) 27 [`hep-ph/9302210`].

[8] A. Katz and A. Riotto, *Baryogenesis and Gravitational Waves from Runaway Bubble Collisions,* JCAP **11** (2016) 011 [`1608.00583`].

[9] I. Baldes, S. Blasi, A. Mariotti, A. Sevrin, and K. Turbang, *Baryogenesis via relativistic bubble expansion,* Phys. Rev. D **104** (2021) 115029 [`2106.15602`].

[10] A. Azatov, M. Vanvlasselaer, and W. Yin, *Baryogenesis via relativistic bubble walls,* JHEP **10** (2021) 043 [`2106.14913`].

[11] C. Grojean and G. Servant, *Gravitational Waves from Phase Transitions at the Electroweak Scale and Beyond,* Phys. Rev. D **75** (2007) 043507 [`hep-ph/0607107`].

[12] C. Caprini and D. G. Figueroa, *Cosmological Backgrounds of Gravitational Waves,* Class. Quant. Grav. **35** (2018) 163001 [`1801.04268`].

[13] C. Caprini *et al.*, *Detecting gravitational waves from cosmological phase transitions with LISA: an update,* JCAP **03** (2020) 024 [`1910.13125`].

[14] **LISA Cosmology Working Group** Collaboration, P. Auclair *et al.*, *Cosmology with the Laser Interferometer Space Antenna,* Living Rev. Rel. **26** (2023) 5 [`2204.05434`].

[15] K. Kajantie, M. Laine, K. Rummukainen, and M. E. Shaposhnikov, *The Electroweak phase transition: A Nonperturbative analysis,* Nucl. Phys. B **466** (1996) 189 [`hep-lat/9510020`].

[16] K. Kajantie, M. Laine, K. Rummukainen, and M. E. Shaposhnikov, *Is there a hot electroweak phase transition at $m_H \gtrsim m_W$?,* Phys. Rev. Lett. **77** (1996) 2887 [`hep-ph/9605288`].

[17] K. Kajantie, M. Laine, K. Rummukainen, and M. E. Shaposhnikov, *A Nonperturbative analysis of the finite T phase transition in SU(2) x U(1) electroweak theory,* Nucl. Phys. B **493** (1997) 413 [`hep-lat/9612006`].

[18] M. Gurtler, E.-M. Ilgenfritz, and A. Schiller, *Where the electroweak phase transition ends,* Phys. Rev. D **56** (1997) 3888 [`hep-lat/9704013`].

[19] F. Csikor, Z. Fodor, and J. Heitger, *Endpoint of the hot electroweak phase transition,* Phys. Rev. Lett. **82** (1999) 21 [`hep-ph/9809291`].

[20] Y. Aoki, F. Csikor, Z. Fodor, and A. Ukawa, *The Endpoint of the first order phase transition of the SU(2) gauge Higgs model on a four-dimensional isotropic lattice,* Phys. Rev. D **60** (1999) 013001 [`hep-lat/9901021`].

[21] Y. Aoki, G. Endrodi, Z. Fodor, S. D. Katz, and K. K. Szabo, *The Order of the quantum chromodynamics transition predicted by the standard model of particle physics,* Nature **443** (2006) 675 [`hep-lat/0611014`].

[22] T. Bhattacharya *et al.*, *QCD Phase Transition with Chiral Quarks and Physical Quark Masses,* Phys. Rev. Lett. **113** (2014) 082001 [`1402.5175`].

[23] J. R. Espinosa, T. Konstandin, J. M. No, and M. Quiros, *Some Cosmological Implications of Hidden Sectors,* Phys. Rev. D **78** (2008) 123528 [`0809.3215`].

[24] M. Breitbach, J. Kopp, E. Madge, T. Opferkuch, and P. Schwaller, *Dark, Cold, and Noisy: Constraining Secluded Hidden Sectors with Gravitational Waves,* JCAP **07** (2019) 007 [`1811.11175`].

[25] F. Ertas, F. Kahlhoefer, and C. Tasillo, *Turn up the volume: listening to phase transitions in hot dark sectors,* JCAP **02** (2022) 014 [`2109.06208`].

[26] J. R. Espinosa, T. Konstandin, J. M. No, and G. Servant, *Energy Budget of Cosmological First-order Phase Transitions,* JCAP **06** (2010) 028 [`1004.4187`].

[27] M. Hindmarsh, S. J. Huber, K. Rummukainen, and D. J. Weir, *Shape of the acoustic gravitational wave power spectrum from a first order phase transition,* Phys. Rev. D **96** (2017) 103520 [`1704.05871`].

[28] M. Hindmarsh, *Sound shell model for acoustic gravitational wave production at a first-order phase transition in the early Universe,* Phys. Rev. Lett. **120** (2018) 071301 [`1608.04735`].

[29] M. Hindmarsh and M. Hijazi, *Gravitational waves from first order cosmological phase transitions in the Sound Shell Model,* JCAP **12** (2019) 062 [`1909.10040`].

[30] R. Jinno, T. Konstandin, and H. Rubira, *A hybrid simulation of gravitational wave production in first-order phase transitions,* JCAP **04** (2021) 014 [`2010.00971`].

[31] R. Jinno, T. Konstandin, H. Rubira, and I. Stomberg, *Higgsless simulations of cosmological phase transitions and gravitational waves,* JCAP **02** (2023) 011 [`2209.04369`].

[32] C. Caprini, R. Jinno, T. Konstandin, A. Roper Pol, H. Rubira, and I. Stomberg, *Gravitational waves from decaying sources in strong phase transitions,* [`2409.03651`].

[33] C. Gowling and M. Hindmarsh, *Observational prospects for phase transitions at LISA: Fisher matrix analysis,* JCAP **10** (2021) 039 [`2106.05984`].

[34] F. Giese, T. Konstandin, and J. van de Vis, *Finding sound shells in LISA mock data using likelihood sampling,* JCAP **11** (2021) 002 [`2107.06275`].

[35] J. Alvey, U. Bhardwaj, V. Domcke, M. Pieroni, and C. Weniger, *Simulation-based inference for stochastic gravitational wave background data analysis,* Phys. Rev. D **109** (2024) 083008 [2309.07954].

[36] J. De Vries, M. Postma, and J. van de Vis, *The role of leptons in electroweak baryogenesis,* JHEP **04** (2019) 024 [1811.11104].

[37] J. M. Cline and K. Kainulainen, *Electroweak baryogenesis at high bubble wall velocities,* Phys. Rev. D **101** (2020) 063525 [2001.00568].

[38] J. M. Cline and B. Laurent, *Electroweak baryogenesis from light fermion sources: A critical study,* Phys. Rev. D **104** (2021) 083507 [2108.04249].

[39] M. Cataldi and B. Shakya, *Leptogenesis via Bubble Collisions,* [2407.16747].

[40] A. Azatov, M. Vanvlasselaer, and W. Yin, *Dark Matter production from relativistic bubble walls,* JHEP **03** (2021) 288 [2101.05721].

[41] G. F. Giudice, H. M. Lee, A. Pomarol, and B. Shakya, *Nonthermal Heavy Dark Matter from a First-Order Phase Transition,* [2403.03252].

[42] B.-H. Liu, L. D. McLerran, and N. Turok, *Bubble nucleation and growth at a baryon number producing electroweak phase transition,* Phys. Rev. D **46** (1992) 2668.

[43] M. Dine, R. G. Leigh, P. Y. Huet, A. D. Linde, and D. A. Linde, *Towards the theory of the electroweak phase transition,* Phys. Rev. D **46** (1992) 550 [hep-ph/9203203].

[44] G. D. Moore and T. Prokopec, *Bubble wall velocity in a first order electroweak phase transition,* Phys. Rev. Lett. **75** (1995) 777 [hep-ph/9503296].

[45] G. D. Moore and T. Prokopec, *How fast can the wall move? A Study of the electroweak phase transition dynamics,* Phys. Rev. D **52** (1995) 7182 [hep-ph/9506475].

[46] T. Konstandin, G. Nardini, and I. Rues, *From Boltzmann equations to steady wall velocities,* JCAP **09** (2014) 028 [1407.3132].

[47] J. Kozaczuk, *Bubble Expansion and the Viability of Singlet-Driven Electroweak Baryogenesis,* JHEP **10** (2015) 135 [1506.04741].

[48] S. De Curtis, L. D. Rose, A. Guiggiani, A. G. Muyor, and G. Panico, *Bubble wall dynamics at the electroweak phase transition,* JHEP **03** (2022) 163 [2201.08220].

[49] S. Jiang, F. P. Huang, and X. Wang, *Bubble wall velocity during electroweak phase transition in the inert doublet model,* Phys. Rev. D **107** (2023) 095005 [2211.13142].

[50] B. Laurent and J. M. Cline, *Fluid equations for fast-moving electroweak bubble walls,* Phys. Rev. D **102** (2020) 063516 [2007.10935].

[51] G. C. Dorsch, S. J. Huber, and T. Konstandin, *A sonic boom in bubble wall friction,* JCAP **04** (2022) 010 [2112.12548].

[52] B. Laurent and J. M. Cline, *First principles determination of bubble wall velocity,* Phys. Rev. D **106** (2022) 023501 [2204.13120].

[53] S. Balaji, M. Spannowsky, and C. Tamarit, *Cosmological bubble friction in local equilibrium,* JCAP **03** (2021) 051 [2010.08013].

[54] W.-Y. Ai, B. Garbrecht, and C. Tamarit, *Bubble wall velocities in local equilibrium,* JCAP **03** (2022) 015 [2109.13710].

[55] W.-Y. Ai, B. Laurent, and J. van de Vis, *Model-independent bubble wall velocities in local thermal equilibrium,* JCAP **07** (2023) 002 [2303.10171].

[56] M. Sanchez-Garitaonandia and J. van de Vis, *Prediction of the bubble wall velocity for a large jump in degrees of freedom,* Phys. Rev. D **110** (2024) 023509 [2312.09964].

[57] D. Croon, O. Gould, P. Schicho, T. V. I. Tenkanen, and G. White, *Theoretical uncertainties for cosmological first-order phase transitions,* JHEP **04** (2021) 055 [2009.10080].

[58] O. Gould and T. V. I. Tenkanen, *On the perturbative expansion at high temperature and implications for cosmological phase transitions,* JHEP **06** (2021) 069 [2104.04399].

[59] O. Gould and C. Xie, *Higher orders for cosmological phase transitions: a global study in a Yukawa model,* JHEP **12** (2023) 049 [2310.02308].

[60] M. Kierkla, B. Swiezewska, T. V. I. Tenkanen, and J. van de Vis, *Gravitational waves from supercooled phase transitions: dimensional transmutation meets dimensional reduction,* JHEP **02** (2024) 234 [2312.12413].

[61] M. Lewicki, M. Merchand, L. Sagunski, P. Schicho, and D. Schmitt, *Impact of theoretical uncertainties on model parameter reconstruction from GW signals sourced by cosmological phase transitions,* Phys. Rev. D **110** (2024) 023538 [2403.03769].

[62] A. Ekstedt, P. Schicho, and T. V. I. Tenkanen, *Cosmological phase transitions at three loops: the final verdict on perturbation theory,* [2405.18349].

[63] M. Hindmarsh, S. J. Huber, K. Rummukainen, and D. J. Weir, *Numerical simulations of acoustically generated gravitational waves at a first order phase transition,* Phys. Rev. D **92** (2015) 123009 [1504.03291].

[64] F. Giese, T. Konstandin, and J. van de Vis, *Model-independent energy budget of cosmological first-order phase transitions—A sound argument to go beyond the bag model,* JCAP **07** (2020) 057 [2004.06995].

[65] F. Giese, T. Konstandin, K. Schmitz, and J. van de Vis, *Model-independent energy budget for LISA,* JCAP **01** (2021) 072 [2010.09744].

[66] D. Bödeker and G. D. Moore, *Can electroweak bubble walls run away?,* JCAP **05** (2009) 009 [0903.4099].

[67] D. Bödeker and G. D. Moore, *Electroweak Bubble Wall Speed Limit,* JCAP **05** (2017) 025 [1703.08215].

[68] S. Höche, J. Kozaczuk, A. J. Long, J. Turner, and Y. Wang, *Towards an all-orders calculation of the electroweak bubble wall velocity,* JCAP **03** (2021) 009 [2007.10343].

[69] A. Azatov and M. Vanvlasselaer, *Bubble wall velocity: heavy physics effects,* JCAP **01** (2021) 058 [2010.02590].

[70] Y. Gouttenoire, R. Jinno, and F. Sala, *Friction pressure on relativistic bubble walls,* JHEP **05** (2022) 004 [2112.07686].

[71] W.-Y. Ai, *Logarithmically divergent friction on ultrarelativistic bubble walls*, JCAP **10** (2023) 052 [`2308.10679`].

[72] A. J. Long and J. Turner, *Thermal pressure on ultrarelativistic bubbles from a semiclassical formalism*, [`2407.18196`].

[73] J. S. Langer, *Metastable states*, Physica **73** (1974) 61.

[74] K. Enqvist, J. Ignatius, K. Kajantie, and K. Rummukainen, *Nucleation and bubble growth in a first order cosmological electroweak phase transition*, Phys. Rev. D **45** (1992) 3415.

[75] J. Ellis, M. Lewicki, and J. M. No, *On the Maximal Strength of a First-Order Electroweak Phase Transition and its Gravitational Wave Signal*, JCAP **04** (2019) 003 [`1809.08242`].

[76] P. Athron, L. Morris, and Z. Xu, *How robust are gravitational wave predictions from cosmological phase transitions?*, JCAP **05** (2024) 075 [`2309.05474`].

[77] C. L. Wainwright, *CosmoTransitions: Computing Cosmological Phase Transition Temperatures and Bubble Profiles with Multiple Fields*, Comput. Phys. Commun. **183** (2012) 2006 [`1109.4189`].

[78] A. Masoumi, K. D. Olum, and B. Shlaer, *Efficient numerical solution to vacuum decay with many fields*, JCAP **01** (2017) 051 [`1610.06594`].

[79] P. Athron, C. Balázs, M. Bardsley, A. Fowlie, D. Harries, and G. White, *BubbleProfiler: finding the field profile and action for cosmological phase transitions*, Comput. Phys. Commun. **244** (2019) 448 [`1901.03714`].

[80] R. Sato, *SimpleBounce : a simple package for the false vacuum decay*, Comput. Phys. Commun. **258** (2021) 107566 [`1908.10868`].

[81] V. Guada, M. Nemevšek, and M. Pintar, *FindBounce: Package for multi-field bounce actions*, Comput. Phys. Commun. **256** (2020) 107480 [`2002.00881`].

[82] A. Ekstedt, O. Gould, and J. Hirvonen, *BubbleDet: a Python package to compute functional determinants for bubble nucleation*, JHEP **12** (2023) 056 [`2308.15652`].

[83] O. Gould and J. Hirvonen, *Effective field theory approach to thermal bubble nucleation*, Phys. Rev. D **104** (2021) 096015 [`2108.04377`].

[84] J. Löfgren, M. J. Ramsey-Musolf, P. Schicho, and T. V. I. Tenkanen, *Nucleation at Finite Temperature: A Gauge-Invariant Perturbative Framework*, Phys. Rev. Lett. **130** (2023) 251801 [`2112.05472`].

[85] J. Hirvonen, J. Löfgren, M. J. Ramsey-Musolf, P. Schicho, and T. V. I. Tenkanen, *Computing the gauge-invariant bubble nucleation rate in finite temperature effective field theory*, JHEP **07** (2022) 135 [`2112.08912`].

[86] A. Ekstedt, *Bubble nucleation to all orders*, JHEP **08** (2022) 115 [`2201.07331`].

[87] J. Hirvonen, *Nucleation Rate in a High-Temperature Quantum Field Theory with Hard Particles*, [`2403.07987`].

[88] J. Ellis, M. Lewicki, M. Merchand, J. M. No, and M. Zych, *The scalar singlet extension of the Standard Model: gravitational waves versus baryogenesis*, JHEP **01** (2023) 093 [`2210.16305`].

[89] P. B. Arnold and L. G. Yaffe, *Effective theories for real time correlations in hot plasmas*, Phys. Rev. D **57** (1998) 1178 [`hep-ph/9709449`].

[90] P. B. Arnold, D. T. Son, and L. G. Yaffe, *Effective dynamics of hot, soft nonAbelian gauge fields. Color conductivity and log(1/alpha) effects*, Phys. Rev. D **59** (1999) 105020 [`hep-ph/9810216`].

[91] S. Jeon and L. G. Yaffe, *From quantum field theory to hydrodynamics: Transport coefficients and effective kinetic theory*, Phys. Rev. D **53** (1996) 5799 [`hep-ph/9512263`].

[92] L. Landau and E. Lifshitz, *Fluid Mechanics.* Pergamon Press, New York, 1989.

[93] M. Kamionkowski, A. Kosowsky, and M. S. Turner, *Gravitational radiation from first order phase transitions,* Phys. Rev. D **49** (1994) 2837 [`astro-ph/9310044`].

[94] H. Kurki-Suonio and M. Laine, *Supersonic deflagrations in cosmological phase transitions,* Phys. Rev. D **51** (1995) 5431 [`hep-ph/9501216`].

[95] A. Friedlander, I. Banta, J. M. Cline, and D. Tucker-Smith, *Wall speed and shape in singlet-assisted strong electroweak phase transitions,* Phys. Rev. D **103** (2021) 055020 [`2009.14295`].

[96] J. Ignatius, K. Kajantie, H. Kurki-Suonio, and M. Laine, *The growth of bubbles in cosmological phase transitions,* Phys. Rev. D **49** (1994) 3854 [`astro-ph/9309059`].

[97] T. Konstandin and J. M. No, *Hydrodynamic obstruction to bubble expansion,* JCAP **02** (2011) 008 [`1011.3735`].

[98] T. Krajewski, M. Lewicki, and M. Zych, *Bubble-wall velocity in local thermal equilibrium: hydrodynamical simulations vs analytical treatment,* JHEP **05** (2024) 011 [`2402.15408`].

[99] S. De Curtis, L. Delle Rose, A. Guiggiani, A. Gil Muyor, and G. Panico, *Non-linearities in cosmological bubble wall dynamics,* JHEP **05** (2024) 009 [`2401.13522`].

[100] J. P. Boyd, *Chebyshev and Fourier spectral methods.* Dover Publications, New York, 2001.

[101] P. B. Arnold, G. D. Moore, and L. G. Yaffe, *Transport coefficients in high temperature gauge theories. 1. Leading log results,* JHEP **11** (2000) 001 [`hep-ph/0010177`].

[102] P. B. Arnold, G. D. Moore, and L. G. Yaffe, *Photon emission from quark gluon plasma: Complete leading order results,* JHEP **12** (2001) 009 [`hep-ph/0111107`].

[103] P. B. Arnold, G. D. Moore, and L. G. Yaffe, *Effective kinetic theory for high temperature gauge theories,* JHEP **01** (2003) 030 [`hep-ph/0209353`].

[104] H. A. Weldon, *Effective Fermion Masses of Order gT in High Temperature Gauge Theories with Exact Chiral Invariance,* Phys. Rev. D **26** (1982) 2789.

[105] A. Ekstedt, *Two-loop hard thermal loops for vector bosons in general models,* JHEP **06** (2023) 135 [`2302.04894`].

[106] G. P. Lepage, *A New Algorithm for Adaptive Multidimensional Integration,* J. Comput. Phys. **27** (1978) 192.

[107] G. Project, *Monte Carlo Integration - VEGAS Algorithm.* GNU Scientific Library.

[108] S. J. Huber and M. Sopena, *An efficient approach to electroweak bubble velocities,* [`1302.1044`].

[109] F. Karsch, T. Neuhaus, A. Patkos, and J. Rank, *Critical Higgs mass and temperature dependence of gauge boson masses in the SU(2) gauge Higgs model,* Nucl. Phys. B Proc. Suppl. **53** (1997) 623 [`hep-lat/9608087`].

[110] K. Rummukainen, M. Tsypin, K. Kajantie, M. Laine, and M. E. Shaposhnikov, *The Universality class of the electroweak theory,* Nucl. Phys. B **532** (1998) 283 [`hep-lat/9805013`].

[111] N. G. Deshpande and E. Ma, *Pattern of Symmetry Breaking with Two Higgs Doublets,* Phys. Rev. D **18** (1978) 2574.

[112] E. Ma, *Verifiable radiative seesaw mechanism of neutrino mass and dark matter,* Phys. Rev. D **73** (2006) 077301 [`hep-ph/0601225`].

[113] R. Barbieri, L. J. Hall, and V. S. Rychkov, *Improved naturalness with a heavy Higgs: An Alternative road to LHC physics,* Phys. Rev. D **74** (2006) 015007 [`hep-ph/0603188`].

[114] L. Lopez Honorez, E. Nezri, J. F. Oliver, and M. H. G. Tytgat, *The Inert Doublet Model: An Archetype for Dark Matter,* JCAP **02** (2007) 028 [`hep-ph/0612275`].

[115] L. Lopez Honorez and C. E. Yaguna, *The inert doublet model of dark matter revisited,* JHEP **09** (2010) 046 [`1003.3125`].

[116] I. F. Ginzburg, K. A. Kanishev, M. Krawczyk, and D. Sokolowska, *Evolution of Universe to the present inert phase,* Phys. Rev. D **82** (2010) 123533 [`1009.4593`].

[117] T. A. Chowdhury, M. Nemevsek, G. Senjanovic, and Y. Zhang, *Dark Matter as the Trigger of Strong Electroweak Phase Transition,* JCAP **02** (2012) 029 [`1110.5334`].

[118] D. Borah and J. M. Cline, *Inert Doublet Dark Matter with Strong Electroweak Phase Transition,* Phys. Rev. D **86** (2012) 055001 [`1204.4722`].

[119] G. Gil, P. Chankowski, and M. Krawczyk, *Inert Dark Matter and Strong Electroweak Phase Transition,* Phys. Lett. B **717** (2012) 396 [`1207.0084`].

[120] J. M. Cline and K. Kainulainen, *Improved Electroweak Phase Transition with Subdominant Inert Doublet Dark Matter,* Phys. Rev. D **87** (2013) 071701 [`1302.2614`].

[121] N. Blinov, J. Kozaczuk, D. E. Morrissey, and C. Tamarit, *Electroweak Baryogenesis from Exotic Electroweak Symmetry Breaking,* Phys. Rev. D **92** (2015) 035012 [`1504.05195`].

[122] N. Blinov, S. Profumo, and T. Stefaniak, *The Electroweak Phase Transition in the Inert Doublet Model,* JCAP **07** (2015) 028 [`1504.05949`].

[123] P. Basler, M. Krause, M. Muhlleitner, J. Wittbrodt, and A. Wlotzka, *Strong First Order Electroweak Phase Transition in the CP-Conserving 2HDM Revisited,* JHEP **02** (2017) 121 [`1612.04086`].

[124] M. Laine, M. Meyer, and G. Nardini, *Thermal phase transition with full 2-loop effective potential,* Nucl. Phys. B **920** (2017) 565 [`1702.07479`].

[125] M. E. Machacek and M. T. Vaughn, *Two Loop Renormalization Group Equations in a General Quantum Field Theory. 2. Yukawa Couplings,* Nucl. Phys. B **236** (1984) 221.

[126] M. E. Machacek and M. T. Vaughn, *Two Loop Renormalization Group Equations in a General Quantum Field Theory. 1. Wave Function Renormalization,* Nucl. Phys. B **222** (1983) 83.

[127] M. E. Machacek and M. T. Vaughn, *Two Loop Renormalization Group Equations in a General Quantum Field Theory. 3. Scalar Quartic Couplings,* Nucl. Phys. B **249** (1985) 70.

[128] S. P. Martin, *Effective potential at three loops,* Phys. Rev. D **96** (2017) 096005 [`1709.02397`].

[129] S. P. Martin and H. H. Patel, *Two-loop effective potential for generalized gauge fixing,* Phys. Rev. D **98** (2018) 076008 [`1808.07615`].

[130] M. L. Mangano and S. J. Parke, *Multiparton amplitudes in gauge theories,* Phys. Rept. **200** (1991) 301 [`hep-th/0509223`].

[131] M. Srednicki, *Quantum field theory.* Cambridge University Press, 1, 2007.

[132] S. Hannestad and J. Madsen, *Neutrino decoupling in the early universe,* Phys. Rev. D **52** (1995) 1764 [`astro-ph/9506015`].

[133] K. Ala-Mattinen, M. Heikinheimo, K. Kainulainen, and K. Tuominen, *Momentum distributions of cosmic relics: Improved analysis,* Phys. Rev. D **105** (2022) 123005 [`2201.06456`].