How fast does the WAGGO?

Yonsei-Konkuk-Sogang Mini-workshop

Jorinde van de Vis 08/02/2025

Program

Friday 07/02

- Introduction & motivation for wall velocity JvdV
- Equilibrium thermodynamics, example computation, nucleation, matrix elements PS
- Example of `DRalgo` and `WallGoMatrix` model files PS

Saturday 08/02

- `WallGo` source code JvdV
- Worked out example for `WallGo` JvdV & PS:
  Matrix elements
  Example base file
  Collision model
  Model file
- Hands-on session:
  start of implementation of `WallGo` model

Monday 10/02

- Discussion of configuration parameters and convergence JvdV
- Hands-on session:
  implementation of `WallGo` model & Q&A

The src folder

```
Jorinde@UU-C02J44ZNQ6LR WallGo % ls src/WallGo
PotentialTools              equationOfMotion.py         hydrodynamicsTemplateModel.py
__init__.py                 exceptions.py               interpolatableFunction.py
__pycache__                 fields.py                   manager.py
boltzmann.py                freeEnergy.py               mathematicaHelpers.py
collisionArray.py           genericModel.py             particle.py
collisionHelpers.py         grid.py                     polynomial.py
config.py                   grid3Scales.py              results.py
containers.py               helpers.py                  thermodynamics.py
effectivePotential.py       hydrodynamics.py            utils.py
Jorinde@UU-C02J44ZNQ6LR WallGo %
```

Computing a wall velocity in `WallGo` does not require modification of the source code, but knowledge of the source code can be helpful for choosing the appropriate configuration settings

`WallGo` is typically run from a model file; we will discuss these later

API reference

https://wallgo.readthedocs.io/en/latest/_autosummary/WallGo.html

- WallGo.PotentialTools
- WallGo.boltzmann
- WallGo.collisionArray
- WallGo.collisionHelpers
- WallGo.config
- WallGo.containers
- WallGo.effectivePotential
- WallGo.equationOfMotion
- WallGo.exceptions
- WallGo.fields
- WallGo.freeEnergy
- WallGo.genericModel
- WallGo.grid
- WallGo.grid3Scales
- WallGo.helpers
- WallGo.hydrodynamics
- WallGo.hydrodynamicsTemplateModel
- WallGo.interpolatableFunction
- WallGo.manager
- WallGo.mathematicaHelpers
- WallGo.particle

/ WallGo

WallGo

Initialising WallGo package

Functions

`isCollisionModuleAvailable` ()	Returns True if the WallGoCollision extension module could be loaded an

`isCollisionModuleAvailable()` [source]

Returns True if the WallGoCollision extension module could be loaded and is ready for use. By default it is loaded together with WallGo, but WallGo can operate in restricted mode even if the load fails. This function can be used to check module availability at runtime if you must operate in an environment where the module may not always be available.

Return type: bool

Modules

`PotentialTools`	Initialisation for PotentialTools module, includes loading of Jb/Jf integral d
`boltzmann`	Classes for solving the Boltzmann equations for out-of-equilibrium particle
`collisionArray`	Class for loading and storing the collision files used in boltzmann.py.

`manager.py`

- Loads the configuration file (default in src/WallGo folder), to be discussed on Monday
- Performs all initializations in the correct order (e.g. interpolates the `freeEnergy`, loads the collisions)
- Verifies the input, e.g.:
  - Does the potential have two different phases
  - Is the number of grid points chosen correctly (it has to be an odd number)
- Has the function `manager.solveWall(wallSolverSettings)` which returns the wall velocity

Recall: scalar field equation of motion and EM-conservation

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

$$T^{30} = w\gamma_{\text{pl}}^2 v_{\text{pl}} + T^{30}_{\text{out}} = c_1$$

$$T^{33} = \frac{1}{2}(\partial_z\phi_i)^2 - V_{\text{eff}}(\overrightarrow{\phi}, T) + w\gamma_{\text{pl}}^2 v_{\text{pl}}^2 + T^{33}_{\text{out}} = c_2$$

`effectivePotential.py`

- `effectivePotential.py`
  - $V_{\text{eff}}(\phi_i, T)$; don't forget to include the field-independent parts here (e.g. T^4)
  - The effective potential is model-dependent, so `effectivePotential.py` is an Abstract Base Class: the user defines the effective potential in the model file

`effectivePotential.py` and `freeEnergy.py`

- `effectivePotential.py`
  - $V_{\text{eff}}(\phi_i, T)$; don't forget to include the field-independent parts here (e.g. T^4)
  - The effective potential is model-dependent, so `effectivePotential.py` is an Abstract Base Class: the user defines the effective potential in the model file
- `freeEnergy.py`
  - Holds the value of the potential at its minima
  - Typically is an `interpolatableFunction` (interpolation is called by the manager); this is done to reduce the computation time significantly

Limited temperature range

- Often one of the phases ceases to exist above/below a certain T
- `freeEnergy` keeps track of these minimum and maximum temperatures
- The plasma can not exceed these temperatures; this sometimes puts a limit on the wall velocity

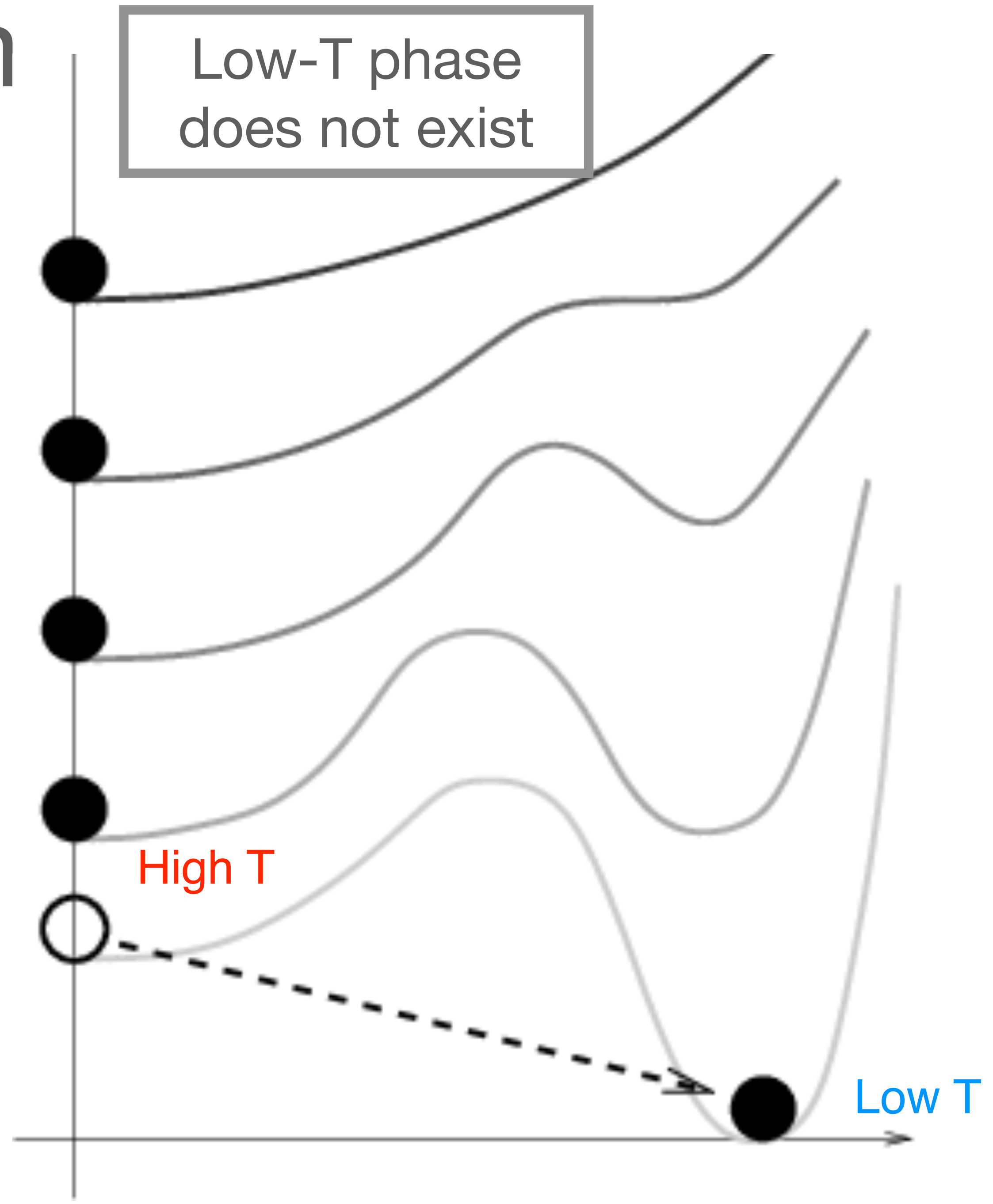

Figure: Rubakov, 2015

thermodynamics.py

- Holds the pressure, enthalpy, energy density and speeds of sound necessary for the hydrodynamics computations, derived from the `freeEnergy`
- Extrapolates the equation of state if the temperature is outside of the allowed range; by mapping onto the “template model”
- This is mere for numerical convenience in `hydrodynamics`; we enforce that the final wall velocity does not depend on this extrapolation

$$p_s = \frac{1}{3}a_+ T^\mu - \epsilon \qquad p_b = \frac{1}{3}a_- T^\nu$$

$$\mu = 1 + \frac{1}{c_{s,\mathrm{sym}}^2} \qquad \nu = 1 + \frac{1}{c_{s,\mathrm{brok}}^2}$$

Leitao, Megevand, 2015

`equationOfMotion.py`

- Solves the scalar field equation of motion, by using a Tanh-Ansatz, and minimizing its action
- Solves energy-momentum conservation to determine the fluid velocity and temperature profiles
- Calls `hydrodynamics.findHydroBoundaries(wallVelocity)` to determine the boundary conditions for the EM-conservation equations
- Calls `self.boltzmannSolver.getDeltas()` to find the out-of-equilibrium contribution for the list of out-of-equilirbium `particles`
- Goes through several iterations to conserve to the right solution for each v_w
- Separate functions for deflagrations/hybrids and detonations
- Defl/hybr: varies the wall velocity between `vmin` (typically 0), `vmax` (Jouguet velocity or given by limited temperature range) to find $P(v_w) = 0$
- Det: looks for $P(v_w) = 0$ starting from Jouguet velocity

hydrodynamics.py

- Finds the boundary conditions for the energy-momentum conservation equations
- Finds the Jouguet velocity (transition between hybrids and detonations)
- Finds the maximum velocity allowed by the limited temperature range

hydrodynamicsTemplateModel.py

- Solves hydrodynamics in the template model
- Results are often very close to the results in the full model-dependent hydrodynamics
- Results from `hydrodynamicsTemplateModel.py` are only used to find reasonable initial values in initialization and in `hydrodynamics`

$$p_s = \frac{1}{3} a_+ T^\mu - \epsilon \qquad p_b = \frac{1}{3} a_- T^\nu$$

$$\mu = 1 + \frac{1}{c^2_{s,\mathrm{sym}}} \qquad \nu = 1 + \frac{1}{c^2_{s,\mathrm{brok}}}$$

Leitao, Megevand, 2015

Recall: Boltzmann equation

Rescaled coordinates

Restricted Chebyshev polynomials

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

Algebraic equation

$$\sum_{i,j,k} \left\{ \partial_\xi \chi \left[\mathscr{P}_w \partial_\chi - \frac{\gamma_w}{2} \partial_\chi (m^2)(\partial_{p_z} \rho_z) \partial_{\rho_z} \right] \bar{T}_i(\chi) \bar{T}_j(\rho_z) \tilde{T}_k(\rho_\|) \delta f^a_{ijk} + \bar{T}_i(\chi) \mathscr{C}^{\text{lin}}_{ab} \left[\bar{T}_j(\rho_z) \tilde{T}_k(\rho_\|) \right] \delta f^b_{ijk} \right\} = \mathscr{S}_a(\chi, \rho_z, \rho_\|)$$

Introduce a grid to convert it to a matrix equation

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

Grid indices

`boltzmann.py`

- Solves the Boltzmann equations for all the out-of-equilibrium particles
- Uses pre-computed collision output — loaded in `manager`:
  `boltzmannSolver.loadCollisions(self.collisionDirectory)`
- Returns the out-of-equilibrium contributions in a `BoltzmannResults` object

`grid3Scales.py`

- It is not numerically efficient to solve the equation of motion on a linear scale; we thus rescale the ξ-coordinate to get many points close to the center of the wall
- We use different rescalings in the tails, and in the bubble wall region

Additional classes

- Helper functions contained in: `collisionHelpers.py, helpers.py mathematicaHelpers.py, utils.py`
- WallGo-specific objects/data classes: `fields.py, containers.py, exceptions.py, polynomial.py results.py`

Folder: src/WallGo/PotentialTools

- Contains functions for the one-loop effective potential without high-temperature expansion
- Tables for the $J_{B/F}$ functions
- Options for how to deal with negative arguments (principal value, absolute value of the argument, absolute value of analytically continued integral)

Questions?