VAH User Manual

M. McNelis[a,*]

[a]*Department of Physics, The Ohio State University, Columbus, OH 43210-1117, USA*

Contents

1 Setup **2**

2 Running the code **2**

3 Parameters **3**
- 3.1 Spatial grid . . . 3
- 3.2 Adaptive time step . . . 3
- 3.3 Bayesian model parameters . . . 3

4 Source code **3**
- 4.1 Simulation . . . 3
- 4.2 Semi-analytic . . . 7
- 4.3 Equation of state and transport coefficients . . . 8
- 4.4 Initial conditions . . . 9
- 4.5 Other . . . 9

5 Workflow **10**

6 Initial conditions **14**
- 6.1 Conformal Bjorken flow test . . . 14
- 6.2 Conformal Gubser flow test . . . 15
- 6.3 Nonconformal Bjorken flow test . . . 16
- 6.4 3+1d conformal hydrodynamic models with smooth T$_R$ENTo profile . . . 17

*Email address: mcnelis.9@osu.edu

1. Setup

The code's default `Makefile` uses the `gcc` compiler. Alternatively, you could use the `icpc` compiler if you have OpenMP support and the library `qopenmp` installed. You can switch out the `Makefile` by doing

```
sh makefiles.sh icpc        # or gcc
```

You may the edit the `Makefile` for your computer, but the GSL libraries `lgsl` and `lgslcblas` need to be installed.

To set up the code on the Ohio Supercomputer Center, for example, login and do

```
git clone https://github.com/mjmcnelis/cpu_vah.git
cd cpu_vah && sh makefile.sh icpc
module load intel/19.0.3
module load python/3.6-conda5.2
```

The scripts in `python` are used to generate model parameter samples and train the auto–grid, but the code can run without them.

2. Running the code

To compile and run the hydrodynamic simulation once, simply type

```
sh hydro.sh 1
```

The code will run with the default runtime parameters in the `parameters` directory and macro parameters in `rhic/include/Macros.h`.[1,2] The results are stored in `output` (or memory). For multiple runs, do

```
sh hydro.sh n               # n = number of events
```

The script is useful for the simpler test runs (e.g. Gubser flow) on your computer, but keep in mind that it clears `output` prior to compiling.

Alternatively, you can clear the results once and run your events (or jobs) by doing

[1] The default mode runs a 2+1d central Pb+Pb collision with nonconformal anisotropic hydrodynamics and fluctuating T$_\mathrm{R}$ENTo initial conditions (customized version). The freezeout surface is written to file.

[2] If you edited the parameters, you can restore the default values by copying the files in `parameters/default` to the appropriate directories.

```
sh clear_results.sh
make clean
make
for((i = 0; i < n; i++))     # n = number of events (or jobs)
do
    ./cpu_vah                # or submit job
done
```

This routine is useful for the tests that require multiple job submissions.

3. Parameters

3.1. Spatial grid

3.2. Adaptive time step

3.3. Bayesian model parameters

4. Source code

The source and header files can be found in `rhic`. We briefly summarize each file and list the main classes, structs and functions.

4.1. Simulation

`Main.cpp`
`Hydrowrapper.cpp`

– Creates an instance of the `HYDRO` class, which runs the hydrodynamic simulation and stores the particlization hypersurface (or outputs results to file). The wrapper can run as a stand-alone program or integrated into a larger program (e.g. JETSCAPE).

`HYDRO`

`start_hydro_no_arguments()`
`store_freezeout_surface()`

`Parameters.cpp`

– Reads the parameter files in `parameters` and sets the runtime parameters

in the structs below. The impact parameter and Bayesian model parameters can be overwritten by the random samples in `python/model_parameters`.

`hydro_parameters`
`initial_condition_parameters`
`lattice_parameters`

`load_hydro_parameters()`
`load_initial_condition_parameters()`
`load_lattice_parameters()`

`DynamicalVariables.cpp`

– Allocates memory for the dynamical and inferred variables on the spatial grid at a given time. The code makes use of `extern` variables and structs to store and access the hydrodynamic quantities.

`hydro_variables`
`fluid_velocity`

`allocate_memory()`
`free_memory()`

`InitialConditions.cpp`

– Sets the initial conditions for the dynamical and inferred variables. The energy density is the most sensitive to the type of initial-state model.

`set_initial_conditions()`

`Hydrodynamics.cpp`

– Configures the spatial grid and evolves the hydrodynamic equations until all fluid cells are below the particlization switching temperature.

`run_hydro()`

`KurganovTadmor.cpp`

– Computes the intermediate Euler steps in the Runge–Kutta scheme using the Kurganov–Tadmor algorithm. The hydrodynamic variables are then updated with the averaged iteration.

`evolve_hydro_one_time_step()`
`euler_step()`

`FluxTerms.cpp`

– Computes the flux terms in the Kurganov–Tadmor algorithm.

`flux_terms()`

`SourceTerms.cpp`

– Computes the external source terms in the Kurganov–Tadmor algorithm.

`source_terms_aniso_hydro()`
`source_terms_viscous_hydro()`

`NeighborCells.cpp`

– Collects the neighbor cells of the fluid cell being evaluated. They are used to compute the spatial derivatives in the flux and source terms.

`get_hydro_variables_neighbor_cells()`
`get_fluid_velocity_neighbor_cells()`

`InferredVariables.cpp`

– Reconstructs the energy density and fluid velocity after each iteration.

`set_inferred_variables_aniso_hydro()`
`set_inferred_variables_viscous_hydro()`

`AnisoVariables.cpp`

– Reconstructs the anisotropic variables after each iteration. The anisotropic variables of a single fluid cell are stored in the `aniso_variables` struct.

`aniso_variables`

`find_anisotropic_variables()`
`set_anisotropic_variables()`

`Regulation.cpp`

– Regulates the residual shear stress (or standard shear stress and bulk viscous pressure) after each iteration.

`regulate_residual_currents()`
`regulate_viscous_currents()`

`GhostCells.cpp`

– Sets the boundary conditions for the ghost cell layers interfaced with the physical spatial grid.

`set_ghost_cells()`

`AdaptiveTimeStep.cpp`

– Computes the adaptive time step for the next Runge–Kutta iteration.

`compute_dt_source()`
`compute_dt_CFL()`

`FreezeoutFinder.cpp`

– The `freezeout_finder` class holds the hydrodynamic variables from the current and previous spatial grids. The energy density hypercubes are then constructed and passed to the `Cornelius` class from `cornelius-c++-1.3` to search for freezeout cells. The freezeout cells' centroid position and surface element vector, along with the interpolated hydrodynamic variables, are appended to the `freezeout_surface` struct.

freezeout_finder
Cornelius

freezeout_surface

load_initial_grid()
load_current_grid()
find_3d_freezeout_cells()

4.2. Semi-analytic

AnisoBjorken.cpp
ViscousBjorken.cpp

- Evolve the (non)conformal Bjorken semi-analytic solutions for anisotropic and viscous hydrodynamics

run_semi_analytic_aniso_bjorken()
run_semi_analytic_viscous_bjorken()

AnisoGubser.cpp
ViscousGubser.cpp
IdealGubser.cpp

- Evolve the conformal Gubser semi-analytic solutions for anisotropic, viscous and ideal hydrodynamics.

run_semi_analytic_aniso_gubser()
run_semi_analytic_viscous_gubser()
run_analytic_ideal_gubser()

4.3. Equation of state and transport coefficients

`EquationOfState.cpp`

– Reads the energy density and computes the equilibrium temperature, which is stored in the `equation_of_state` class. Other thermodynamic variables such as the equilibrium pressure and beta transport coefficients can also be evaluated.

`equation_of_state`

`equilibrium_pressure()`
`beta_shear()`
`beta_bulk()`

`Viscosities.cpp`

– Computes the JETSCAPE SIMS temperature-dependent parameterization of the specific shear and bulk viscosities.

`eta_over_s()`
`zeta_over_s()`

`TransportViscous.cpp`

– The `viscous_transport_coefficients` class computes and stores the second-order transport coefficients in viscous hydrodynamics.

`viscous_transport_coefficients`

`compute_shear_transport_coefficients()`
`compute_bulk_transport_coefficients()`

`TransportAniso.cpp`
`TransportAnisoNonconformal.cpp`

– The `aniso_transport_coefficients` classes compute and store the conformal and nonconformal transport coefficients in anisotropic hydrodynam-

ics.

aniso_transport_coefficients
aniso_transport_coefficients_nonconformal

compute_transport_coefficients()

4.4. Initial conditions

`Trento.cpp`

– A simple, customized version of the TRENTo code to make fluctuating (or event-averaged) initial energy density profiles for the code validation tests.

set_trento_energy_density_and_flow_profile()

4.5. Other

`Projections.cpp`

– The projection classes compute and store the spatial (or transverse) projection tensors. These tensors can then be used to project any vector or rank-2 tensor along the spatial (or transverse) directions.

`Output.cpp`

– Outputs hydrodynamic quantities from the simulation (or semi-analytic solution) to file. This is used to study the hydrodynamic evolution in the code validation tests.

output_hydro_simulation()
output_semi_analytic_solution_if_any()

`Print.cpp`

– Prints the hydrodynamic simulation model, parameters, runtime status

and benchmarks.

`Memory.cpp`

– Allocates memory for multi-dimensional arrays. They are mainly used by the routines in `freezeout_finder`.

`Precision.h`

– The macro variable type `precision` determines the numerical precision of float-type variables in the algorithm.[3] Almost all the float-type variables in the code are declared as `precision`.

`LatticeData.h`

– Contains a hard-coded table of the QCD equilibrium pressure as a backup to the rational polynomial fit used in the `equation_of_state` class.

5. Workflow

The program creates an instance of the `HYDRO` class and starts the hydrodynamic simulation. The particlization hypersurface is deallocated at the end of the program.[4]

```
// note: .. represents arguments
int main(..)
{
    HYDRO vah;                          // hydro wrapper
    vah.start_hydro_no_arguments();     // start hydro simulation
    vah.free_freezeout_surface();       // deallocate hypersurface
    return 0;
}
```

[3] Only the variable type `double` works at the moment.
[4] Here we oversimplify the code blocks to highlight the main features of the program.

By default, the HYDRO wrapper runs as a stand-alone program but it can integrated into a larger program, such as the JETSCAPE framework.

If the code crashes or fails to finish within the allotted time, the particlization hypersurface written to memory (or to the file `output/surface.dat`) will be empty.

```
void HYDRO::store_freezeout_surface(freezeout_surface surface)
{   // load surface components to individual vectors in HYDRO
    for(long i = 0; i < total_cells; i++)
    {
        tau.push_back(surface.tau[i]);              // x^\mu
        x.push_back(surface.x[i]);
        y.push_back(surface.y[i]);
        eta.push_back(surface.eta[i]);
        dsigma_tau.push_back(surface.dsigma_tau[i]);// dsigma_\mu
        dsigma_x.push_back(surface.dsigma_x[i]);
        dsigma_y.push_back(surface.dsigma_y[i]);
        dsigma_eta.push_back(surface.dsigma_eta[i]);
        ux.push_back(surface.ux[i]);                // u^\mu
        uy.push_back(surface.uy[i]);
        un.push_back(surface.un[i]);
        E.push_back(surface.E[i]);                  // e
        T.push_back(surface.T[i]);                  // T
        P.push_back(surface.P[i]);                  // Peq
        pixx.push_back(surface.pixx[i]);            // pi^\munu
        pixy.push_back(surface.pixy[i]);
        pixn.push_back(surface.pixn[i]);
        piyy.push_back(surface.piyy[i]);
        piyn.push_back(surface.piyn[i]);
        Pi.push_back(surface.Pi[i]);                // Pi
    }
}
void HYDRO::start_hydro_no_arguments()
{
    // read runtime parameters
    hydro_parameters hydro = load_hydro_parameters(..);
    lattice_parameters lattice = load_lattice_parameters(..);
    initial_condition_parameters initial;
    initial = load_initial_condition_parameters(..);

    // run hydro simulation and store hypersurface
    store_freezeout_surface(run_hydro(..));
}
```

```
freezeout_surface run_hydro(..)
{
    allocate_memory(..);            // grid allocation

    double t = tau_initial;         // starting time
    double dt = 0.05 * t;           // time step

    set_initial_conditions(..);     // initialize grid
    set_ghost_cells();

    freezeout_finder finder(..);    // initialize freezeout finder
    finder.load_initial_grid(..);

    // evolution loop
    for(int n = 0; n < max_time_steps; n++)
    {
        if(n > 0)
        {
            dt = set_time_step(..);     // adaptive time step

            // load current grid and search for freezeout cells
            finder.load_current_grid(..);
            finder.find_3d_freezeout_cells(..);
            if(all_cells_below_freezeout_temperature(..))
            {
                break;                  // stop hydro evolution
            }
        }

        evolve_hydro_one_time_step(..); // RK2 iteration
        t += dt;                        // update time
    }

    free_memory();                  // deallocate grid

    return finder.surface;          // return hypersurface
}
```

```
void evolve_hydro_one_time_step(..)
{
    // first intermediate Euler step
    euler_step(..);                          // compute qI
    swap_fluid_velocity();                   // swap u <=> up
    set_inferred_variables_aniso_hydro();    // compute (e,u)
    set_anisotropic_variables();             // compute X
    regulate_residual_currents();            // regulate qI
    set_ghost_cells();                       // for (qI,u)

    // second intermediate Euler step
    euler_step(..);                          // RK2 update => Q
    set_inferred_variables_aniso_hydro();    // compute (e,u)
    set_anisotropic_variables();             // compute X
    regulate_residual_currents();            // regulate Q
    swap_hydro_variables();                  // swap q <=> Q
    set_ghost_cells();                       // for (q,u)
}
```

6. Initial conditions

Several initial condition models are built into the code. They are primarily used for testing the simulation and comparing hydrodynamic models. For practical applications, you will want to read in the energy density profile from an initial-state module (see last subsections).

6.1. Conformal Bjorken flow test

To run anisotropic hydrodynamics with conformal Bjorken flow, adjust the following `parameters` and `MACROS`:

```
run_hydro = 1
tau_initial = 0.01
pl_pt_ratio_initial = 0.001
temperature_etas = 0
constant_etas = 0.2
freezeout_temperature_GeV = 0.136
```

```
initial_condition_type = 1
initial_central_temperature_GeV = 1.05

lattice_points_x = 1
lattice_points_y = 1

#define BOOST_INVARIANT
#define CONFORMAL
```

Turning off the `temperature_etas` switch sets the shear viscosity $\eta/\mathcal{S}$ to the constant value `constant_etas`.

The `run_hydro` mode outputs both the simulation results and semi-analytic solution. Alternatively, you can run the conformal Bjorken test by doing

```
cd scripts/conformal_bjorken
sh run_conformal_bjorken_test.sh
```

which copies the files from `tests/conformal_bjorken/parameters` to the appropriate directories and runs the code. You can then go to the Mathematica notebook in `tests/conformal_bjorken` to plot the test results.

6.2. Conformal Gubser flow test

To run anisotropic hydrodynamics with conformal Gubser flow, adjust the parameters

```
run_hydro = 1
tau_initial = 0.01
pl_pt_ratio_initial = 0.001
temperature_etas = 0
constant_etas = 0.2
freezeout_temperature_GeV = 0.065

initial_condition_type = 3
initial_central_temperature_GeV = 1.05

lattice_points_x = 281
lattice_points_y = 281
lattice_spacing_x = 0.05
```

```
lattice_spacing_y = 0.05

#define BOOST_INVARIANT
#define CONFORMAL
```

or run the script

```
cd scripts/conformal_gubser
sh run_conformal_gubser_test.sh
```

You can plot the simulation results and semi-analytic solution with the Mathematica notebook in `tests/conformal_gubser`

6.3. Nonconformal Bjorken flow test

To run nonconformal anisotropic hydrodynamics with Bjorken flow, adjust the parameters

```
run_hydro = 1
tau_initial = 0.05
pl_pt_ratio_initial = 0.3
kinetic_theory_model = 0 or 1
temperature_etas = 1
freezeout_temperature_GeV = 0.136

initial_condition_type = 1
initial_central_temperature_GeV = 0.718

lattice_points_x = 1
lattice_points_y = 1

#define ANISO_HYDRO
#define BOOST_INVARIANT
//#define CONFORMAL
```

Turning on the `temperature_etas` switch uses the temperature-dependent parameterization for $(\eta/\mathcal{S})(T)$.

Make sure you comment the `CONFORMAL` macro to use the QCD equation of state.

To run the nonconformal second-order viscous hydrodynamic models, comment `ANISO_HYDRO` and set `kinetic_theory_model` to either 0 (standard) or 1 (quasiparticle).

You can also do

```
cd scripts/lattice_bjorken
sh run_lattice_bjorken_test.sh vah      # or vh, vh2
```

to run the test with one of the three hydrodynamic models. Once you run all three tests, plot them using the notebook in `tests/lattice_bjorken`.

6.4. 3+1d conformal hydrodynamic models with smooth T$_R$ENTo *profile*

To run 3+1d conformal anisotropic hydrodynamics with smooth T$_R$ENTo initial conditions, adjust the parameters

```
run_hydro = 1
tau_initial = 0.01
pl_pt_ratio_initial = 0.001
temperature_etas = 0
constant_etas = 0.2
freezeout_temperature_GeV = 0.165

initial_condition_type = 4
trento_average_over_events = 1

lattice_points_x = 281
lattice_points_y = 281
lattice_spacing_x = 0.1
lattice_spacing_y = 0.1
resolve_nucleons = 1
fit_rapidity_plateau = 1

#define ANISO_HYDRO
//#define BOOST_INVARIANT
#define CONFORMAL
```

Turning on the `trento_average_over_events` switch event-averages multiple fluctuating T$_R$ENTo events to make a smooth initial energy density profile.

Turning on the resolve_nucleons switch replaces the transverse lattice spacing with $0.2 \times$ trento_nucleon_width; the transverse grid points are also adjusted to keep the original grid length ($L_x = L_y = 28$ fm in this case).

Comment BOOST_INVARIANT to run the simulation in (3+1)–dimensions; turning on the fit_rapidity_plateau switch automatically configures the lattice_points_eta and lattice_spacing_eta parameters to fit and resolve the custom rapidity plateau extension used in the paper.

To run conformal standard viscous hydrodynamics, comment the macro ANISO_HYDRO.

You can also run the script

```
cd scripts/conformal_trento
sh run_conformal_trento_job.sh vah      # or vh
```

which submits a job to run the code with OpenMP acceleration. If you don't have access to a computing node, you can run

```
cd scripts/conformal_trento
sh run_conformal_trento_local.sh vah    # or vh
```

on your computer, although it would take much longer. Once you run the anisotropic and viscous hydrodynamic models, you can compare them in the notebook in tests/conformal_trento.