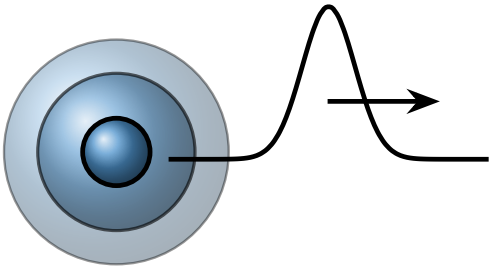
APECSS

(Acoustic Pulse Emitted by Cavitation in Spherical Symmetry)



Fabian Denner
Sören Schenke
Pierre Coulombel

11 July 2025

Contents

Chapter 1

About APECSS

APECSS is a software toolbox to compute pressure-driven bubble dynamics and the resulting acoustic emissions. It is written exclusively in `C` and has been developed with simplicity, versatility and performance in mind. The acronym APECSS stands for "Acoustic Pulse Emitted by Cavitation in Spherical Symmetry".

The main features of APECSS are:

- Bubble dynamics using widely-used models (Rayleigh-Plesset, Keller-Miksis, Gilmore).
- Acoustic emissions of the bubble under different assumptions (incompressible, quasi-acoustic, fully compressible).
- Acoustic interactions of multiple bubbles under different assumptions (incompressible, quasi-acoustic)
- Prediction of the formation and attenuation of shock fronts emitted by the bubble.
- Viscoelastic media (Kelvin-Voigt, Zener, Oldroyd-B).
- Lipid monolayer coating of the bubble as used for ultrasound contrast agents.
- All ODEs are solved with in-built fourth- and fifth-order Runge-Kutta scheme with adaptive time stepping.
- APECSS has, aside from the C standard library, no external dependencies.

The APECSS repository is located at `https://github.com/polycfd/apecss` and structured as:

- The `documentation/` folder contains this short documentation of APECSS.
- The `examples/` folder contains representative examples of how to use APECSS and to demonstrate the most important features of APECSS.
- The `include/` folder contains the `apecss.h` header file, in which all variables, macros and functions of APECSS are defined.
- The `lib/` folder in which the APECSS library is compiled (at least if you follow the installation instructions in Section 2.2).
- The `src/` folder contains all source files (`*.c`) of APECSS.
- The `.clang-format` file, which defines the formatting rules for the source code, see Section 2.4.7.
- The `.gitignore` file tells *git* which folders and files to ignore.
- The `CITATION.cff` file contains information on how to cite APECSS.
- The `CODE_OF_CONDUCT.md` file outlines the code of conduct for this repository.
- The `CONTRIBUTING.md` file contains brief guidelines on how to contribute to APECSS.
- The `JOSS-Paper.pdf` file is the paper about APECSS in the Journal of Open Source Software.
- The `LICENSE.txt` file contains the Mozilla Public License Version 2.0.
- The `README.md` file with the most important information about APECSS.

The development of APECSS has directly benefitted from research funding provided by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant number 441063377, and by the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2024-04805.

Chapter 2

Using APECSS

2.1 Requirements

APECSS has been developed and tested on Unix systems, using Linux and MacOS operating systems. APECSS has also been installed and executed successfully on Windows 11. The only mandatory requirement to compile and run APECSS is a standard C compiler, such as `gcc`.

Optionally, `cmake` (version 3.12 or higher) is required if you would like to make use of the provided compilation and test scripts. To use the provided Python scripts for visualizing the output of APECSS, `Python` (version 3), `numpy` and `matplotlib` are required.

2.2 Installation

2.2.1 Unix installation

Installing APECSS on Unix systems, such as Linux and MacOS, is easy. After downloading APECSS in the folder `<path to APECSS>`, define the following environment variables:

- `APECSS_DIR` to the folder in which APECSS is located. Using `bash`, for instance, simply execute the command `export APECSS_DIR=<path to APECSS>` or, even better, add this command to your `bash` profile.
- `USRLIB_DIR` to the folder in which `libm.a` or `libm.dylib` (the standard *math* library) is located. This may, for instance, be `/usr/lib64/` on Linux systems or `/usr/lib/` on MacOS systems.

Now, navigate into the folder `$APECSS_DIR/lib` and execute `./compile_lib.sh`. This shell script will compile the APECSS library using `cmake` with the `CMakeLists.txt` file provided in this folder. By default, APECSS is compiled with double precision and in *Release* mode, meaning all optimization flags are enabled. That's it, you've successfully installed APECSS!

If you wish to install APECSS with *Debug* flags, simply change the line

```
cmake CMakeLists.txt -DCMAKE_BUILD_TYPE=Release
```

to

```
cmake CMakeLists.txt -DCMAKE_BUILD_TYPE=Debug
```

in the `./compile_lib.sh` shell script and, if applicable, in the shell script that compiles the desired example.

2.2.2 Windows 11 installation

Installing APECSS on Windows is a little more involved than the installation on Unix. The installation[1] described below presents a minimalist installation of APECSS on Windows 11, using `MinGW` and `MSYS2`.

Installation of MSYS2

`MSYS2` is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software.

1. Download and install `MSYS2` from https://www.msys2.org/.
2. At the end of the installation, leave the *Run MSYS2 now* option checked and click *Finish*. `MSYS2` is now ready for use and a terminal will launch.
3. It's also necessary to install the `gcc` compiler, `cmake` and `make`. Run the following command:

   ```
   pacman -S mingw-w64-x86_64-gcc
   pacman -S mingw-w64-x86_64-cmake
   pacman -S mingw-w64-x86_64-make
   ```

   Enter `Y` when prompted to proceed with the installation. Note that all libraries will be downloaded in the `mingw64` folder. To check the installation, you have to use the `MSYS2 MINGW64` terminal and run these commands:

   ```
   gcc --version
   cmake --version
   mingw32-make --version
   ```

   You can now close the window.
4. Add the `MinGW` path to your system's `PATH` environment variable. For example: `C:\msys64\mingw64\bin`.
5. You can now use the `gcc` compilation tools on your Windows machine.

Installation of Git Bash

Git Bash is a terminal emulator for Windows that allows you to use Git and Unix/Linux commands in a Windows environment. Git Bash can be installed from https://git-scm.com/downloads.

Compiling APECSS

Open a Git Bash terminal in the folder of your choice and clone the APECSS repository.

```
git clone <repo>
```

To run, the scripts require some adaptations due to differences between Unix and Windows:

1. In the file `./lib/compile_lib.sh`, modify the lines:

   ```
   # cmake CMakeLists.txt -DCMAKE_BUILD_TYPE=Release
   cmake CMakeLists.txt -DCMAKE_BUILD_TYPE=Release -G "MinGW Makefiles"
   # make
   mingw32-make
   ```

   The first line instructs the compiler to generate Unix `Makefiles`, and the second line accounts for the difference in the `make` command's name between Windows and Unix.
2. In the file `./lib/CMakeLists.txt`, modify the line:

[1] We are grateful to Maxime Montois for these detailed installation instructions, and to Pierre Coulombel for further improvements.

```
# set(CMAKE_C_COMPILER /usr/bin/gcc)
set(CMAKE_C_COMPILER "C:/msys64/mingw64/bin/gcc.exe")
```

This changes the compiler path. Adjust the path if you installed `MSYS2` in a different location.

3. In the file `./src/bubble.c`, remove the second argument from the `mkdir` function. For example:

```
//mkdir(Bubble->Results->dir, 0700)
mkdir(Bubble->Results->dir)
```

On Windows, the `mkdir` function only takes one argument; the second argument for permissions does not exist.

You can also use the `make` command without using `mingw32-make` by copying the `mingw32-make.exe` file to `C:\msys64\mingw64\bin` (or your `MSYS2` installation path) and renaming it to `make.exe`.

With `MinGW`, the math libraries are automatically added during compilation. There is no need to add them in the `CMakeLists.txt`. Therefore, you can remove the code lines related to the math libraries in the `CMake`.

The rest of the installation follows the instruction given for the Unix installation in Section 2.2.1.

Compiling the examples

Each example has a `build` folder containing two files: `CMakeLists.txt` and `compile.sh`. You need to make the same modifications as before:

1. Modify `compile.sh` files,

```
# cmake CMakeLists.txt -DCMAKE_BUILD_TYPE=Release
cmake CMakeLists.txt -DCMAKE_BUILD_TYPE=Release -G "MinGW Makefiles"
# make
mingw32-make
```

2. Modify `CMakeLists.txt` files.

```
# set(CMAKE_C_COMPILER /usr/bin/gcc)
set(CMAKE_C_COMPILER "C:/msys64/mingw64/bin/gcc.exe")
```

3. Modify `*.c` files.

```
mkdir(Bubble->Results->dir, 0700)
// mkdir(Bubble->Results->dir)
```

You can follow the instructions in the project's `README` to run the examples.

2.3 Running APECSS

There are several ways in which you can use the APECSS library. You can either incorporate selected features of APECSS into your own software code or you can program an interface to use APECSS as a standalone software tool.

2.3.1 The `*.apecss` options file

The `*.apecss` file is the primary way of passing options, such as the size of the bubble, the density of the liquid or the type of results you want to have written out, to APECSS.

Any `*.apecss` file may contain the following sections (each terminated with `END`):

- `BUBBLE`: Information related to the bubble, such as its initial radius R_0 or the Rayleigh-Plesset model that is used to solve its dynamics.

- `GAS`: Properties and equation of state of the gas.
- `LIQUID`: Properties, type (i.e. Newtonian or viscoelastic) and equation of state of the liquid.
- `INTERFACE`: Properties of the gas-liquid interface.
- `RESULTS`: Results of the bubble dynamics and the acoustic emissions that should be written out.
- `ODESOLVER`: Parameters of the ODE solver.

The options file is read by the functions `apecss_*_readoptions()`. Any of the sections in the `*.apecss` file, and any of the options that may be defined within each section, are optional; they are read if they are present, otherwise the default values (typically set in `apecss_*_setdefaultoptions()`) or values set in the code calling it will be used. Comments can be added using `#`.

The relevant options that are available are discussed in the following chapters of this documentation in the context of the theoretical framework of APECSS. For instance, the available options used to define a certain Rayleigh-Plesset model are discussed in Section 3.1, where the theory of the implemented Rayleigh-Plesset models is described.

2.3.2 Examples

Some representative examples are available in the `$APECSS_DIR/examples` folder. Each directory contains the following:

- A `README.md` file explaining the purpose and specificities of this/these example(s).
- A `src/` folder with a file called `*_apecss.c` that acts as the standalone interface to the APECSS library. This file contains the `main()` function and any additional functionality required to simulate a specific scenario.
- A `build/` folder containing the `CMakeLists.txt` file and a shell script `compile.sh` with which this example can be compiled using the command `./compile.sh`.
- One or several `*.apecss` file(s) in which the options for a specific case are defined.
- One or several `plot_*.py` Python scripts to plot the results of the example. These Python scripts require `numpy` and `matplotlib`, and plot the results into a pdf-file.
- Some examples have a `reference-data` folder, which contains results of previous studies to quickly validate the overall correct working of APECSS and which are plotted alongside the results produced by APECSS using the Python script(s).

In the examples provided in the `$APECSS_DIR/examples` folder, the name of the `*.apecss` file is passed as an argument with the call to run APECSS, e.g. executing `./<APECSS-example> -options run.apecss` to use an options file named `run.apecss`. Note that, for most test cases, the settings and options given by the execution command and the corresponding `*.apecss` file reproduce a case previously presented in the literature, as indicated in the `README.md` files, which also serves to validate the implementation of APECSS.

You can run **all**[2] examples by executing `./run_all.sh` in the `examples` folder. This shell script compiles, runs and plots the results of the test cases. This is a great way of testing the installation of APECSS and see what sort of results can be output and how these results may be visualized. The script also compares the results computed for a respresentative example (the Rayleigh collapse with emissions) with reference results of APECSS, checking whether the installation produces correct results. Note that `run_all.sh` assumes that *Python* can be executed using the command `python3`; please adjust this if necessary.

Alternatively, if you're, for instance, specifically interested in an ultrasound-driven bubble, navigate to `$APECSS_DIR/examples/ultrasound`. There you can find different test cases, described in more detail in the accompanying `README.md`. Execute the `compile.sh` script in the `build` directory. You

[2] With the expection of the example simulating the response of a bubble cluster with 250 bubbles to a tension wave (see `$APECSS_DIR/examples/sphericalclustertensionwave`), as it requires several minutes to complete the simulation.

can now choose which test case to run, by using the corresponding execution command suggested in the `README.md`.

The example of the response of a monodisperse bubble cluster with 250 bubbles to a tension wave, see `$APECSS_DIR/examples/sphericalclustertensionwave`, in which the bubble-bubble interactions are modelled based on the quasi-acoustic assumption (see Section 4.3), provides an example of how simulations of multi-bubble systems can be parallelized using MPI.

More information about each example, how to run it and what the results might be compared to can be found in the accompanying `README.md` file.

2.4 Programming in and with APECSS

All functions are located in source files (`*.c`) that relate to parts of the code, distinguished by physical phenomena (e.g. `emissions.c`), fluid type (e.g. `liquid.c`) or computational operations (e.g. `results.c`).

All declarations and definitions are located in the header file `include/apecss.h`.

2.4.1 Macros

Macros are used as shortcuts to define frequently-used constants (e.g. `APECSS_PI`), for frequently-used computational operations (e.g. `APECSS_MAX`) and for computational operations that depend on the chosen machine precision (e.g. `APECSS_SQRT`). Furthermore, options related to different numerical models are represented by logically named flags.

2.4.1.1 Macros related to machine precision

APECSS can be used with different floating point precisions: double precision (default) and long double precision (`APECSS_PRECISION_LONGDOUBLE`).

Based on the chosen precision, `APECSS_FLOAT` is defined as the standard floating point type. In addition, the following precision-dependent computational operations are defined based on the chosen floating point precision:

- `APECSS_ABS(a)`: Absolute value of a.
- `APECSS_CEIL(a)`: a rounded to the nearest integer larger than a.
- `APECSS_COS(a)`: Cosine of a.
- `APECSS_EPS`: Returns a value that is close to machine epsilon.
- `APECSS_EXP(a)`: e to the power a.
- `APECSS_LOG(a)`: Natural logarithm of a.
- `APECSS_POW(a,b)`: a to the power b.
- `APECSS_SIN(a)`: Sine of a.
- `APECSS_SMALL`: Returns a number significantly smaller than machine epsilon.
- `APECSS_SQRT(a)`: Square root of a.
- `APECSS_STRINGTOFLOAT(a)`: Conversion of string a to a float.

To ensure compatibility for different floating point precisions, it is paramount to use the standard floating point type `APECSS_FLOAT` and the operator definitions given above consistently throughout APECSS.

2.4.1.2 Computational operations and predefined constants

Macros that provide a shortcut to frequently-used computational operations are:

- `APECSS_POW2(a)`: Returns a^2.
- `APECSS_POW3(a)`: Returns a^3.
- `APECSS_POW4(a)`: Returns a^4.
- `APECSS_MAX(a,b)`: Returns the maximum of a and b.
- `APECSS_MAX3(a,b,c)`: Returns the maximum of a, b and c.
- `APECSS_MIN(a,b)`: Returns the minimum of a and b.
- `APECSS_MIN3(a,b,c)`: Returns the minimum of a, b and c.

Macros that provide a shortcut to frequently-used constants are:

- `APECSS_PI`: Returns $\pi = 3.14159265358979323846264338327950288419 72$.
- `APECSS_E`: Returns e = 2.71828182845904523536028747135266249775 72.
- `APECSS_ONETHIRD`: Returns 1/3.
- `APECSS_ONESIXTH`: Returns 1/6.
- `APECSS_AVOGADRO`: Returns the Avogadro constant, $N_\mathrm{A} = 6.02214076 \times 10^{23}\,\mathrm{mol}^{-1}$.
- `APECSS_LN_OF_2`: Returns the natural logarithm of 2, $\ln(2) = 0.693147180559945309$.
- `APECSS_LN_OF_10`: Returns the natural logarithm of 10, $\ln(10) = 2.302585092994045684$.
- `APECSS_LARGE`: Returns a large number, defined as 10^{15}.

2.4.1.3 Flags for model options

All model options are represented by human-readable flags.

If explicitly indicated as such, these flags are defined in such a way (with integer values being a multiple of 2), that a bit-wise comparison can be performed. Bit-wise comparison may be used for options that are checked frequently and for options that can have several building blocks.

Options of the embedded Runge-Kutta scheme of Dormand and Prince [7] used to discretize the governing ODEs:

- `APECSS_RK54_7M`: RK5(4)7M (minimum truncation) coefficients of Dormand and Prince [7]
- `APECSS_RK54_7S`: RK5(4)7S (stability optimized) coefficients of Dormand and Prince [7]

Rayleigh-Plesset schemes:

- `APECSS_BUBBLEMODEL_RP`: Standard Rayleigh-Plesset model, Eq. (3.1).
- `APECSS_BUBBLEMODEL_RP_ACOUSTICRADIATION`: Rayleigh-Plesset model with acoustic radiation damping, Eq. (3.2).
- `APECSS_BUBBLEMODEL_KELLERMIKSIS`: Keller-Miksis model, Eq. (3.3).
- `APECSS_BUBBLEMODEL_GILMORE`: Gilmore model, Eq. (3.4).

Equation of state of the gas:

- `APECSS_GAS_IG`: Ideal gas EoS.
- `APECSS_GAS_HC`: Ideal gas EoS with van-der-Waals hardcore.
- `APECSS_GAS_NASG`: Noble-Abel-stiffened-gas EoS.

Equation of state of the liquid:

- `APECSS_LIQUID_TAIT`: Tait EoS.
- `APECSS_LIQUID_NASG`: Noble-Abel-stiffened-gas EoS.

Viscoelasticity of the liquid:

- `APECSS_LIQUID_NEWTONIAN`: Newtonian liquid.
- `APECSS_LIQUID_KELVINVOIGT`: Kelvin-Voigt solid.
- `APECSS_LIQUID_ZENER`: Zener solid (standard linear solid model).
- `APECSS_LIQUID_OLDROYDB`: Oldroyd-B liquid.

Lipid monolayer coating of the gas-liquid interface (bit-wise):

- `APECSS_LIPIDCOATING_NONE`: No lipid monolayer coating.
- `APECSS_LIPIDCOATING_MARMOTTANT`: Lipid monolayer coating described by the model of Marmottant *et al.* [21].
- `APECSS_LIPIDCOATING_GOMPERTZFUNCTION`: Redefine the Marmottant model with a Gompertz function [10].

Acoustic excitation applied to the bubble:

- `APECSS_EXCITATION_NONE`: No external excitation.
- `APECSS_EXCITATION_SIN`: Sinusoidal excitation.

Model to compute the acoustic emissions of the bubble (bit-wise):

- `APECSS_EMISSION_NONE`: Emissions are not modelled.
- `APECSS_EMISSION_INCOMPRESSIBLE`: Emissions are assumed to occur in an incompressible fluid.
- `APECSS_EMISSION_FINITE_SPEED_INCOMPRESSIBLE`: Emissions are assumed to occur in an incompressible fluid, but the finite propagation speed given by the speed of sound is taken into account.
- `APECSS_EMISSION_QUASIACOUSTIC`: Emissions are modelled under the quasi-acoustic assumption of Trilling [25] and Gilmore [9].
- `APECSS_EMISSION_KIRKWOODBETHE`: A model based on the Kirkwood-Bethe hypothesis (EKB, GFC, HPE) is used.
- `APECSS_EMISSION_EV`: Emissions are modelled using the explicit expression based on the Kirkwood-Bethe hypothesis.
- `APECSS_EMISSION_TIV`: Emissions are modelled using the temporally-integrated velocity, based on the Kirkwood-Bethe hypothesis, of Hickling and Plesset [11].

2.4.1.4 Others

Macros for the output of certain results produced by APECSS:

- `APECSS_RESULTS_DISCARD`: Discard the results and do not write results to file.
- `APECSS_RESULTS_WRITE`: Write all data to file in one go.
- `APECSS_RESULTS_APPEND`: Append the results file with a new batch of results.

Other predefined macros are used to define the length of strings and arrays, as well as to help with debugging:

- `APECSS_DATA_ALLOC_INCREMENT`: The increment for dynamic re-allocation of arrays.
- `APECSS_STRINGLENGTH`: The standard length of a string.
- `APECSS_STRINGLENGTH_SPRINTF`: The standard length of a string to be written out in the terminal.
- `APECSS_STRINGLENGTH_SPRINTF_LONG`: The standard length of a long string to be written out in the terminal.
- `APECSS_WHERE`: Outputs in the terminal the file name and line number where the macro is called.
- `APECSS_WHERE_INT(a)`: Outputs in the terminal the file name and line number where the macro is called, plus the integer value a.
- `APECSS_WHERE_FLOAT(a)`: Outputs in the terminal the file name and line number where the macro is called, plus the floating point value a.

2.4.2 Structures

Structures (`struct`) are used in APECSS to group variables and functions, and to provide a modular layout of the code that enables reusing different parts of it.

The structure `APECSS_Bubble` is the central structure of APECSS as it contains all the information related to a bubble. There is, of course, no *a priori* limit on how many copies of this structure a simulation can have, for instance, a multi-bubble simulation with 100 bubbles would naturally have 100 objects of type `struct APECSS_Bubble`. Aside from key information about the bubble, such as the bubble radius, the `APECSS_Bubble` structure contains pointers to the properties of the liquid the bubble is immersed in (`struct APECSS_Liquid`), the properties of the gas the bubble contains (`struct APECSS_Gas`) as wel as the properties of its interface (`struct APECSS_Interface`). If applicable, the `APECSS_Bubble` structure also points to the structure with the information of the driving acoustic excitation (`struct APECSS_Excitation`), the structure handling the acoustic emissions (`struct APECSS_Emissions`), and the structure containing the desired results (`struct APECSS_Results`).

The (optional) structure `APECSS_Emissions` is, as the name suggests, related to the acoustic emissions of a bubble. If allocated, it contains information about how to handle the acoustic emissions, function pointers referring to the functions used to advance the acoustic emissions using a Lagrangian wave tracking approach and, very importantly, the linked list of emissions nodes (`struct APECSS_EmissionNode`) that carry the actual information of the acoustic emissions. The structure `APECSS_EmissionNode` holds the information (e.g. radial position, velocity, enthalpy, pressure) associated with a specific emission node.

The (optional) structure `APECSS_Interaction` contains additional information required to simulate bubble-bubble interactions of multi-bubble systems, such as the bubble location.

The (optional) structure `APECSS_Results` holds all the results the user may want to have written out. For performance reasons, the results are in general not written to disk on-the-fly, but are stored in arrays and dumped to disk at the end of the simulation. The `APECSS_Results` structure contains optional structures for the results of the Rayleigh-Plesset model (`struct APECSS_ResultsBubble`) and for the acoustic emissions (`APECSS_ResultsEmissions`).

2.4.3 Important functions

Whenever using APECSS, at least one `APECSS_Bubble` structure has to be available and, if multiple bubbles are part of a simulation, each bubble has to be represented by a separate `APECSS_Bubble` structure. A single `APECSS_Bubble` structure is allocated and initialized as follows:

```
struct APECSS_Bubble *Bubble = (struct APECSS_Bubble *) malloc(sizeof(struct
  APECSS_Bubble));
apecss_bubble_initializestruct(Bubble);
```

The function `apecss_bubble_initializestruct()` ensures that all pointers (to arrays, other structures and functions) that are part of the `APECSS_Bubble` structure are initialized to `NULL`. This is important because the processing of options and any checks for allocation depend on `NULL` to indicate that a given pointer in not yet allocated or not in use. Then, we wish to set default values for the bubble parameters and read the relevant options from the options file:

```
apecss_bubble_setdefaultoptions(Bubble);
apecss_bubble_readoptions(Bubble, OptionsDir);
```

While setting default values is strongly advised, it is optional. The relative path to the options file may be set as

```
char OptionsDir[APECSS_STRINGLENGTH];
sprintf(OptionsDir, "./run.apecss"); // Relative path to the options file.
```

In general, the properties of a gas (`struct APECSS_Gas`), a liquid (`struct APECSS_Liquid`) and a gas-liquid interface (`struct APECSS_Interface`), as well as the parameters for the ODE solver (`struct APECSS_NumericsODE`), have to be associated with a bubble, through the structure pointers `*Gas`, `*Liquid`, `*Interface` and `*NumericsODE` readily available in the `APECSS_Bubble` structure. In a single-bubble simulation this is obviously straightforward, we have one gas, one liquid and one interface that are associated with the bubble. In addition we have one set of solver parameters. In a multi-bubble simulation, however, we likely also have, for instance, only a single liquid (i.e. all bubbles are situated in the same body of liquid), to which, in this case, all bubbles are associated to. Generally, the user is free in defining as many gases, liquids, interfaces and sets of solver parameters as deemed necessary, the only rule is that each bubble has to be associated with a gas, a liquid, an interface and a single set of solver parameters. Once these structures holding the fluid properties and the solver parameters are allocated, default values ought to be set, the user-defined options are read from file and the bubble(s) is/are successfully associated with its/their desired fluid properties and solver parameters. For a single-bubble simulation, this may look in a general form like:

```
struct APECSS_Gas *Gas = (struct APECSS_Gas *) malloc(sizeof(struct APECSS_Gas));
struct APECSS_Liquid *Liquid = (struct APECSS_Liquid *) malloc(sizeof(struct
  APECSS_Liquid));
struct APECSS_Interface *Interface = (struct APECSS_Interface *) malloc(sizeof(
  struct APECSS_Interface));
struct APECSS_NumericsODE *NumericsODE = (struct APECSS_NumericsODE *) malloc(
  sizeof(struct APECSS_NumericsODE));

apecss_gas_setdefaultoptions(Gas);
apecss_liquid_setdefaultoptions(Liquid);
apecss_interface_setdefaultoptions(Interface);
apecss_odesolver_setdefaultoptions(NumericsODE);

apecss_gas_readoptions(Gas, OptionsDir);
apecss_liquid_readoptions(Liquid, OptionsDir);
apecss_interface_readoptions(Interface, OptionsDir);
apecss_odesolver_readoptions(NumericsODE, OptionsDir);

Bubble->Gas = Gas;
Bubble->Liquid = Liquid;
Bubble->Interface = Interface;
Bubble->NumericsODE = NumericsODE;
```

Note that opening and reading the options file from disk is a relatively expensive (we are talking about a few microseconds) operation. For some of the single-bubble examples in `$APECSS_DIR/examples` reading the options file is the most expensive operation by some margin. Therefore, if performance is of the essence, it might be worth hard coding the options instead of reading the options file.

After reading the options file, the `apecss_*_processoptions()` functions are called to process options and make the relevant modeling choices:

```
apecss_gas_processoptions(Gas);
apecss_liquid_processoptions(Liquid);
apecss_interface_processoptions(Interface);
apecss_odesolver_processoptions(NumericsODE);
apecss_bubble_processoptions(Bubble);
```

Processing the given options correctly is critical to the working of APECSS.

Now that all options have been read and processed, the bubble has to be initialized based on the given options. This includes, for instance, computing the initial gas pressure inside the bubble (if it is not specified by the user) or, if applicable, the hardcore radius of the bubble.

```
apecss_bubble_initialize(Bubble);
```

The heart of APECSS is, of course, the solver for the bubble dynamics. The solver is split into three separate functions that (i) initialize the solver, (ii) run the solver and (iii) wrap up (i.e. finalize) the solver:

```
apecss_bubble_solver_initialize(Bubble);
apecss_bubble_solver_run(tend, Bubble);
apecss_bubble_solver_finalize(Bubble);
```

The initialization of the solver with `apecss_bubble_solver_initialize()` makes sure all counters, the solution error variable and, if applicable, result variables are initialized correctly. The function `apecss_bubble_solver_run()` contains the time-stepping procedure that executes the solver until a specified end time `tend`, given as the first argument of the function call. In the examples found in `$APECSS_DIR/examples`, the function `apecss_bubble_solver_run()` is only called once with `tend = Bubble->tEnd`, i.e. the end of the simulation. However, the user is free to call the function `apecss_bubble_solver_run()` an arbitrary number of times, with any meaningful end time. This facilitates coupling APECSS to other numerical software frameworks, where `tend` then could for instance be the end of the next time step of a fluid dynamics solver. As an example simply chopping the simulation up into five equal-sized parts would look like:

```
apecss_bubble_solver_initialize(Bubble);
apecss_bubble_solver_run(0.2 * tend, Bubble);
apecss_bubble_solver_run(0.4 * tend, Bubble);
apecss_bubble_solver_run(0.6 * tend, Bubble);
apecss_bubble_solver_run(0.8 * tend, Bubble);
apecss_bubble_solver_run(tend, Bubble);
apecss_bubble_solver_finalize(Bubble);
```

A solver run is ended with the function `apecss_bubble_solver_finalize()` where, for instance, the arrays and linked list of the acoustic emissions are freed.

2.4.4 Output

By default, APECSS does not write out any output, neither to the terminal nor to a file.

Basic information about the version and compile options of APECSS can be written out in the terminal using the function `apecss_infoscreen()` and any user-defined message by passing the desired string to the function `apecss_writeonscreen()`, as demonstrated in many of the available examples.

Various simulation results associated with the bubble dynamics and the acoustic emissions can be written to file by passing the appropriate options to APECSS (see Sections 3.6 and 4.6 for more details). In the interest of performance (writing data to file is very expensive), the results are stored in arrays and only written to file when the appropriate `apecss_results_*_write()` function is called, at the end of the simulation or at any appropriate point during the simulation, as shown in the examples. Exporting the results associated with the radial bubble dynamics using the function `apecss_results_rayleighplesset_write()` and the recording of the acoustic emissions at predefined spatial locations using the function `apecss_results_emissionsspace_write()` additionally requires the user to tell APECSS how the data should be handled, using the macros described in Section 2.4.1.4. Either the appropriate file is appended (`APECSS_RESULTS_APPEND`), which makes sense if the results should be written out multiple times during a simulation, or the data is written to file in one go (`APECSS_RESULTS_WRITE`), which should be used if the results are written out once at the end of the simulation. In both cases, the file is created automatically if it does not exist yet. Alternatively, the option `APECSS_RESULTS_DISCARD` deletes the results without writing them to file, which can be useful if the results are used to collect, for instance, some statistics during the simulations but should not be stored in a file.

2.4.5 The void data pointer

Additional data associated with a bubble, an emission node, a gas, a liquid or an interface might be needed for more complex simulations, for instance neighbor information when multiple bubbles interact with each other or the thermal conductivity and heat capacity of the gas inside the bubble when heat transfer is considered. In APECSS, to retain flexibility and avoid unnecessary overhead,

the structures `struct APECSS_Bubble`, `struct APECSS_EmissionNode`, `struct APECSS_Gas`, `struct APECSS_Liquid` and `struct APECSS_Interface` contain a `void` pointer called `user_data` specifically for the purpose of associating additional data with those structures. This void pointer is not associated with a specific data type, but rather points to some memory location, i.e. a memory address.

A typical use of this void pointer would be to generate a structure that contains all additional data and point the void pointer to the address of this structure. For instance, the void pointer of the bubble structure is used in such a way in the example `$APECSS_DIR/examples/laserinducedcavitation`. The structure

```
struct LIC
{
  APECSS_FLOAT tauL, Rnbd, Rnc1, Rnc2, tmax1, tmax2;
};
```

is allocated

```
struct LIC *lic_data = (struct LIC *) malloc(sizeof(struct LIC));
```

and associated with the void pointer

```
Bubble->user_data = lic_data;
```

The data stored in this structure can then be used or read by again assigning the correct data type

```
struct LIC *lic_data = Bubble->user_data;
```

and used as

```
tmax = lic_data->tmax1;
```

At the end of the simulation, when the data is no longer needed, the memory occupied by the structure is freed

```
free(lic_data);
```

The example `$APECSS_DIR/examples/gastemperature` uses both the void pointers associated with the bubble and the gas.

2.4.6 A word on function pointers

APECSS uses function pointers extensively. Function pointers are an elegant means in `C` to add complexity and functionality yet still retain a slim code, avoid redundant code and, if nothing else, avoid a large number of costly conditional statements. However, function pointers can quickly make a code unreadable and obfuscate what is actually happening, if they are used without care. In order to keep the use of function pointers in APECSS transparent, the adopted convention is that all function pointers are set in `apecss_*_processoptions()` functions, e.g. `apecss_gas_processoptions()`.

2.4.7 Code formatting

To ensure a consistent formatting, please use a *clang* formatter that formats the file automatically upon saving. The file defining the formatting of the APECSS source code (`.clang-format`) is part of the repository. A *clang* formatter (supported by most IDEs and editors) should be used for contributions to APECSS. The formatter should recognize this `.clang-format` file automatically.

2.5 Units

APECSS assumes SI units or any appropriate combination of SI units at all times, e.g. when reading user-defined options, in all internal computations and when outputting data. To avoid any misunderstanding, the SI base units are the following: time in seconds [s], length in meter [m], mass in kilogram

[kg], temperature in Kelvin [K], electric current in Ampere [A], amount of substance in mole [mol], luminosity in candela [cd].

2.6 Automated testing

Github *workflows* are run automatically to test the functionality of APECSS everytime a change is *pushed* to the `main` branch of APECSS or if a *pull request* is opened. This includes a `Build test` and a `Run test`, the results of which are displayed in the rendered `README.md` file of the Github repository.

The `Build test` checks whether the APECSS library compiles correctly on Linux and MacOS operating systems. The `Run test`, also conducted on both Linux and MacOS operating systems, is more comprehensive, in that it first compiles the APECSS library, then compiles and runs each example. Note that a successful `Run test` does not imply correct results - the `Run test` only tests the basic functionality of the code (e.g. no segmentation faults), not the correctness of the results APECSS produces.

2.7 Citing APECSS

If you use APECSS for your scientific work, please consider citing the paper introducing the features and capabilities of APECSS,

> F. Denner and S. Schenke, APECSS: A software library for cavitation bubble dynamics and acoustic emissions. *Journal of Open Source Software* 8 (2023), 5435.
> https://doi.org/10.21105/joss.05435

as well as the version of APECSS you've used for your work, for instance

> F. Denner and S. Schenke, APECSS (v1.2), *Zenodo* (2022).
> https://doi.org/10.5281/zenodo.7465050

All releases of APECSS and the corresponding DOIs can be found on the Zenodo page of APECSS.

Chapter 3

Bubble dynamics

The dynamic behaviour of the bubble is modelled with a Rayleigh-Plesset-type (RP) model, assuming spherical symmetry. This requires to choose a suitable RP-type model (Section 3.1) and define appropriate conditions for the gas (Section 3.2), the liquid (Section 3.3), the interface (Section 3.4), as well as at infinity (Section 3.5). The results that APECSS can write out based on the RP model are explained in Section 3.6.

APECSS solves all ordinary differential equations (ODEs) associated with the bubble dynamics using the embedded RK5(4) scheme of Dormand and Prince [7], whereby a fifth-order Runge-Kutta scheme is used to solve the ODEs and the corresponding fourth-order Runge-Kutta scheme is used to estimate the solution error. Based on this solution error, the time step Δt used to advance the solution of the ODEs is adapted. If the solution error of a newly computed solution in a time step does not satisfy the error tolerance, the solution is rewound and recomputed with a smaller Δt (cf. sub-iteration), adapted based on the solution error of the previous attempt.

Section	Command	Description
`BUBBLE`	`InitialRadius <float>`	The initial radius R_0 of the bubble.
	`PressureAmbient <float>`	The ambient pressure p_0.
	`InitialGasPressure <float>`	The initial gas pressure $p_{G,0}$, if different from p_0 or the corresponding Laplace pressure.
	`Dimensionality Planar`	Gas cavity in a plane one-dimensional domain (limited to Kirkwood-Bethe based models).
	`Dimensionality Cylinder`	Cylindrical bubble (limited to Kirkwood-Bethe based models).
	`Dimensionality Sphere`	Spherical bubble (default).
`ODESOLVER`	`RK 7M`	Minimum truncation (7M) coefficients of the RK5(4) scheme of Dormand and Prince [7]. This is the default.
	`RK 7S`	Stability optimized (7S) coefficients of the RK5(4) scheme of Dormand and Prince [7].
	`Tolerance <float>`	The desired solution tolerance.
	`MinTimeStep <float>`	Minimum time step Δt.
	`MaxTimeStep <float>`	Maximum time step Δt.
	`MaxSubIterations <float>`	Maximum number of sub-iterations in a given time step.

3.1 Rayleigh-Plesset models

APECSS offers four RP-type models to simulate pressure-driven bubble dynamics: the standard Rayleigh-Plesset model without and with acoustic radiation damping, the Keller-Miksis model and the Gilmore model.

Section	Command	Description
`BUBBLE`	`RPModel RP`	Standard Rayleigh-Plesset model, Eq. (3.1). This is the default.
	`RPModel RPAR`	Rayleigh-Plesset model including acoustic radiation damping, Eq. (3.2).
	`RPModel KM`	Keller-Miksis model, Eq. (3.3).
	`RPModel Gilmore`	Gilmore model, Eq. (3.4).

The standard Rayleigh-Plesset (RP) model is given as [19]

$$R\ddot{R} + \frac{3}{2}\dot{R}^2 = \frac{p_\mathrm{L} - p_\infty}{\rho_{\ell,\mathrm{ref}}}, \tag{3.1}$$

where R is the bubble radius, p_L is the pressure of the liquid at the bubble wall, p_∞ is the pressure of the liquid in the far field, p_G is the pressure of the gas inside the bubble and $\rho_{\ell,\mathrm{ref}}$ is the *constant* density of the liquid.

To incorporate acoustic radiation in the liquid and the associated damping, the modified Rayleigh-Plesset model is given as [1]

$$R\ddot{R} + \frac{3}{2}\dot{R}^2 = \frac{p_\mathrm{L} - p_\infty}{\rho_{\ell,\mathrm{ref}}} + \frac{R\,\dot{p}_\mathrm{G}}{\rho_{\ell,\mathrm{ref}}\,c_{\ell,\mathrm{ref}}}, \tag{3.2}$$

where $c_{\ell,\mathrm{ref}}$ is the *constant* reference speed of sound of the liquid. The last term on the right-hand side accounts for acoustic radiation in the liquid. This modified RP model is frequently used to simulate medical ultrasound applications [26] as well as sonoluminescence [1]. It follows directly from the Keller-Miksis model, Eq. (3.3), which incorporates the compressibility of the liquid, by assuming the Mach number of the bubble wall is vanishingly small, $M_\ell = \dot{R}/c_{\ell,\mathrm{ref}} \simeq 1$. Eq. (3.2) is, consequently, only valid for small Mach numbers $M_\ell \ll 1$ [22, 24].

The Keller-Miksis model [16, 24], which incorporates the compressibility of the liquid to first order, is given as

$$\left(1 - \frac{\dot{R}}{c_{\ell,\mathrm{ref}}}\right) R\ddot{R} + \frac{3}{2}\left(1 - \frac{\dot{R}}{3\,c_{\ell,\mathrm{ref}}}\right)\dot{R}^2 = \left(1 + \frac{\dot{R}}{c_{\ell,\mathrm{ref}}}\right)\frac{p_\mathrm{L} - p_\infty}{\rho_{\ell,\mathrm{ref}}} + R\,\frac{\dot{p}_\mathrm{L} - \dot{p}_\infty}{\rho_{\ell,\mathrm{ref}}\,c_{\ell,\mathrm{ref}}}, \tag{3.3}$$

where $c_{\ell,\mathrm{ref}}$ is the speed of sound of the liquid. Both $\rho_{\ell,\mathrm{ref}}$ and $c_{\ell,\mathrm{ref}}$ are assumed to be constant, limiting the Keller-Miksis model to moderate liquid pressures ($p_\mathrm{L} \lesssim 10^8\,\mathrm{Pa}$).

Based on the Kirkwood-Bethe hypothesis [5, 17], Gilmore [9] derived a second-order ordinary differential equation describing the radial dynamics of a bubble in a compressible liquid,

$$\left(1 - \frac{\dot{R}}{c_\mathrm{L}}\right) R\ddot{R} + \frac{3}{2}\left(1 - \frac{\dot{R}}{3c_\mathrm{L}}\right)\dot{R}^2 = \left(1 + \frac{\dot{R}}{c_\mathrm{L}}\right) H + \left(1 - \frac{\dot{R}}{c_\mathrm{L}}\right)\frac{R\dot{H}}{c_\mathrm{L}}, \tag{3.4}$$

where c_L is the speed of sound of the liquid at the bubble wall, $H = h_\mathrm{L} - h_\infty$ is the enthalpy difference between the bubble wall and infinity, and $\dot{H} = \dot{h}_\mathrm{L} - \dot{h}_\infty$ is the derivative of H. The enthalpy h and the speed of sound c are defined by an appropriate equation of state as a function of pressure, with $h_\mathrm{L} = h(p_\mathrm{L})$, $h_\infty = h(p_\infty)$ and $c_\mathrm{L} = c(p_\mathrm{L})$, as detailed in 3.3. The Gilmore model is also available in a

more general formulation,

$$\left(1-\frac{\dot{R}}{c_\mathrm{L}}\right)R\ddot{R}+\frac{3}{4}\alpha\left(1-\frac{\dot{R}}{3c_\mathrm{L}}\right)\dot{R}^2=\frac{\alpha}{2}\left(1+\frac{\dot{R}}{c_\mathrm{L}}\right)H+\left(1-\frac{\dot{R}}{c_\mathrm{L}}\right)\frac{R\dot{H}}{c_\mathrm{L}}, \tag{3.5}$$

that is applicable to planar ($\alpha = 0$), cylindrical ($\alpha = 1$) and spherical ($\alpha = 2$) bubbles – see [5] for more detailed information.

3.2 The gas

In APECSS, every bubble contains a gas, which requires to select an appropriate equation of state and define meaningful properties.

Section	Command	Description
`GAS`	`EoS IG`	Ideal gas equation of state. This is the default.
	`EoS HC`	Ideal gas equation of state with a van-der-Waals hard core.
	`EoS NASG`	Noble-Abel-stiffened-gas equation of state.
	`PolytropicExponent <float>`	Polytropic exponent Γ_g.
	`ReferencePressure <float>`	Reference pressure $p_\mathrm{g,ref}$.
	`ReferenceDensity <float>`	Reference density $\rho_\mathrm{g,ref}$.
	`CoVolume <float>`	Co-volume b_g.
	`TaitPressureConst <float>`	Pressure constant B_g.
	`MolecularWeight <float>`	Molecular weight $\mathcal{M}_\mathrm{g}$ of the gas.
	`MolecularDiameter <float>`	Molecular kinematic diameter $\mathcal{D}_\mathrm{g}$ of the gas.
`BUBBLE`	`HardcoreRadius <float>`	Radius of the van-der-Waals hard core, r_hc.

For a spherical bubble, using the ideal gas EoS, the pressure and its derivative are given as

$$p_\mathrm{G}=p_\mathrm{G,ref}\left(\frac{R_0}{R}\right)^{3\Gamma_\mathrm{g}} \tag{3.6}$$

$$\dot{p}_\mathrm{G}=-3\,\frac{p_\mathrm{G}\,\Gamma_\mathrm{g}\,\dot{R}}{R}, \tag{3.7}$$

$$ \tag{3.8}$$

including a van-der-Waals hardcore (HC) in the ideal gas model, the pressure and its derivative follow as

$$p_\mathrm{G}=p_\mathrm{G,ref}\left(\frac{R_0^3-r_\mathrm{hc}^3}{R^3-r_\mathrm{hc}^3}\right)^{\Gamma_\mathrm{g}} \tag{3.9}$$

$$\dot{p}_\mathrm{G}=-3\,\frac{p_\mathrm{G}\,\Gamma_\mathrm{g}\,R^2\,\dot{R}}{R^3-r_\mathrm{hc}^3}, \tag{3.10}$$

and using the Noble-Abel-stiffened-gas (NASG) EoS, the pressure and its derivative are [4]

$$p_\mathrm{G}=(p_\mathrm{G,ref}+B_\mathrm{g})\left[\frac{\rho_\mathrm{g}\,(1-b_\mathrm{g}\,\rho_\mathrm{g,ref})}{\rho_\mathrm{g,ref}\,(1-b_\mathrm{g}\,\rho_\mathrm{G})}\right]^{\Gamma_\mathrm{g}}-B_\mathrm{g} \tag{3.11}$$

$$\dot{p}_\mathrm{G}=\frac{\dot{\rho}_\mathrm{G}\,\Gamma_\mathrm{g}\,(p_\mathrm{G}+B_\mathrm{g})}{\rho_\mathrm{G}\,(1-b_\mathrm{g}\,\rho_\mathrm{G})}, \tag{3.12}$$

where Γ_g is the polytropic exponent, r_hc is the hardcore radius, b_g is the co-volume and B_g is a pressure constant. Assuming mass conservation, the gas density and its derivative are given by

$$\rho_\mathrm{G} = \rho_\mathrm{g,ref} \left(\frac{R_0}{R} \right)^3 \tag{3.13}$$

$$\dot{\rho}_\mathrm{G} = -3\,\rho_\mathrm{G}\,\frac{\dot{R}}{R}. \tag{3.14}$$

The hardcore radius r_hc and the co-volume b_g are set by default to -1. If the `HC` or `NASG` model is chosen, the user has to pass values for the hardcore radius r_hc or the co-volume b_g, respectively. Alternatively, for a spherical bubble, the molecular weight $\mathcal{M}_\mathrm{g}$ and the molecular kinematic diameter $\mathcal{D}_\mathrm{g}$ of the gas may be defined instead of r_hc or b_g; APECSS then computes the correct co-volume b_g or, based on the bubble size, hardcore radius r_hc. Assuming the molecular weight $\mathcal{M}_\mathrm{g}$, the bubble contains

$$N_\mathrm{G} = N_\mathrm{A}\,\frac{\rho_\mathrm{G,0} V_0}{\mathcal{M}_\mathrm{g}} \tag{3.15}$$

molecules, where N_A is the Avogadro constant (see macro `APECSS_AVOGADRO`), $\rho_\mathrm{G,0}$ is the initial gas density and V_0 is the initial bubble volume. As per the molecular kinetic diameter $\mathcal{D}_\mathrm{g}$ of the gas molecules, the volume of each molecule is

$$V_\mathrm{mol} = \frac{\pi}{6}\,\mathcal{D}_\mathrm{g}^3. \tag{3.16}$$

The van-der-Waals hardcore radius is then readily defined as

$$r_\mathrm{hc} = \sqrt[3]{\frac{3}{4\pi} f_\mathrm{mol}\, V_\mathrm{mol}\, N_\mathrm{G}} \tag{3.17}$$

and the co-volume of the gas is given as

$$b_\mathrm{g} = f_\mathrm{mol}\, N_\mathrm{A}\,\frac{V_\mathrm{mol}}{\mathcal{M}_\mathrm{g}}. \tag{3.18}$$

The semi-empirical constant f_mol is based on the repulsive forces acting between the molecules [18], and is typically taken to be $f_\mathrm{mol} = 4$.

3.3 The liquid

In the same way that every bubble contains a gas, in APECSS every bubble is surrounded by a liquid, which requires to select an appropriate equation of state and liquid type, as well as define meaningful properties.

Section	Command	Description
LIQUID	EoS Tait	The Tait EoS is applied to the liquid. Only relevant for the Gilmore model and acoustic emissions based on the Kirkwood-Bethe hypothesis.
	EoS NASG	The Noble-Abel-stiffened-gas EoS is applied to the liquid. Only relevant for the Gilmore model and acoustic emissions based on the Kirkwood-Bethe hypothesis.
	LiquidType Newtonian	Newtonian liquid. This is the default.
	LiquidType KelvinVoigt	Kelvin-Voigt solid.
	LiquidType Zener	Zener solid.
	LiquidType OldroydB	Oldroyd-B (or upper-convected Maxwell) liquid.
	LiquidType PowerLaw	Power-law liquid.
	PolytropicExponent <float>	Polytropic exponent Γ_ℓ.
	ReferencePressure <float>	Reference pressure $p_{\ell,\mathrm{ref}}$.
	ReferenceDensity <float>	Reference density $\rho_{\ell,\mathrm{ref}}$.
	ReferenceSpeedofSound <float>	Reference speed of sound $\rho_{\ell,\mathrm{ref}}$.
	CoVolume <float>	Co-volume b_ℓ.
	TaitPressureConst <float>	Pressure constant B_ℓ.
	Viscosity <float>	Newtonian viscosity μ_ℓ.
	PolymerViscosity <float>	Polymer viscosity η_ℓ associated with viscoelasticity.
	ShearModulus <float>	Shear modulus G_ℓ associated with viscoelasticity.
	RelaxationTime <float>	Relaxation time λ_ℓ associated with viscoelasticity.
	PowerLawExponent <float>	Exponent n of a power-law liquid.
	PowerLawConsistencyCoeff <float>	Consistency coefficient k of a power-law liquid.

The pressure of a Newtonian liquid at the wall of a spherical bubble is given as

$$p_\mathrm{L} = p_\mathrm{G} - \frac{2\sigma}{R} - 4\,\mu_\ell \frac{\dot{R}}{R}, \tag{3.19}$$

where p_G is the gas pressure, see Section 3.2, σ is the surface tension coefficient of the interface, see Section 3.4, and μ_ℓ is the liquid (Newtonian) viscosity. The derivative of Eq. (3.19) follows as

$$\dot{p}_\mathrm{L} = \dot{p}_\mathrm{G} + \frac{2\sigma\dot{R}}{R^2} + 4\,\mu_\ell \left(\frac{\dot{R}^2}{R^2} - \frac{\ddot{R}}{R} \right). \tag{3.20}$$

The pressure of a power-law liquid at the wall of a spherical bubble is given as [15]

$$p_\mathrm{L} = p_\mathrm{G} - \frac{2\sigma}{R} - \frac{4\,k^\star}{n} |\dot{R}|^{n-1} \frac{\dot{R}}{R^n}, \tag{3.21}$$

where p_G is the gas pressure, see Section 3.2, σ is the surface tension coefficient of the interface, see Section 3.4, $k^\star = k(2\sqrt{3})^{n-1}$ is an extended consistency coefficient [15], and n is the power-law exponent. The derivative of Eq. (3.21) follows as

$$\dot{p}_\mathrm{L} = \dot{p}_\mathrm{G} + \frac{2\sigma\dot{R}}{R^2} + 4\,k^\star |\dot{R}|^{n-3}\dot{R}^2 \left(\frac{\dot{R}^2}{R^2} - \frac{\ddot{R}}{R} \right). \tag{3.22}$$

The power-law liquid, described by Eqs. (3.21) and (3.22), reduces to a Newtonian liquid, described by Eqs. (3.19) and (3.20), for $n = 1$ and $k = \mu_\ell$.

Table 3.1: Model constants of the NASG EoS for water proposed in recent years.

	Le Métayer and Saurel [20]	Chandran and Salih [2]	Denner and Schenke [6]
Γ_ℓ	1.19	1.19	1.11
B_ℓ [Pa]	7028×10^5	6218×10^5	6480×10^5
b_ℓ [m^3/kg]	6.61×10^{-4}	6.72×10^{-4}	6.80×10^{-4}
$\rho_{\ell,\mathrm{ref}}$ [kg/m^3]	957.7	997.0	997.0
$p_{\ell,\mathrm{ref}}$ [Pa]	1.0453×10^5	10^5	10^5

3.3.1 Equation of state

For the Gilmore model (3.4) and the acoustic emissions based on the Kirkwood-Bethe hypothesis (see Section 4.5), an equation of state (EoS) for the liquid has to be defined. Two EoS are currently available in APECSS: the Tait EoS and the NASG EoS.

Since the seminal work of Gilmore [9], the Tait EoS is traditionally used to describe the properties of the liquid in Eq. (3.4). The Tait EoS defines the density ρ, enthalpy h and speed of sound c as

$$\rho = \rho_{\ell,\mathrm{ref}} \left(\frac{p + B_\ell}{p_{\ell,\mathrm{ref}} + B_\ell} \right)^{\frac{1}{\Gamma_\ell}} \tag{3.23}$$

$$h = \frac{\Gamma_\ell}{\Gamma_\ell - 1} \frac{p + B_\ell}{\rho} \tag{3.24}$$

$$c = \sqrt{(\Gamma_\ell - 1)\, h}, \tag{3.25}$$

respectively, where B_ℓ is a pressure constant, Γ_ℓ is the polytropic exponent, $p_{\ell,\mathrm{ref}}$ is the reference pressure and $\rho_{\ell,\mathrm{ref}}$ is the reference density. For water, typical values are $\Gamma_\ell = 7.15$, $B_\ell = 3.046 \times 10^8$ Pa, $\rho_{\ell,\mathrm{ref}} = 997\,\mathrm{kg/m^3}$ and $p_{\ell,\mathrm{ref}} = 10^5$ Pa.

Using the Noble-Abel stiffened-gas (NASG) EoS [20] instead of the Tait EoS, the fluid properties are defined as [4]

$$\rho = \frac{K_\ell\,(p + B_\ell)^{\frac{1}{\Gamma_\ell}}}{1 + b_\ell\, K_\ell\,(p + B_\ell)^{\frac{1}{\Gamma_\ell}}} \tag{3.26}$$

$$h = \frac{\Gamma_\ell}{\Gamma_\ell - 1} \frac{p + B_\ell}{\rho} - \frac{\Gamma_\ell\, b_\ell}{\Gamma_\ell - 1} (p + B_\ell) + b_\ell\, p \tag{3.27}$$

$$c = \sqrt{\Gamma_\ell \frac{(p + B_\ell)}{\rho - b_\ell \rho^2}}, \tag{3.28}$$

with $K_\ell = \rho_{\ell,\mathrm{ref}} / [(p_{\ell,\mathrm{ref}} + B_\ell)^{1/\Gamma_\ell}\,(1 - b_\ell \rho_{\ell,\mathrm{ref}})]$ describing a constant reference state, and where b_ℓ is the co-volume of the liquid molecules. The NASG EoS reduces to the Tait EoS for $b_\ell = 0$. Appropriate properties for water are listed in Table 3.1.

3.3.2 Viscoelasticity

Currently, APECSS supports three widely-used models for viscoelastic media in spherical symmetry: the Kelvin-Voigt model, the Zener model and the Oldroyd-B model. While the Kelvin-Voigt model merely yields an additional term in the expression for the liquid pressure at the bubble wall, the Zener and Oldroyd-B models each require to solve two additional ODEs.

3.3.2.1 Kelvin-Voigt model

To model a Kelvin-Voigt medium, the elasticity of the medium is described by the additional term

$$\frac{4}{3} G_\ell \frac{R^3 - R_0^3}{R^3}, \tag{3.29}$$

which contributes to Eq. (3.19) to obtain

$$p_\mathrm{L} = p_\mathrm{G} - \frac{2\sigma}{R} - 4\,\mu_\ell \frac{\dot{R}}{R} - \frac{4}{3} G_\ell \frac{R^3 - R_0^3}{R^3}, \tag{3.30}$$

where G_ℓ is the elastic shear modulus. The derivative of the liquid pressure at the bubble wall is then given as

$$\dot{p}_\mathrm{L} = \dot{p}_\mathrm{G} + \frac{2\,\sigma\,\dot{R}}{R^2} + 4\,\mu_\ell \left(\frac{\dot{R}^2}{R^2} - \frac{\ddot{R}}{R} \right) - 4\,G_\ell \frac{R_0^3 \dot{R}}{R^4}. \tag{3.31}$$

3.3.2.2 Zener model

A more sophisticated viscoelastic model than the Kelvin-Voigt model is the Zener model, also known as standard linear solid model. With the Zener model, the stresses in the medium surrounding the bubble are incorporated in the liquid pressure at the bubble wall as [13]

$$p_\mathrm{L} = p_\mathrm{G} - \frac{2\sigma}{R} + 3\varsigma \tag{3.32}$$

where

$$\varsigma = \int_R^\infty \frac{\tau_{rr}(r,t)}{r}\,\mathrm{d}r \tag{3.33}$$

is an auxiliary variable associated with the rr-component of the viscous stress tensor $\boldsymbol{\tau}(r,t)$. The auxiliary stress variable is governed by

$$\lambda_\ell \dot{\varsigma} + \varsigma + \lambda_\ell \frac{\dot{R}}{R} \tau_{rr|R} = -\frac{S}{3}, \tag{3.34}$$

with

$$S = \frac{4}{3} G_\ell \left(1 - \frac{R_0^3}{R^3} \right) + 4\mu_\ell \frac{\dot{R}}{R} \tag{3.35}$$

the combined viscous and elastic contributions, where λ_ℓ is the relaxation time, G_ℓ is the shear modulus and μ_ℓ the viscosity. The stress at the bubble wall, $\tau_{rr|R}$, evolves as

$$\lambda_\ell \dot{\tau}_{rr|R} + \tau_{rr|R} = -S. \tag{3.36}$$

The question is now how to solve the ODEs for ς and τ_{rr} in such a way that we always obtain a meaningful result, even if $\lambda_\ell = 0$. In order for a customary ODE solver to handle this correctly, we rearrange Eqs. (3.34) and (3.36). Under the discrete assumption

$$\dot{\varsigma} = \frac{\varsigma_{n+1} - \varsigma_n}{\Delta t}, \tag{3.37}$$

Eq. (3.34) becomes

$$\lambda_\ell \frac{\varsigma_{n+1} - \varsigma_n}{\Delta t} + \varsigma_{n+1} + \lambda_\ell \frac{\dot{R}}{R} \tau_{rr|R} = -\frac{S}{3} \tag{3.38}$$

so that, after some further manipulation,

$$\varsigma_{n+1} = \varsigma_n + \Delta t \frac{-\frac{S}{3} - \lambda_\ell \frac{\dot{R}}{R} \tau_{rr|R,n} - \varsigma_n}{\lambda_\ell + \Delta t}. \tag{3.39}$$

Similarly, Eq. (3.36) follows as

$$\tau_{rr|R,n+1} = \tau_{rr|R,n} + \Delta t \frac{-S - \tau_{rr|R,n}}{\lambda_\ell + \Delta t}. \tag{3.40}$$

Even in the limit $\lambda_\ell = 0$, we can now obtain a meaningful answer, that is Eq. (3.39) reduces to

$$\varsigma = -\frac{S}{3}. \tag{3.41}$$

After inserting Eq. (3.41) into Eq. (3.32) we recover the Kelvin-Voigt model. For $\lambda_\ell = 0$, Eq. (3.40) becomes redundant.

3.3.2.3 Oldroyd-B model

The Oldroyd-B model is a widely used constitutive model for viscoelastic liquids. Following the work of Jiménez-Fernández and Crespo [14], the liquid pressure at the bubble wall including the Oldroyd-B model is given as

$$p_\mathrm{L} = p_\mathrm{G} - \frac{2\sigma}{R} - 4\mu_\ell \frac{\dot{R}}{R} + \mathcal{S}. \tag{3.42}$$

The polymer stress $\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2$ is split into two constitutive ODEs,

$$\lambda_\ell \dot{\mathcal{S}}_1 + \mathcal{S}_1 + 4\lambda_\ell \frac{\dot{R}}{R} \mathcal{S}_1 = -2\eta_\ell \frac{\dot{R}}{R} \tag{3.43}$$

$$\lambda_\ell \dot{\mathcal{S}}_2 + \mathcal{S}_2 + \lambda_\ell \frac{\dot{R}}{R} \mathcal{S}_2 = -2\eta_\ell \frac{\dot{R}}{R} \tag{3.44}$$

where η_ℓ is the polymer viscosity. These ODEs are reformulated in a similar manner as for the Zener model shown above, to yield

$$\mathcal{S}_{1,n+1} = \mathcal{S}_{1,n} + \Delta t \frac{-\left(4\lambda_\ell \frac{\dot{R}}{R} + 1\right) \mathcal{S}_{1,n} - 2\eta_\ell \frac{\dot{R}}{R}}{\lambda_\ell + \Delta t} \tag{3.45}$$

$$\mathcal{S}_{2,n+1} = \mathcal{S}_{2,n} + \Delta t \frac{-\left(\lambda_\ell \frac{\dot{R}}{R} + 1\right) \mathcal{S}_{2,n} - 2\eta_\ell \frac{\dot{R}}{R}}{\lambda_\ell + \Delta t}. \tag{3.46}$$

For $\lambda_\ell = 0$ Eqs. (3.45) and (3.46) still give a meaningful result and reduce to a Newtonian liquid with $\mathcal{S} = -4\eta_\ell \dot{R}/R$.

3.4 The interface

APECSS readily supports the gas-liquid interface, also often referred to as the *bubble wall*, to be either clean, for which only the surface tension coefficient has to be defined, or coated with a lipid monolayer.

Section	Command	Description
`INTERFACE`	`SurfaceTensionCoeff <float>`	Surface tension coefficient σ_c of the clean interface.
	`LipidCoatingModel None`	No lipid coating model is applied. This is the default.
	`LipidCoatingModel Marmottant`	The lipid coating model of Marmottant *et al.* [21] is applied.
	`LipidCoatingModel Gompertz-Marmottant`	The continuous variant of the lipid coating model of Marmottant proposed by Gümmer *et al.* [10] is applied.
	`SigmaInit <float>`	Initial surface tension coefficient σ_0 of the lipid coating model at R_0.
	`Elasticity <float>`	Elasticity χ of the lipid coating model.
	`DilatationalViscosity <float>`	Dilatational viscosity κ_s of the lipid coating model.

The influence of surface tension, the rheology of the lipid-monolayer coating and the viscous dissipation in the liquid is accounted for through the definition of the liquid pressure at the bubble wall, given as [21]

$$p_L = p_G - \frac{2\sigma}{R} - 4\,\mu_\ell \frac{\dot{R}}{R} - 4\,\kappa_s \frac{\dot{R}}{R^2}, \tag{3.47}$$

where σ is the surface tension coefficient, μ_ℓ is the dynamic viscosity of the liquid and κ_s is the surface dilatational viscosity of the lipid monolayer.

A clean gas-liquid interface has a surface tension coefficient of $\sigma = \sigma_c$ and a surface dilatational viscosity of $\kappa_s = 0$.

Using the model of Marmottant *et al.* [21] to describe a lipid monolayer coating of the interface, the surface tension is defined as

$$\sigma = \begin{cases} 0 & \text{for } R \le R_{buck} \\ \chi\left(\dfrac{R^2}{R_{buck}^2} - 1\right) & \text{for } R_{buck} < R < R_{rupt} \\ \sigma_c & \text{for } R \ge R_{rupt} \end{cases} \tag{3.48}$$

where χ is the surface elasticity of the lipid monolayer, the buckling radius is [23]

$$R_{buck} = \frac{R_0}{\sqrt{1 + \sigma_0/\chi}}, \tag{3.49}$$

where σ_0 is the surface tension coefficient of the lipid-coated bubble at $R = R_0$, and the rupture radius is

$$R_{rupt} = R_{buck}\sqrt{1 + \frac{\sigma_c}{\chi}}. \tag{3.50}$$

The radius-dependent surface tension coefficient of the Marmottant model [21], defined in Eq. (3.48), contains two discontinuities at $R = R_{buck}$ and $R = R_{rupt}$. These discontinuities render the Marmottant model sensitive to the applied time step when numerically solving the primary ordinary differential equation [26]. A continuously differentiable form of the Marmottant model a Gompertz function of the form $f(x) = a\,e^{-b\,e^{-cx}}$, a special case of the generalized logistics function, was proposed by Gümmer *et al.* [10]. Using this Marmottant-Gompertz model, the surface tension coefficient is defined as

$$\sigma = \sigma_c\, e^{-b\,e^{c(1-R/R_{buck})}}, \tag{3.51}$$

with

$$b = -\frac{\ln(\sigma_0/\sigma_c)}{e^{c(1-R_0/R_{buck})}} \tag{3.52}$$

and

$$c = \frac{2\chi e}{\sigma_c}\sqrt{1+\frac{\sigma_c}{2\chi}}. \tag{3.53}$$

The buckling radius R_{buck} is given by Eq. (3.49). The derivative of the surface tension coefficient follows as

$$\dot{\sigma} = \sigma\, b\, c\, e^{c(1-R/R_{buck})}\,\frac{\dot{R}}{R}. \tag{3.54}$$

The Marmottant-Gompertz model reproduces the main features of the original Marmottant model [10], but with a smooth transition between the surface tension regimes, using the same set of input parameters (σ_0, σ_c, χ) as the original Marmottant model.

3.5 Infinity

The pressure at infinity, p_∞, is used to apply a driving pressure difference for the bubble dynamics. Presently, APECSS readily supports a constant ambient pressure $p_\infty = p_0$, which may also be replaced by a pressure defined on-the-fly (e.g. provided by a fluid dynamics solver running concurrently with APECSS), or a sinusoidal excitation.

A sinusoidal excitation is defined as $p_\infty = p_0 - \Delta p_a \sin(2\pi f_a t)$, where f_a and Δp_a are the frequency and pressure amplitude of the excitation. In order to use the sinusoidal excitation, the user has to allocate the pointer `*Excitation`, in the structure `APECSS_Bubble` structure, with `struct APECSS_Excitation` and define the desired values for f_a and Δp_a. The example found in the folder `example/ultrasound/` provides a template of how to do this.

3.6 Results

The results of the bubble dynamics can be written to disk, if so desired by the user. Note that APECSS does not write any results to disk unless it is specifically asked to do so.

Section	Command	Description
`RESULTS`	`Bubble`	Results of the bubble dynamics are written to file.
	`OutputFreqRP <int>`	Results of the bubble dynamics are stored every so many time steps (default: 1).
	`OutputPath <string>`	Path to the folder where all the results should be written in to (default: `./`).
	`OutputDigits <int>`	Results are written out with as many digits (default: 6).

For the bubble dynamics, the following quantities as a function of time are written into a text file, named by the employed RP model and (if applicable) the excitation parameters used:

- Time-step number.
- Time, t.
- Time step, Δt.
- Bubble radius, R.

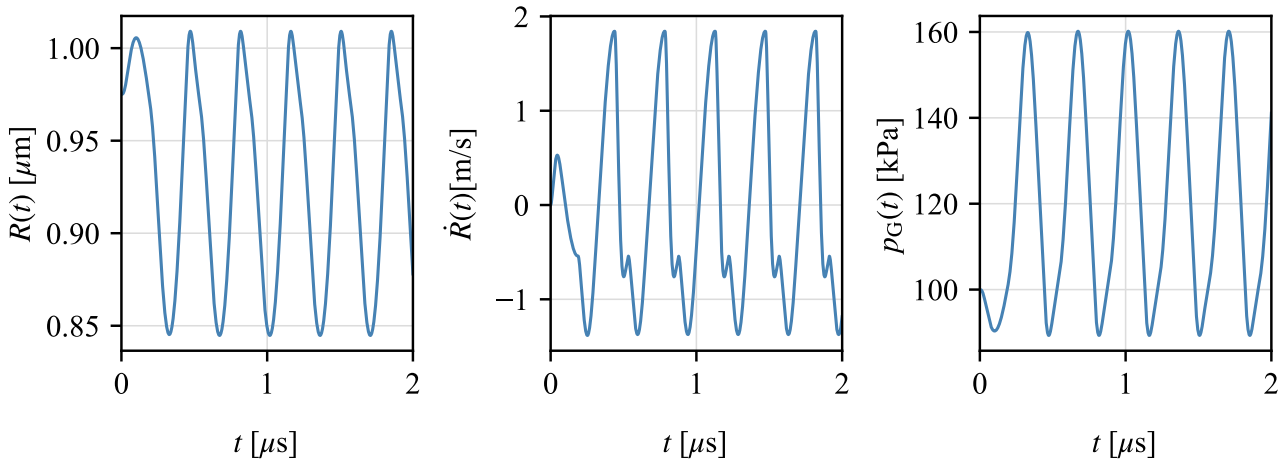


Figure 3.1: Temporal evolution of the radius $R(t)$, bubble wall velocity $\dot{R}(t)$ and gas pressure $p_\mathrm{G}(t)$ of a lipid-coated microbubble with initial radius $R_0 = 0.975\,\mu\mathrm{m}$, driven by ultrasound with a frequency of 2.9 MHz and a pressure amplitude of 130 kPa, as previously considered by Marmottant *et al.* [21]. This example can be found in `$APECSS_DIR/examples/ultrasound`, called `lipidcoated_simple`.

- Velocity of the bubble wall, $\dot{R}$.
- Pressure of the gas, p_G.
- Pressure of the liquid at the bubble wall, p_L.
- Pressure of the liquid in the far field, p_∞.
- Speed of sound of the liquid at the bubble wall, c_L, if the Gilmore model is applied.
- The result of any additional user-defined ODE solved, if applicable.

The first line of the results file(s) lists the variables that were written out and their order. For instance, Figure 3.1 shows the output of the "simple" ultrasound-driven lipid-coated microbubble examples (`$APECSS_DIR/examples/ultrasound/lipidcoated_simple.apecss`).

Chapter 4

Acoustic emissions

Modeling the acoustic emissions is a core feature of APECSS. To this end, APECSS offers different models for the acoustic emissions, assuming an incompressible liquid, a weakly-compressible liquid or a fully compressible liquid. Modeling the acoustic emissions also allows to model interaction between bubbles in multi-bubble systems. To account for a finite propagation speed, the information associated with an emitted acoustic wave is propagated along the radial coordinate axis using a Lagrangian wave tracking approach. Please refer to the work of Denner and Schenke [6] for a detailed explanation and validation of the different emission models, and the review of Denner [5] for the methodology applied to treat shock waves. Unless specifically told to do so, APECSS does not compute any acoustic emissions.

Section	Command	Description
`BUBBLE`	`Emissions IC <float>`	Computes the acoustic emissions using the standard incompressible model, Section 4.2.
	`Emissions FSIC <float>`	Computes the acoustic emissions using the finite-speed incompressible model, Section 4.4.
	`Emissions QA <float>`	Computes the acoustic emissions using the quasi-acoustic model, Section 4.3.
	`Emissions EV <float>`	Computes the acoustic emissions based on the Kirkwood-Bethe hypothesis, Section 4.5, with the explicit expression for velocity, see Eq. (4.11).
	`Emissions TIV <float>`	Computes the acoustic emissions using the model of Hickling and Plesset [11] based on the Kirkwood-Bethe hypothesis, Section 4.5, with the temporally-integrated velocity, see Eq. (4.13).
	`EmissionIntegration Euler`	Integrates the radial position and, if applicable, the velocity using an Euler scheme.
	`EmissionIntegration RK4`	Integrates the radial position and, if applicable, the velocity using a conventional fourth-order Runge-Kutta scheme. This is the default.
	`KBIterTolerance <float>`	Tolerance η for the evaluation of the pressure using a model based on the Kirkwood-Bethe hypothesis in conjunction with the NASG EoS.
	`PruneEmissions`	Prune the list of emission nodes according to a test function provided by the user. This test function must be hooked up to the function pointer `Bubble->Emissions->prune_test()`.

The floating-point value given as the final argument of the `Emissions` command defines the cut-off distance beyond which the emissions are not computed. For the standard incompressible assumption

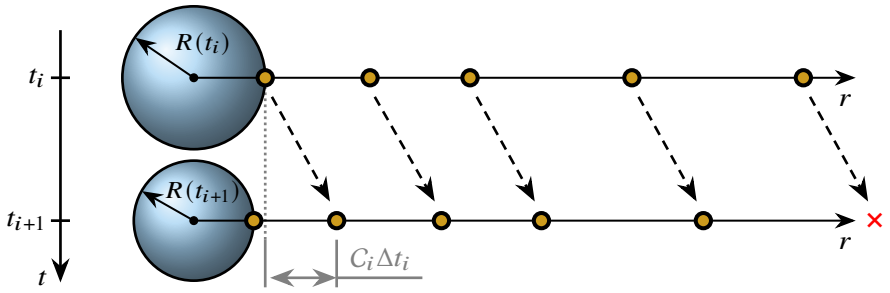


Figure 4.1: Illustration of the Lagrangian transport of the emission nodes, updated at each discrete time instance t_i. Nodes that pass a predefined maximum radial coordinate are discarded.

this value has no meaning, but a value is required as a dummy to facilitate the correct reading of the options. In addition to the emissions of a spherical bubble, the emissions computed based on the Kirkwood-Bethe hypothesis can also be used for cylindrical bubbles and planar cavities [5]. The description of the theory below, however, is limited to the spherical case.

4.1 Lagrangian wave tracking

APECSS tracks acoustic emissions using a Lagrangian wave tracking approach [6], illustrated in Figure 4.1, in which so-called *emission nodes* are propagated in the radial direction with propagation speed $\mathcal{C}$. Each emission node, represented in APECSS as a structure `struct APECSS_EmissionNode` and part of a linked list of these structures, holds the current radial coordinate $r(t)$, the flow velocity $u(r,t)$, the pressure $p(r,t)$ and, if applicable, the enthalpy $h(r,t)$, as well as the invariants $f(\tau)$ and $g(\tau)$ computed based on the solution of the RP model. The radial position of an emission node at time t is given as

$$r(t) = R(\tau) + \int_{\tau}^{t} \mathcal{C}(r,t)\,\mathrm{d}t, \tag{4.1}$$

In general, the propagation speed is defined by $\mathcal{C} = c + u$, but the actual value used depends on the chosen model.

4.2 Standard incompressible model

Assuming an incompressible liquid ($c_{\ell,\mathrm{ref}} \to \infty$) with density $\rho_{\ell,\mathrm{ref}}$, the velocity $u(r,t)$ and pressure $p(r,t)$ at a given radial position $r(t)$ are defined as [22]

$$u(r,t) = \frac{R(t)^2 \dot{R}(t)}{r^2} \tag{4.2}$$

and

$$p(r,t) = p_\infty(t) + \rho_{\ell,\mathrm{ref}} \left[\frac{R(t)^2\,\ddot{R}(t) + 2\,R(t)\,\dot{R}(t)^2}{r} - \frac{R(t)^4\,\dot{R}(t)^2}{2\,r^4} \right], \tag{4.3}$$

respectively. The assumption of an incompressible fluid is consistent with the Rayleigh-Plesset models in Eqs. (3.1) and (3.2). Note that, because $\mathcal{C} \to \infty$, these simple incompressible acoustic emissions do not use the Lagrangian wave tracking and no emission nodes are defined and processed, since pressure and velocity are defined instantaneously for all r.

4.3 Quasi-acoustic model

Assuming the liquid is compressible but accurately described by a constant density $\rho_{\ell,\mathrm{ref}}$ and constant speed of sound $c_{\ell,\mathrm{ref}}$, with $u \ll c_{\ell,\mathrm{ref}}$ and $\mathcal{C} = c_{\ell,\mathrm{ref}}$, Trilling [25] and Gilmore [9] derived the *quasi-acoustic model* for the acoustic emissions. With the quasi-acoustic model, the velocity, pressure and radial position follow as

$$u(r,t) = \frac{f(\tau)}{r(t)^2} + \frac{g(\tau)}{r(t)\, c_{\ell,\mathrm{ref}}} \tag{4.4}$$

$$p(r,t) = p_\infty(t) + \rho_{\ell,\mathrm{ref}} \left[\frac{g(\tau)}{r(t)} - \frac{u(r,t)^2}{2} \right] \tag{4.5}$$

$$r(t) \approx R(\tau) + c_{\ell,\mathrm{ref}} \sum_{i=1}^{N} \Delta t_{i-1}, \tag{4.6}$$

where $f(\tau)$ and $g(\tau)$ are invariants defined based on the result of the employed RP model as

$$f(\tau) = R(\tau)^2 \dot{R}(\tau) - \frac{R(\tau) g(\tau)}{c_{\ell,\mathrm{ref}}} \tag{4.7}$$

$$g(\tau) = R(\tau) \left[\frac{p_\mathrm{L}(\tau) - p_\infty(\tau)}{\rho_{\ell,\mathrm{ref}}} + \frac{\dot{R}(\tau)^2}{2} \right], \tag{4.8}$$

and τ is the time at which the acoustic information is emitted at the bubble wall. For $t = \tau$ with $r(t) = R(\tau)$, Eq. (4.4) reduces to $u(R,\tau) = \dot{R}(\tau)$ and Eq. (4.5) reduces to $p(R,\tau) = p_\mathrm{L}(\tau)$, thus satisfying the boundary conditions at the bubble wall.

The quasi-acoustic model is consistent in its modelling assumptions with the Keller-Miksis model, Eq. (3.3). The applicability of the quasi-acoustic model is limited to small Mach numbers, $(\dot{R}/c_0)^2 \ll 1$, as it incorporates a finite propagation speed of the acoustic emissions and the nonlinear pressure contributions resulting from the flow, but since all parts of the wave propagate with speed c_0, the quasi-acoustic model can neither describe the nonlinear distortion of acoustic waves nor the formation of shock fronts.

4.4 Finite-speed incompressible model

APECSS also supports the assumption that the liquid is incompressible but the information associated with the acoustic emissions still propagates with finite speed $\mathcal{C} = c_{\ell,\mathrm{ref}}$ using the Lagrangian wave tracking approach, with the radial location given by Eq. (4.6). By assuming the specific acoustic impedance of an incompressible liquid, $\rho_{\ell,\mathrm{ref}} c_{\ell,\mathrm{ref}} \to \infty$, the velocity reduces to

$$u(r,t) = \frac{R(\tau)^2 \dot{R}(\tau)}{r(t)^2}, \tag{4.9}$$

while $p(r,t)$ and $g(\tau)$ are given by Eq. (4.5) and Eq. (4.8), respectively. This approach, referred to in APECSS as `FSIC` or finite-speed incompressible model, recovers the time delay between emitting information at the bubble wall and this information arriving in a certain location, but treats the flow field as incompressible.

4.5 Emissions based on the Kirkwood-Bethe hypothesis

Under the Kirkwood-Bethe hypothesis [3, 17], the propagation speed along the outgoing characteristic is given as $\mathcal{C}(r,t) = c(r,t) + u(r,t)$. The ODE describing the radial position along the outgoing

characteristics is, thus, defined as

$$\left.\frac{\mathrm{d}r(t)}{\mathrm{d}t}\right|_{\mathrm{c}} = c(r,t) + u(r,t). \tag{4.10}$$

This ODE is numerically integrated using either an Euler scheme or a fourth-order Runge-Kutta (RK4) scheme, with initial condition $r(t) = R(\tau)$ for $t = \tau$, where τ is the time at which the acoustic information is emitted at the bubble wall. The time step Δt is taken to be the same as used for the integration of the model describing the bubble dynamics (see Section 3).

Two models to compute the velocity $u(r,t)$ at a given emission node are available in APECSS: (i) an explicit expression for $u(r,t)$ [6] and (ii) integrating the temporal derivative of the velocity, $\mathrm{d}u(r,t)/\mathrm{d}t$, along the outgoing characteristic, as proposed by Hickling and Plesset [11]. The assumptions used to derive these models for the acoustic emissions are consistent with the Gilmore model, Eq. (3.4).

Following a similar derivation as for the quasi-acoustic model discussed in Section 4.3, but assuming a fully-compressible liquid described by a suitable equation of state, the *explicit velocity (EV)* is given as

$$u(r,t) = \frac{f(\tau)}{r(t)^2} + \frac{g(\tau)}{r(t)\left[c(r,t) + u(r,t)\right]}, \tag{4.11}$$

with

$$f(\tau) = R(\tau)^2 \dot{R}(\tau) - \frac{R(\tau)\, g(\tau)}{c_{\mathrm{L}}(\tau) + \dot{R}(\tau)}. \tag{4.12}$$

For $t = \tau$ with $r(t) = R(\tau)$, this expression reduces to $u(R,\tau) = \dot{R}(\tau)$, thus satisfying the boundary conditions at the bubble wall.

Hickling and Plesset [11] proposed to integrate the velocity with respect to time, in APECSS referred to as *temporally-integrated velocity (TIV)*, with the temporal derivative of the velocity along the outgoing characteristic given as

$$\left.\frac{\mathrm{d}u(r,t)}{\mathrm{d}t}\right|_{\mathrm{c}} = -\frac{2c(r,t)^2 u(r,t)}{r(t)\left[c(r,t) - u(r,t)\right]} + \frac{g(\tau)}{r(t)^2}\,\frac{c(r,t) + u(r,t)}{c(r,t) - u(r,t)}. \tag{4.13}$$

This differential equation for the velocity is integrated together with the equation for $\mathrm{d}r(t)/\mathrm{d}t$, Eq. (4.10), using the initial condition $u(R,\tau) = \dot{R}(\tau)$.

Regardless of the choice of velocity model, the invariant $g(\tau)$ is defined as

$$g(\tau) = R(\tau)\left[h_{\mathrm{L}}(\tau) - h_\infty(\tau) + \frac{\dot{R}(\tau)^2}{2}\right]. \tag{4.14}$$

For a given radial position $r(t)$ and flow velocity $u(r,t)$, irrespective of which model is used to compute the velocity, the enthalpy is readily evaluated as

$$h(r,t) = h_\infty(t) + \frac{g(\tau)}{r(t)} - \frac{u(r,t)^2}{2}, \tag{4.15}$$

where h_∞ is spatially invariant and only depends on time. For $t = \tau$ with $r(t) = R(\tau)$, this expression for enthalpy reduces to $h(R,\tau) = h_{\mathrm{L}}(\tau)$, satisfying the boundary conditions at the bubble wall.

Using the Tait EoS, the pressure can be readily computed from the enthalpy defined in Eq. (4.15) by inserting Eq. (3.23) into Eq. (3.24) and rearranging to yield

$$p(r,t) = \left[\frac{(\Gamma - 1)\,\rho_0}{\Gamma\,(p_0 + B)^{1/\Gamma}}\, h(r,t)\right]^{\frac{1}{1-1/\Gamma}} - B. \tag{4.16}$$

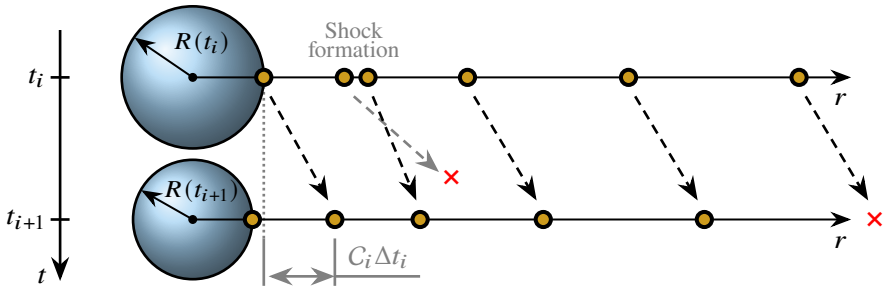


Figure 4.2: Illustration of the Lagrangian transport of the emission nodes, updated at each discrete time instance t_i. Nodes that either overtake the forerunning node, which represents the formation of a shock front, or that pass a predefined maximum radial coordinate are discarded.

Solving Eq. (4.16) is straightforward because the enthalpy $h(r,t)$ is the only variable, all other quantities are predefined fluid properties. Using the NASG EoS, pressure $p(r,t)$ follows by rearranging Eq. (3.27) as

$$p(r,t) = \frac{[\Gamma - 1]\,\rho(r,t)\,h(r,t) - [1 - b\,\rho(r,t)]\,\Gamma B}{\Gamma - b\,\rho(r,t)}. \tag{4.17}$$

Since the pressure $p(r,t)$ and the density $\rho(r,t)$ depend explicitly on each other in this formulation, Eq. (4.17) has to be solved iteratively. As a convergence criterion for the iterative approximation we use $|p_j(r,t) - p_{j-1}(r,t)| < \eta\,|p_j(r,t)|$, where j denotes the iteration counter and η is a predefined tolerance (see option `KBIterTolerance`). Preliminary tests identified a tolerance of $\eta = 10^{-4}$ to be sufficiently small.

Emission nodes with a higher pressure propagate faster than nodes with a lower pressure, which in turn leads to progressive steepening of the acoustic wave. As a result, an emission node may overtake the forerunning emission node, yielding an unphysical multivalued solution. In reality, such a multivalued solution is avoided by the formation of a shock front [8]. While treating such multivalued solutions is often done in a post-processing step, APECSS deals with multivalued solutions at run time. An emission node that overtakes its forerunning neighbor is discarded, see Fig. 4.2, and the thermodynamic values of the forerunning neighbor are reevaluated by a simple averaging procedure, as described in [5]. Thus, a unique and physically plausible solution is maintained.

4.6 Results

APECSS can write out different results based on the acoustic emissions. Note that APECSS does not write any results to disk unless it is specifically ask to do so.

The acoustic emissions can be recorded as a function of time at one or multiple radial locations (cf. `EmissionsSpace`), or the emissions are written out with respect to their radial location at one or multiple time instances (cf. `EmissionsTime`) or emission nodes (cf. `EmissionsNode`), or for selected extrema in a specified period (cf. `EmissionsMinMax`). This calls can be used multiple times to defined, for instance, multiple radial locations or time instances.

Section	Command	Description
RESULTS	`OutputPath <string>`	Path to the folder where all the results should be written in to (default: `./`).
	`OutputDigits <int>`	Results are written out with as many digits (default: 6).
	`EmissionsSpace <float>`	Defines a radial location at which the emissions in the liquid are written out as a function of time. If/while the location is in the gas phase, 0 is recorded.
	`OutputFreqEmissionsSpace <int>`	Results of the emissions at a specific radial location are stored every so many time steps (default: 1).
	`EmissionsTime <float>`	Defines a time instance at which the emission in the liquid are written out as a function of the radial coordinate.
	`EmissionsNode <int>`	Defines a node ID of which the emission in the liquid are written out as a function of the radial coordinate.
	`EmissionsMinMax <int>`	Defines the period in which the emission in the liquid are written out as a function of the radial coordinate for the node representing $R_{\min}$, $\dot{R}_{\min}$ and $p_{\mathrm{L,max}}$.

The first line of the results file(s) lists the variables that were written out and their order. For instance, Figure 4.3 shows the different ways in which emissions can be recorded, using the sonoluminescence example found in the `$APECSS_DIR/examples/ultrasound/` folder.

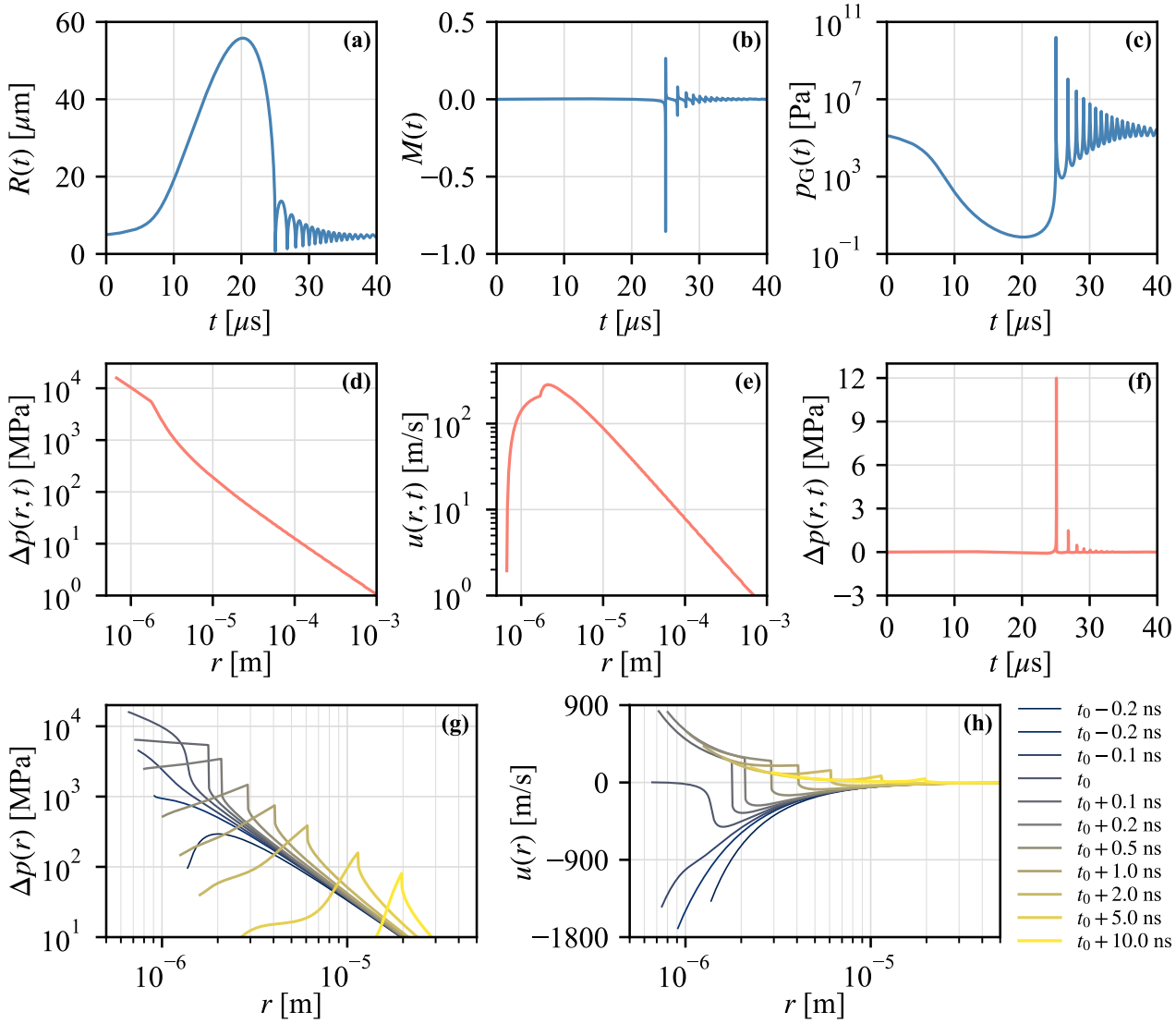


Figure 4.3: Results of an argon bubble with initial radius $R_0 = 5\,\mu\text{m}$ in water, driven by ultrasound with a frequency of 23.5 kHz and a pressure amplitude of 145 kPa, as previously considered by Holzfuss [12]. **(a)-(c)** The bubble radius $R(t)$, bubble-wall Mach number $M(t) = \dot{R}(t)/c_\text{L}(t)$, and gas pressure $p_\text{G}(t)$ as a function of time t. **(d)-(e)** The pressure amplitude $\Delta p(r,t)$ and velocity $u(r,t)$ of the acoustic wave generated by the primary collapse of the bubble as a function of the radial coordinate r. **(f)** The pressure amplitude $\Delta p(r,t)$ emitted by the bubble at a fixed radial distance $r = 100\,\mu\text{m}$ from the bubble center as a function of time t. **(g)-(h)** Spatial profiles of the pressure amplitude $\Delta p(r,t)$ and the velocity $u(r,t)$ at selected time instances, where t_0 is the time at which the bubble assumes its minimum radius. This example can be found in `$APECSS_DIR/examples/ultrasound`, called `sonolum_emissions`.

Bibliography

[1] Brenner, M. P., Hilgenfeldt, S., and Lohse, D. (2002). Single-bubble sonoluminescence. *Reviews of Modern Physics*, **74**(2), 425–484.

[2] Chandran, J. and Salih, A. (2019). A modified equation of state for water for a wide range of pressure and the concept of water shock tube. *Fluid Phase Equilibria*, **483**, 182–188.

[3] Cole, R. H. (1948). *Underwater Explosions*. Princeton University Press, Princeton, New Jersey.

[4] Denner, F. (2021). The Gilmore-NASG model to predict single-bubble cavitation in compressible liquids. *Ultrasonics Sonochemistry*, **70**, 105307.

[5] Denner, F. (2024). The Kirkwood–Bethe hypothesis for bubble dynamics, cavitation, and underwater explosions. *Physics of Fluids*, **36**, 051302.

[6] Denner, F. and Schenke, S. (2023). Modeling acoustic emissions and shock formation of cavitation bubbles. *Physics of Fluids*, **35**(1), 012114.

[7] Dormand, J. and Prince, P. (1980). A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, **6**(1), 19–26.

[8] Fay, R. D. (1931). Plane sound waves of finite amplitude. *The Journal of the Acoustical Society of America*, **3**(2A), 222–241.

[9] Gilmore, F. R. (1952). The growth or collapse of a spherical bubble in a viscous compressible liquid. Technical Report Report No. 26-4, California Institute of Technology, Pasadena, California, USA.

[10] Gümmer, J., Schenke, S., and Denner, F. (2021). Modelling Lipid-Coated Microbubbles in Focused Ultrasound Applications at Subresonance Frequencies. *Ultrasound in Medicine & Biology*, **47**(10), 2958–2979.

[11] Hickling, R. and Plesset, M. S. (1963). The collapse of a spherical cavity in a compressible liquid. Technical Report 85-24, California Institute of Technology, Pasadena, California, USA.

[12] Holzfuss, J. (2010). Acoustic energy radiated by nonlinear spherical oscillations of strongly driven bubbles. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **466**(2118), 1829–1847.

[13] Hua, C. and Johnsen, E. (2013). Nonlinear oscillations following the Rayleigh collapse of a gas bubble in a linear viscoelastic (tissue-like) medium. *Physics of Fluids*, **25**(8), 083101.

[14] Jiménez-Fernández, J. and Crespo, A. (2005). Bubble oscillation and inertial cavitation in viscoelastic fluids. *Ultrasonics*, **43**(8), 643–651.

[15] Kaykanat, S. I. and Uguz, K. (2024). Shape stability of a microbubble in a power–law liquid. *The European Physical Journal Special Topics*, **233**(8-9), 1625–1635.

[16] Keller, J. B. and Miksis, M. (1980). Bubble oscillations of large amplitude. *The Journal of the Acoustical Society of America*, **68**(2), 628–633.

[17] Kirkwood, J. G. and Bethe, H. A. (1942). The pressure wave produced by an underwater explosion I. Technical Report Report No. 588, Office of Scientific Research and Development.

[18] Kontogeorgis, G. M., Privat, R., and Jaubert, J.-N. (2019). Taking Another Look at the van der Waals Equation of State–Almost 150 Years Later. *Journal of Chemical & Engineering Data*, **64**(11), 4619–4637.

[19] Lauterborn, W. and Kurz, T. (2010). Physics of bubble oscillations. *Reports on Progress in Physics*, **73**(10), 106501.

[20] Le Métayer, O. and Saurel, R. (2016). The Noble-Abel Stiffened-Gas equation of state. *Physics of Fluids*, **28**(4), 046102.

[21] Marmottant, P., van der Meer, S., Emmer, M., Versluis, M., de Jong, N., Hilgenfeldt, S., and Lohse, D. (2005). A model for large amplitude oscillations of coated bubbles accounting for buckling and rupture. *The Journal of the Acoustical Society of America*, **118**(6), 3499–3505.

[22] Neppiras, E. A. (1980). Acoustic cavitation. *Physics Reports*, **61**(3), 159–251.

[23] Overvelde, M., Garbin, V., Sijl, J., Dollet, B., de Jong, N., Lohse, D., and Versluis, M. (2010). Nonlinear Shell Behavior of Phospholipid-Coated Microbubbles. *Ultrasound in Medicine & Biology*, **36**(12), 2080–2092.

[24] Prosperetti, A. and Lezzi, A. (1986). Bubble dynamics in a compressible liquid. Part 1. First-order theory. *Journal of Fluid Mechanics*, **168**, 457–478.

[25] Trilling, L. (1952). The Collapse and Rebound of a Gas Bubble. *Journal of Applied Physics*, **23**(1), 14–17.

[26] Versluis, M., Stride, E., Lajoinie, G., Dollet, B., and Segers, T. (2020). Ultrasound Contrast Agent Modeling: A Review. *Ultrasound in Medicine & Biology*, **46**(9), 2117–2144.