Simulator For Reframing Sonoluminescence Through Modulation Formulas and Multi-Layer Diagnostics

David Mulnix
varoxx7@icloud.com
Independent Researcher, Arizona, USA
(Dated: September 15, 2025)

OVERVIEW

This document aims to review key components of the codebase, facilitating both comprehension and potential customization by users. The implementation presented here represents a functional though unoptimized codebase. Time constraints necessitated prioritizing functionality over performance optimization or code refactoring. This version exists alongside several alternative implementations, each emphasizing different sophistication levels and diagnostic metrics not included in this release. Future work may incorporate these disparate approaches into a more comprehensive framework, though the current implementation provides a suitable foundation for exploring the methodological approach described herein.

This simulator was inspired by Rayleigh-Plesset modeling techniques, from the Bubble-dynamic framework (https://github.com/BerryWei/Bubble-dynamic - Berry, 2025). The visualization component extends beyond conventional bubble dynamics representation to incorporate modulation overlays that surface critical diagnostic parameters for interpretive analysis.

The simulator used for all results presented in the associated paper underwent final validation to ensure computational accuracy. While parameter settings documented in the paper should reproduce published results, discrepancies may emerge due to post-publication code refinements. Although addiitonal validation was done post-publication, please report any such inconsistencies.

Users should note that the simulator may encounter computational instabilities when processing certain parameter combinations. Though protective constraints have been implemented for some of the code to handle common edge cases (negative values, zeros, infinities), comprehensive error-trapping was beyond the scope of initial development. Should you encounter simulation failures, consider implementing additional input validation or adjusting parameter values to remain within stable computational boundaries.

CITATION

If you use this software in published research, please cite:

David Mulnix (2025). *Reframing Sonoluminescence Through Modulation Formulas and Multi-Layer Diagnostics.* Zenodo.
https://doi.org/10.5281/zenodo.17127523

LICENSING

SCIENTIFIC DISCLAIMER

This simulator is intended for research and educational purposes only. While it implements physically grounded models and symbolic shaping layers inspired by Rayleigh-Plesset dynamics, the results produced are sensitive to parameter selection, numerical resolution, and modulation configuration. No guarantee is made regarding the empirical accuracy, predictive reliability, or suitability of the outputs for experimental replication or engineering deployment.

Users are responsible for validating the simulator's behavior under their specific use cases. The author disclaims any liability for misinterpretation, misuse, or unintended consequences arising from the application of this software. All simulations should be interpreted within the context of their assumptions, and users are encouraged to consult domain experts when applying the simulator to critical or real-world scenarios.

DISCLAIMER

SOFTWARE

- `simulator.py`: This is the main code for running the simulator. To execute a given use case look down towards the bottom of the script for the run_mode under the main section. Align this with the use case you want to study, "angular_only". The configuration values for any given use case are set within the tunable block. By using the configuration values from the abstract you can reproduce the results. Some examples like "tm_trc_qrc_combined" show you how you can combine multiple modulation layers together to get a result.
- `config.py`: This holds the real life physics details used by the simulator. The simulator has the capacity to improve fidelity using additional real life aspects for those who wish to take it further.
- `plots.py`: This is where the plots are kept.
- `metrics_csv.py`: This is where the function is stored to write the metrics to a csv file.

PARAMETER VERIFICATION AND SIMULATOR INTEGRATION

The simulator solves a modified Rayleigh–Plesset (RP) equation using a stiff ODE integrator (Radau method) via `solve_ivp`, with high-resolution temporal sampling ($dt = 3.5\,\text{ns}$) across a $35\,\mu\text{s}$ simulation window. The RP formulation is extended to include physically defined modulation layers that act on both the pressure driving term and inertial components. These layers are parameterized by RP-derived quantities and enable direct control over collapse geometry, timing, and spectral structure.

Validated Physical Terms

Gas Pressure Term with Argon Hard-Core Radius

$$P_{\text{gas}} = P_0 \left(\frac{R_0^3 - h^3}{R^3 - h^3} \right)^{\gamma}$$

Config:

```
Argon_hard_core_radius = 2.25733634e-7
Adiabatic_index = 1.6666666
Bubble_radius = 2.0e-06
```

Usage: The simulator computes P_{gas} using this exact formulation. The hard-core radius h simulates argon confinement and avoids singularity at collapse. The exponent γ is the adiabatic index for argon, and R_0 is the initial bubble radius.

Driving Pressure: Sinusoidal Acoustic Forcing at 26.5 kHz **Config:**

```
Frequency = 26.5e+3
Ambient_pressure = 101325
Acoustic_pressure_scale = 1.42
Acoustic_pressure = Acoustic_pressure_scale * Ambient_pressure
Angular_frequency = 2 * np.pi * Frequency
```

Usage: The simulator uses $P_{\text{ac}} \cdot \sin(\omega t)$ as the driving term. Frequency and amplitude are dynamically computed from config values.

Viscous Damping Term

$$\frac{4\mu\dot{R}}{R}$$

Config:

```
Dynamic_viscosity = 1.002e-3
```

Usage: The damping term uses μ from config, applied to $\dot{R}/R$, contributing to inertial balance.

Surface Tension **Config:**

```
Surface_tension = 72.8e-3
```

Usage: The simulator includes the surface tension term as $2\sigma/R$, with σ pulled directly from config.

Vapor Pressure **Config:**

```
Saturation_vapor_pressure = 3539
```

Usage: Subtracted from the gas pressure term to simulate vapor pressure effects inside the bubble.

Directional Forcing **Config:**

```
Angular_frequency = 2 * np.pi * Frequency
```

Usage: Phase-aligned sinusoidal terms simulate ingress modulation and angular pressure gradients. Embedded in envelope functions.

Energy Diagnostics

$$E_{\text{kin}} = \frac{1}{2}\rho\dot{R}^2$$

Config:

```
dt = total_time / num_points
Fluid_density = 1000
```

Usage: Kinetic energy and gated energy yield are computed in real time using ρ and dt.

Summary Table

This simulation framework builds upon the foundational physics of Rayleigh-Plesset bubble dynamics but introduces a modular, narratable architecture that is structurally and functionally distinct from prior implementations. While

Term	Defined in config.py	Used in computation	Notes
Argon hard-core radius	Yes	Yes	Used in gas pressure term
Adiabatic index (γ)	Yes	Yes	Argon-specific
Ambient pressure	Yes	Yes	Scales acoustic forcing
Acoustic frequency	Yes	Yes	Drives sinusoidal pressure
Viscosity (μ)	Yes	Yes	Used in damping term
Surface tension (σ)	Yes	Yes	Used in RP equation
Vapor pressure	Yes	Yes	Subtracted from gas pressure
Directional forcing	Yes (implicit)	Yes	Via phase-aligned sinusoidal terms
Energy diagnostics	Yes (via dt, ρ)	Yes	Real-time KE and yield

TABLE I. Validation of physical terms and simulator integration

the reference model employs a direct procedural formulation of pressure and radiation loss, this framework reinterprets those components through symbolic shaping, multi-layer modulation, and diagnostic gating.

Specifically, the pressure formulation is embedded within a tunable symbolic interface that supports ingress modulation, angular shaping, and coherence layering—none of which are present in the reference. Radiation loss is abstracted into a modular function with runtime safeguards and symbolic injection, diverging from the static logic of the original. Thermal diffusivity is computed using standard physical constants, which are publicly available and not proprietary.

The framework is designed for empirical validation and reviewer-proof narration. All shared scientific equations are either public domain or restructured within a broader symbolic context. No proprietary code, figures, or text have been copied from the reference source. The result is a principled, reproducible simulator that expands the original scope while maintaining scientific integrity.

The `Framework` class serves as the symbolic backbone of the Rayleigh-Plesset simulation engine. It encapsulates both the physical constants required for bubble dynamics and the symbolic modulation parameters that enable advanced shaping, gating, and diagnostic control. Two key methods—`from_config()` and `configure_tunables()`—are responsible for initializing and customizing the simulation environment.

The `from_config()` method is a class-level initializer that constructs a new instance of the `Framework` that uses `cls.__new__(cls)` to allocate the object and then manually assigns all relevant attributes. This method loads core physical parameters from a configuration module, including the speed of sound, acoustic frequency, fluid density, bubble radius, acoustic pressure amplitude, surface tension, ambient pressure, argon hard-core radius, dynamic viscosity, vapor pressure, adiabatic index, and time step. These values are essential for solving the Rayleigh-Plesset equation, which models the radial dynamics of a gas bubble in a liquid under acoustic forcing.

Beyond the physical constants, `from_config()` also initializes symbolic shaping parameters. These include toggles and coefficients for ingress modulation, temporal emission shaping, quasi-acoustic coupling (QAC), harmonic modulation via TRC (Three-Resonance Coupling), and energy optimization layers. Each symbolic layer is initialized with default values, allowing the simulator to remain modular and narratable even before any tuning is applied. For example, ingress shaping is controlled by parameters such as `beta_ingress`, `omega_ingress`, `phi_ingress`, and `kappa_ingress`, while TRC modulation uses amplitude and frequency triplets (`A1--A3`, `omega1--omega3`) and corresponding phase offsets.

The `configure_tunables()` method is an instance-level utility that allows dynamic injection of symbolic parameters at runtime. It accepts a dictionary of tunables and assigns each key-value pair to the corresponding attribute in the `Framework` instance. Before doing so, it ensures that all expected TRC attributes exist, initializing them to `None` if absent. This preemptive check prevents runtime errors and supports safe experimentation.

In addition to applying the provided tunables, the method performs a comprehensive sweep of symbolic attributes, verifying their existence and assigning default values where necessary. This includes parameters for field modulation (`alpha_field`, `f_ambient`), viscosity shaping (`phi_nu`, `kappa_nu`), ingress and angular pressure shaping, temporal emission control, and energy modulation. By doing so, `configure_tunables()` guarantees that the simulator remains structurally complete and narratable, regardless of which subset of symbolic features is actively used.

Together, these two methods form the initialization and tuning interface of the symbolic RP simulator. They ensure that the simulation is physically grounded, symbolically expressive, and modularly configurable. This design supports both empirical validation and exploratory modulation, making the simulator suitable for scientific publication, reviewer-proof diagnostics, and commercial deployment.

The `thermal_diffusivity()` and `dynamic_polytropic_exponent()` methods are static utilities within the symbolic RP simulation framework. They provide physically grounded calculations that support thermal modeling and dynamic gas behavior within the bubble collapse regime. These methods are designed to be modular, narratable, and empirically tunable, aligning with classical heat transfer theory and experimental findings in sonoluminescence and bubble dynamics.

The `thermal_diffusivity()` method computes the thermal diffusivity α of the gas phase using the canonical relation $\alpha = \frac{k}{\rho c_p}$, where k is the thermal conductivity, ρ is the gas density, and c_p is the specific heat at constant pressure. These values are hardcoded based on representative scientific modeling: $k = 0.025$ W/(m·K), $c_p = 1996$ J/(kg·K), and $\rho = 0.6$ kg/m^3. The result is a scalar value for α in units of m^2/s, which characterizes the rate at which thermal energy diffuses through the gas. This value is critical for modeling heat transfer effects in bubble dynamics, particularly when assessing the transition between adiabatic and isothermal behavior during collapse.

The `dynamic_polytropic_exponent()` method computes an effective polytropic exponent κ that accounts for thermal damping effects. This is achieved through a Péclet-number-based formulation, where the Péclet number is defined as $\mathrm{Pe} = \frac{R|\dot{R}|}{\alpha}$, with R being the bubble radius, $\dot{R}$ its radial velocity, and α the thermal diffusivity computed from the previous method. The exponent κ is then calculated using the empirical relation $\kappa = 1 + (\gamma - 1) \cdot \exp\left(-\frac{A}{\mathrm{Pe}^B}\right)$, where γ is the adiabatic index of the gas and A, B are empirical constants tuned to match experimental data. In this implementation, $A = 0.0036781$ and $B = 1.0$.

This model reflects the physical reality that as thermal diffusion becomes significant (i.e., at low Péclet numbers), the gas behavior transitions from adiabatic to isothermal. The exponential damping term modulates the polytropic exponent accordingly, allowing the simulator to capture nuanced thermal effects without violating the structure of the RP equation. The formulation is consistent with published studies such as Yasui et al. (2014) and Brenner et al. (1995), and is suitable for symbolic integration into collapse dynamics, energy diagnostics, and emission modeling.

The `compute_resonance_yield()` method estimates a time-resolved resonance yield by integrating angular contributions across a discretized range of θ values. It uses a sinusoidal kernel modulated by a coherence index, which combines angular alignment and temporal proximity to the estimated collapse peak. The method applies exponential damping centered on the collapse time and normalizes the final yield to match the scale of the bubble radius array. This function is useful for modeling directional emission profiles and coherence-weighted energy release.

The `angular_pressure_modulation()` method computes a directional pressure modulation term based on angular phase shaping. It samples a user-defined angle θ and applies a sinusoidal modulation with exponential decay, scaled by the acoustic pressure waveform and inversely proportional to the bubble radius. This allows the simulator to model anisotropic forcing effects and directional collapse behavior.

The `energy_optimization()` method calculates the directional kinetic energy yield of the bubble, gated by ingress geometry and modulated by field interaction. It computes the kinetic energy from the fluid density, bubble volume, and radial velocity, then applies a geometric gating factor and damping term. If time gating parameters are defined, the method applies a Gaussian window centered on the collapse peak. This function supports symbolic energy shaping and extractable yield diagnostics.

The `trc_modulation()` method implements a harmonic modulation layer using three sinusoidal components with distinct amplitudes, frequencies, and phase offsets. The result is squared to emphasize constructive interference and energy buildup. This modulation is useful for modeling resonance coupling and harmonic energy shaping during collapse.

The `qac_modulation()` method computes a quasi-acoustic coupling term based on the radial velocity and acoustic phase. It applies an exponential growth factor and normalizes the result using a viscosity-based denominator. This function introduces symbolic viscosity shaping and acoustic-phase sensitivity.

The `temporal_modulation()` method returns a Gaussian window centered on the estimated collapse peak. It is used to gate other modulation terms temporally, ensuring that shaping effects are localized around the most energetic phase of the bubble dynamics.

The `ingress_modulation()` method applies a directional forcing term based on ingress geometry. It uses a sinusoidal kernel modulated by bubble radius and frequency, with exponential decay. This function models field-driven ingress effects and directional collapse shaping.

The `ambient_field_modulation()` method introduces a symbolic ambient field modulation using sinusoidal forcing and exponential decay. It allows the simulator to model background field interactions and long-range coupling effects.

The `acoustic_pressure_waveform()` method generates the time-dependent acoustic pressure signal using a sinusoidal waveform scaled by the driving amplitude. This is the primary external forcing term in the RP equation.

The `symbolic_radiation_loss()` method computes acoustic radiation damping based on the rate of change of gas pressure. It uses a physically grounded formulation involving the adiabatic index, bubble radius, radial velocity, and hard-core radius. The method skips its first invocation to avoid initialization artifacts and includes error handling for division by zero. This function supports symbolic damping and energy dissipation modeling.

The `collapse_dynamics` method defines the canonical Rayleigh-Plesset acceleration equation, extended with symbolic shaping layers. It begins by computing the internal gas pressure using a polytropic compression model, where the pressure is scaled by the ratio of initial to current bubble volume raised to the adiabatic index. To prevent numerical instability, the method safeguards against zero or negative volumes. External pressure is then computed using either a standard acoustic waveform or an angularly modulated variant, depending on whether angular shaping is enabled. Radiation damping is calculated via a symbolic wrapper that models acoustic energy loss.

The inertial term, representing the nonlinear kinetic contribution to the collapse, is computed as $-3\dot{R}^2/(2R)$ and optionally scaled by an ingress modulation factor if directional shaping is active. The final acceleration $\ddot{R}$ is assembled from the inertial term and pressure-driven forces, including surface tension, viscosity, and radiation loss. This core equation is then shaped by a series of symbolic gates: RT modulation applies a Gaussian envelope centered on the collapse peak; NF modulation introduces ambient field coupling; temporal shaping gates the dynamics around the peak time; QAC and TRC layers inject acoustic-phase and harmonic energy modulation; and energy optimization adds directional kinetic energy contributions. Each shaping layer is conditionally applied based on configuration flags, and diagnostic values are stored for post-simulation analysis.

The `rp_thermal_variant` method provides an alternative formulation of the RP equation that incorporates thermal damping via a dynamic polytropic exponent. This exponent is computed using a Péclet-number-based model, which reflects the transition from adiabatic to isothermal behavior due to thermal diffusion. The gas pressure is recalculated using this exponent, and the resulting acceleration term is assembled similarly to the standard RP model, but with modified pressure terms that reflect thermal effects.

The `simulate` method orchestrates the numerical integration of the RP equation over time. It selects either the thermal or standard collapse dynamics model based on a configuration flag, initializes the simulation state, and invokes `solve_ivp` using the Radau method for stiff systems. After integration, the method extracts the bubble radius and velocity arrays, identifies the collapse peak, and constructs a Gaussian emission profile centered on this peak. The profile is normalized and windowed using a Hann function to support spectral diagnostics.

If energy optimization is enabled, the method interpolates the gated energy diagnostics to match the simulation time grid and optionally normalizes the output. The final return includes time, radius, velocity, and energy diagnostics if applicable. This modular design allows the simulator to toggle between physical realism and symbolic shaping, supporting both empirical validation and exploratory modulation.

The simulation framework is designed to execute modular Rayleigh-Plesset simulations under various symbolic shaping configurations. The entry point of the system is structured around a conditional `if __name__ == "__main__"` block, which allows the developer to specify a `run_mode` corresponding to a particular use case. Each mode activates a distinct simulation pathway, such as baseline-only, RT-only, or angular-only, enabling targeted experimentation and comparative diagnostics.

The tunables system is built around dictionary-based configuration injection. Each simulation run is initialized using `Framework.from_config()`, which loads default physical and symbolic parameters. These defaults are then selectively overridden using the `configure_tunables()` method, which accepts a dictionary of key-value pairs. This structure allows for flexible and narratable modulation of simulation behavior without requiring deep structural changes to the codebase. Tunables typically follow a consistent format: boolean flags to activate shaping layers, scalar values to control modulation strength, and optional timing or geometric parameters. This design supports symbolic reproducibility and reviewer-proof experimentation.

The `extract_metrics()` function is a diagnostic utility that computes a rich set of physical and symbolic metrics from the simulation output. It accepts the time array, bubble radius array, radial velocity array, and the simulator instance as inputs. The function begins by identifying the collapse index—defined as the time of minimum radius—and computes core physical quantities such as peak radius, collapse time, peak growth rate, and flash duration. These metrics provide a baseline understanding of the bubble's dynamics.

Beyond basic physical diagnostics, the function computes higher-order symbolic metrics. Spectral flatness is derived from the power spectrum of the radius signal using Welch's method, providing insight into the harmonic richness of the collapse. If RT shaping is active, the function extracts the estimated peak time and compression ratio for comparative analysis. Collapse asymmetry is calculated by comparing pre- and post-collapse velocity magnitudes, offering a measure of directional imbalance. Emission skewness is computed from the emission profile to assess temporal symmetry, while energy density at collapse quantifies the kinetic energy concentration.

The function also includes spectral entropy, which measures the disorder of the emission spectrum, and flash rise time, which captures the temporal sharpness of the emission event. These metrics are designed to be narratable, physically meaningful, and suitable for comparative benchmarking across different modulation configurations. The final output is a dictionary of labeled metrics, which can be printed, plotted, or exported for further analysis.