LA-UR-24-29056

Approved for public release; distribution is unlimited.

Title:	Black Box Equations of State: Creating Semi-analytic Solutions to the Noh Problem and Verifying Equation of State Interfaces
Author(s):	Gerberding, Seth James
Intended for:	Report
Issued:	2024-08-21

Black Box Equations of State: Creating Semi-analytic Solutions to the Noh Problem and Verifying Equation of State Interfaces

Seth J. Gerberding

August 2024

Abstract

The objective of this report is threefold. First, it details a method for deriving a semi-analytic solution to the Noh Problem when using a "black-box" equation of state. Such capability allows us to perform verification on complicated, more realistic equations of state. Examples include Steinberg equations of state for materials and tabulated equations of state. The second objective is to apply the methodology to verify the `singularity-eos` equation of state library. We do so by solving the Rankine-Hugoinot jump conditions for the Noh Problem, ensuring `singularity` derives the correct solution and comparing the error to an exact implementation of the equation of state. The third objective is to perform verification of the `xRAGE` Eulerian hydrodynamics code when interfaced with `singularity`. We provide the theory, analysis, documentation for a python implementation of the proposed solver, and verification results.

1 Introduction

The purpose of this report is to detail a method for deriving a semi-analytic solution to the Noh Problem Noh [1987] when a "black-box" equation of state (EoS) is used and then applying the method to verify the `singularity-eos` equation of state library Peterson et al. [2022] and the Eulerian hydrodynamics solver `xRAGE` Gittings et al. [2008].

Broadly speaking, this project is a continuation of the work and ideas initiated in Burnett et al. [2017], Ramsey et al. [2017]. The goal is to develop semi-analytic solutions to compare numerical simulations against. Of course, this requires having access to analytic solutions. In the context of hydrodynamics, analytic solutions are few and far between, and often depend on the equation state, initial conditions, and other parameters. Therefore, being able to find solutions with arbitrary initial conditions, parameters, and equations of state is of great importance. One problem that has received a lot of attention with regards to different EoSs is the *Noh Problem* Axford [2000]. It is this project—finding solutions based on different equations of state—that we continue, with a focus on the Noh Problem.

More specifically, this report is a direct sequel to the work done in Gerberding and Musick [2023]. In that report, the authors analyzed a class of equations of state and the analytic solutions to the Noh problem which they admit. However, that class, while more general, does not address many well-known or commonly used equations of state, such Steinberg (or Mie-Gruneisen) equations of state Steinberg et al. [1996] and tabulated equations of state. Gerberding and Musick [2023] did touch on the question, but the work was still preliminary. This work continues that analysis. In particular, we provide a technique for deriving semi-analytic solutions to the Noh Problem that is agnostic, up to theoretical assumptions, to the equation of state and initial conditions.

Our approach rests upon the observation made in Burnett et al. [2017]: the Rankine-Hugoinot jump conditions create a system of nonlinear equations involving the shocked density, pressure, and shock speed variables. If this system can be solved, then the semi-analytic solution to the Noh problem is an immediate consequence. Indeed, Burnett et al. [2017] used this approach to find a solution when using the Carnahan-Starling equation of state, and used it to run a verification test.

Our technique generalizes this approach. In particular, we use a Newton iteration solver to solve the jump conditions. We also analyze a special case when the initial pressure and symmetry is zero. In this case, the jump conditions simplify to a system of two nonlinear equations. From a numerical perspective, this setting is much simpler as the determinant and the inverse of the Jacobian can be computed directly. From a verification perspective, this is a much "easier" problem to solve since only two variables need to solved for instead of three. However, from a computation hydrodynamics perspective, zero pressure can be a difficult state to simulate. We touch on how to combine these two problems to find solutions to more realistic problems. In either case, our technique is completely agnostic to the equation of state, allowing us to use black-box EoSs.

By "black-box" we mean that we have limited access to EoS information. For example, we do not *a priori* assume any particular form or properties beyond those required by solution theory. Thus, when we say the EoS is given by a relationship of either $e = e(\rho, P)$ or $P = P(\rho, e)$, we only mean that energy can be found given density

and pressure, and similarly for pressure given density and energy. The same assumptions are made regarding partial derivatives, but even these assumptions are not strictly necessary as they can be approximated using finite differences. The key point, however, is that as long as these values are available (along with initial conditions and other problem parameters), the solver can iterate. Put more succinctly, if the mathematics permits a solution to exist, then our technique has hope of finding the necessary information to construct it. While we cannot prove convergence of the method, our tests show promising capabilities for the approach. Our technique, therefore, allows an user to verify their codes for different initial conditions, geometries, and equations of state.

One question in the context of verification is how to verify an EoS framework. One test is point-queries, where specific points are tested. Another test is running the EoS framework in a hydrodynamics code. We have developed a new test, one that is more intense than point queries but does not require a complicated hydro code. That test is to solve the Noh Problem jump conditions. That is, we view the jump conditions as an *independent* problem outside the context of hydrodynamics. We apply this test to `singularity-eos`.

Hence, while our technique is seemingly simple, it provides a plethora of verification capabilities. It helps provide semi-analytic solutions to the Noh problem based on different initial conditions, geometries, and equations of state. These solutions help verify hydrocodes. Furthermore, the jump conditions *themselves* provides a verification test for equation of state frameworks.

The rest of this report is organized as follows. Section 2 details the theory behind the solution method; section 3 discusses the `singularity-eos` equation of state library. Section 4 provides documentation for our python code which implements the solver method described in section 2. Section 5 provides examples of solutions to the jump conditions for a variety of equations of state, ranging from polytropic ideal gas to a tabulated EoS. Section 6 details work done to integrate the solver into the exact solution package `ExactPack`, and section 7 provides the verification results. We conclude the report with section 8 where we discuss the results of the project and future research.

2 Theory

In this section, we present the mathematical theory underpinning our solution technique. We briefly survey the Noh Problem, including its specific jump conditions and solution. We state several mathematical results that motivate the proposed method, summarize the theoretical restrictions on the equation of state, and present the method both in terms of equations of state posed as $e = e(\rho, P)$ or $P = P(\rho, e)$. In both cases, we present the algorithm in its full generality (three variable residual) and a simplified version (two variable residual).

2.1 The Noh Problem

We follow the setting described in Ramsey et al. [2017], Axford [2000], Burnett et al. [2017]. The *Noh Problem* Noh [1987], consists of an initially uniform inflow of constant velocity colliding against a wall (planar, 1D case), a central axis (cylindrical, in 2D), or a point (spherical, in 3D). A hydrodynamic shock forms at $t > 0$ and propagates into the still incoming fluid. To be more specific, we are concerned with the "classic" Noh Problem insofar that we insist that the solution consists of two regions separated by shock, and that each region has constant velocity and pressure. There have been further generalizations of the Noh Problem that do not make such demands. For example, in Velikovich and Giuliani [2018] the authors do *not* assume a uniform inflow velocity.

However, unlike previous works, we remain agnostic to the particular form of the EoS. That is, we do not *a priori* assume that the EoS is of the kind $e = e(\rho, P)$ or any other. The only thing we do assume is that there exists a relation between e, P, and ρ. Furthermore, we do not assume, beyond the definition of the Noh Problem, any further restrictions on the initial conditions.

We let D denote the shock speed, which analysis shows is constant. We let ρ_L, P_L, e_L denote the variable quantities (density, pressure, and internal energy, respectively) behind the shock and ρ_R, P_R, e_R in front of the shock. We also let $m \in \{0, 1, 2\}$ denote the symmetry variable; that is, $m = 0$ corresponds to the Cartesian (planar) case, $m = 1$ the axisymmetric (cylindrical) case, and $m = 2$ the spherically symmetric case. We let x denote the spatial variable *in radial coordinates*, $t \geq 0$ the time variable, and (ρ_0, P_0, u_0) the initial conditions of the problem. Observe that $u_0 < 0$ by assumption. Finally, we let $\xi := \frac{x}{t}$ denote the self-similar variable and K the adiabatic bulk modulus.

2.1.1 General Solution

The solution to the Noh Problem is found by exploiting symmetries in the problem to leverage self-similar variables. The problem is cast in terms of $\xi := \frac{x}{t}$, which turns the Euler equations into a system of ODEs. Using the initial conditions, those ODEs can be solved in the pre and post shock regions. The solutions are presented below in a concise form:

$$\rho(t,x) = \begin{cases} \rho_L & \frac{x}{t} < D \\ \rho_0 \left(1 - \frac{u_0 t}{x}\right)^m & \frac{x}{t} > D \end{cases} \tag{1a}$$

$$P(t,x) = \begin{cases} P_L & \frac{x}{t} < D \\ P_0 & \frac{x}{t} > D \end{cases} \tag{1b}$$

$$u(t,x) = \begin{cases} 0 & \frac{x}{t} < D \\ u_0 & \frac{x}{t} > D \end{cases} \tag{1c}$$

2.1.2 Rankine-Hugoinot Jump Conditions

Of paramount importance is the Rankine-Hugoinot jump conditions, which are derived from conservation laws enforced at the shock. The general jump conditions—that is, not specific to the Noh Problem—are given by:

$$(u_L - D)\rho_L = (u_R - D)\rho_R \tag{2a}$$

$$P_L + (u_L - D)\rho_L u_L = P_R + (u_R - D)\rho_R u_R \tag{2b}$$

$$e_L + \frac{P_L}{\rho_L} + \frac{1}{2}(u_L - D)^2 = e_R + \frac{P_R}{\rho_R} + \frac{1}{2}(u_R - D)^2. \tag{2c}$$

To leverage the assumptions of the Noh Problem, we begin by using (2a) to rewrite (2b):

$$P_L + (u_L - D)\rho_L u_L = P_R + (u_L - D)\rho_L u_R. \tag{3}$$

We now use the assumption that $u_L = 0$ and $u_R = u_0$, therefore arriving at the following system of equations:

$$-D\rho_L = (u_0 - D)\rho_R \tag{4a}$$

$$P_L = P_R - \rho_L D u_0 \tag{4b}$$

$$e_L + \frac{P_L}{\rho_L} = e_R + \frac{P_R}{\rho_R} + \frac{1}{2}u_0(u_0 - 2D) \tag{4c}$$

To cast the jump conditions in terms of ρ_L, P_L, D, we first substitute $\rho_R = \rho_0 \left(1 - \frac{u_0}{D}\right)^m$ into (4a). We keep (4b) as is, and using the fact that (4b) gives $D = \frac{P_0 - P_L}{\rho_L u_0}$, we manipulate (4c) to obtain the following form of the jump conditions:

$$\rho_L = \rho_0 \left(1 - \frac{u_0}{D}\right)^{m+1} \tag{5a}$$

$$P_L = P_0 + \rho_L u_0 D \tag{5b}$$

$$e_L + \frac{u_0 P_0}{\rho_L D} = e_R + \frac{1}{2}u_0^2 \tag{5c}$$

Equations (5a)-(5c) form the system that we propose to solve for ρ_L, P_L, and D. The key to doing so is using the equation of state to close the system by providing the relationship between e, ρ, and P. As noted in theorem 2.1 (see later), this is sufficient to construct the solution.

2.1.3 EoS restrictions

Before we proceed, it is imperative to review restrictions on the equation of state that may prevent a solution from existing. These are necessary rather than sufficient conditions: if these restrictions are violated, then a solution simply cannot exist. It also provides users with guidance on the scope of verification that the Noh Problem provides. We refer the reader to Burnett et al. [2017], Ramsey et al. [2017] for details.

The Euler equations for the Noh Problem, after being transformed into self-similar variables, become a system of ODEs. One of the ODEs (namely the pressure equation) implies, for $\xi := \frac{x}{t}$, the following identity:

$$\frac{Km}{\xi} = 0. \tag{6}$$

The constraint is trivially satisfied for $m = 0$, i.e., the planar problem. However, when $m \neq 0$, then $K = 0$ is a must. This restriction in turn produces the following constraint on the equation of state in the unshocked region:

$$e + \frac{P_0}{\rho_0}\left(1 - \frac{u_0 t}{x}\right)^{-m} = \text{constant}. \tag{7}$$

Since $P = P_0$ in the unshocked region, in practice, this restriction is satisfied by setting $P_0 = 0$ while the equation of state must also satisfy $P_0 = 0 \implies e = \text{const}$. This constraint is going to prohibit many equations of state in higher geometries. We summarize in table 1:

Symmetry	Restriction on the EoS & Pressure
$m = 0$	None
$m = 1$	$P_0 = 0$ and $P_0 = 0 \implies e = \text{const}$.
$m = 2$	$P_0 = 0$ and $P_0 = 0 \implies e = \text{const}$.

Table 1: Summary of the restrictions on the equation of state for the Noh Problem.

The important thing to notice is that an equation of state can *always* be tested in $m = 0$ with *any* choice of initial conditions (assuming they satisfy the setting for the Noh Problem, of course). However, this does not mean a solution will exist. For example, arbitrary initial conditions may not be in an EoS's admissible set, even though it satisfies the assumptions of the Noh Problem.

2.1.4 Mathematical theory

In this section, we state the critical mathematical results that underpin the remainder of our work. These results are not particularly difficult. Indeed, we do not claim any form of originality. If anything, these are just pointing out facts about the Noh Problem they have not been explicitly state before. Observe that these results are agnostic to the equation of state. We assume that the equation of state and initial conditions admit a solution to the Noh Problem.

Theorem 2.1. *The solution to the Noh Problem is completely determined by ρ_L, P_L, D.*

Proof. Once these quantities are known, (1a) - (1c) are well-defined. □

Corollary 2.2. *Solving the Noh Problem is equivalent to solving the Rankine-Hugoinot conditions.*

Proof. If the solution to the Noh Problem is known, then the quantities must satisfy the Rankine-Hugoinot conditions. Alternatively, if the solution to the Rankine-Hugoinot jump conditions are known, then theorem 2.1 implies that the solution is known. □

Remark 2.3. Corollary 2.2 is the critical mathematical observation that underpins the remainder of this work. The ability to solve the jump conditions is sufficient to construct a semi-analytic solution to the Noh Problem. Thus, the objective becomes solving (2a)-(2c) for ρ_L, P_L, and D.

With these facts, we may now present our method for the solving the jump conditions, and thus, the Noh Problem. First, we use the equation of state to provide a relation between the e, ρ and P variables, thereby closing (5a) - (5c). We then form a residual function $\mathbf{F}$ (note that $\mathbf{F}$ depends on the EoS), its Jacobian, $\mathbf{F}'$, the inverse of the Jacobian $(\mathbf{F}')^{-1}$, choose and initial guess x_0, and perform a Newton iteration:

$$x_{n+1} = x_n - (\mathbf{F}')^{-1}(x_n)\mathbf{F}(x_n). \tag{8}$$

If there is a computational problem, (division by zero, zero determinant, etc.) then another x_0 must be chosen. A solution is selected when $\|x_{n+1} - x_n\|$ is below some tolerance (testing for convergence) and when $\|\mathbf{F}(x)\|$ is also below a tolerance (testing for a solution).

Remark 2.4 (Initial Guess)**.** As with all nonlinear solvers, the choice of x_0 is a tricky question. While there is not a complete answer available for our problem, we can narrow our search. First, (5b) shows that $P_L > P_R = P_0$, and so $D > 0$. Next, using (2a), we have the following relation for ρ_L:

$$\rho_L = \rho_0 \left(\frac{D - u_0}{D} \right)^{m+1}. \tag{9}$$

Since $u_0 < 0$, (9) implies that $\rho_L > \rho_0$. Therefore, we can narrow our search to $S := \{(\rho, P, D) \in \mathbb{R}^3 : \rho > \rho_0,\ P > P_0,\ D > 0\}$ and choose $x_0 \in S$.

2.1.5 Three variable residual

In this section we detail two approaches to solving the jump conditions (5a)-(5c). They are based on two formulations of the EoS: $e = e(\rho, P)$ and $P = P(\rho, e)$. However, we do *not* assume that these are given in a closed form. Indeed, the only thing we assume is that we can obtain the energy via density and pressure or pressure via density and energy through some "black box" mechanism. For implicitly, we further assume we have access to the relevant partial derivatives through the black box, but this is not strictly necessary as we can approximate these values using finite differences.

Using $e = e(\rho, P)$

Let us first present the method using $e = e(\rho, P)$. The residual for solving (5a)-(5c) for ρ_L, P_L, D is thus:

$$\mathbf{F_e}(\rho, P, D) = \begin{pmatrix} \rho - \rho_0 \left(1 - \frac{u_0}{D}\right)^{m+1} \\ P - P_0 + \rho u_0 D \\ e(\rho, P) - e_0 - \frac{1}{2}u_0^2 + \frac{u_0 P_0}{\rho D} \end{pmatrix} \tag{10}$$

Remark 2.5. The careful reader may have noticed that the third component contains e_0. At first, this may seem strange since its true value from equation (5c) is e_R, which is dependent on ρ, which may not be constant in higher geometries. That is, when $m = 0$, then $\rho_R = \rho_0$, and so $e_R = e(\rho_0, P_0) = e_0$, and (10) is correct. But, when $m \neq 0$, then $e_R = e(\rho_R, P_0)$, but ρ_R is not constant, and so (10) appears to be incorrect in the third component. However, this is not the case. When $m \neq 0$, table 1 says that $P_0 = 0$ and $e(\rho, P = 0) = \text{const}$ are required. Hence, $e_R = e(\rho_R, P_R) = e(\rho_R, P_0) = e(\rho_R, 0) = \text{const} := e_0$, and so (10) is in fact correct, even in higher dimensions.

Let us compute the derivative (or Jacobian) of $\mathbf{F_e}$:

$$(\mathbf{F_e'})(\rho, P, D) = \begin{pmatrix} 1 & 0 & -\rho_0(m+1)\left(1 - \frac{u_0}{D}\right)^m \frac{u_0}{D^2} \\ u_0 D & 1 & \rho u_0 \\ \frac{\partial e}{\partial \rho}(\rho, P) - \frac{u_0 P_0}{\rho^2 D} & \frac{\partial e}{\partial P}(\rho, P) & -\frac{u_0 P_0}{\rho D^2} \end{pmatrix} \tag{11}$$

Using $P = P(\rho, e)$

Let us now present the method using $P = P(\rho, e)$. Now the residual for solving (5a)-(5c) for ρ_L, e_L and D is given by:

$$\mathbf{F_p}(\rho, e, D) = \begin{pmatrix} \rho - \rho_0 \left(1 - \frac{u_0}{D}\right)^{m+1} \\ P(\rho, e) - P_0 + \rho u_0 D \\ e - e_0 - \frac{1}{2}u_0^2 + \frac{u_0 P_0}{\rho D} \end{pmatrix}. \tag{12}$$

The Jacobian of $\mathbf{F}$ is then given by:

$$\mathbf{F_p'}(\rho, e, D) = \begin{pmatrix} 1 & 0 & -\rho_0(m+1)\left(1 - \frac{u_0}{D}\right)^m \frac{u_0}{D^2} \\ \frac{\partial P}{\partial \rho}(\rho, P) + u_0 D & \frac{\partial P}{\partial e}(\rho, e) & \rho u_0 \\ -\frac{u_0 P_0}{\rho^2 D} & 1 & -\frac{u_0 P_0}{\rho D^2} \end{pmatrix} \tag{13}$$

Remark 2.6. The important observation is this method is completely agnostic to the equation of state, initial conditions, and geometry. At no points have we assumed that the EoS has any form or properties, nor do we insist on any special initial conditions. Thus, this method permits us to find semi-analytic solutions (when they are permissible by theory) for arbitrary equations of state, with arbitrary initial conditions, in arbitrary geometries.

2.1.6 Simplified two variable residual

In this section, we present further assumptions that can be made to simplify the problem. The key idea is that assuming $P_0 = m = 0$ allows us to reduce the jump conditions from three to two equations and no longer in terms of the shock speed. Doing so has several advantages. First, as we are only solving for two variables, it is easier to find the solution. Second, a good initial guess can be easier to supply using physical considerations, whereas the shock speed may be difficult to estimate. Thirdly, the inverse of the Jacobian can be computed explicitly.

Proposition 2.7. *The solution of the Noh Problem when $P_0 = 0$, $m = 0$ is completely determined by ρ_L, P_L.*

Proof. We follow lemma 2.6 in Gerberding and Musick [2023]. As per corollary 2.2, we only need to show that the Rankine-Hugoinot conditions are completely determined by ρ_L and P_L. If $P_0 = P_R = 0$, then (5b) simplifies to

$$\frac{P_L}{\rho_L} = -D u_0. \tag{14}$$

Now, using $u_L = P_R = 0$ and $\rho_R = \rho_0$, equation (2b) simplifies to

$$P_L = (u_0 - D)\rho_R u_0 = (u_0 - D)\rho_0 u_0. \tag{15}$$

Hence, using (14), we obtain:

$$P_L - \rho_0 u_0^2 - \rho_0 \frac{P_L}{\rho_L} = 0. \tag{16}$$

Similarly, (5c) simplifies to

$$e_L = e_R + \frac{1}{2}u_0^2 = e_0 + \frac{1}{2}u_0^2. \tag{17}$$

In short, (16)- (17) constitute the jump conditions. Therefore, once they have been solved for ρ_L, P_L, the first equation in this proof provides D, and thus we have the requisite information to construct the solution to the Noh Problem. □

We can now use (16) and (17) as a simplified Rankine-Hugoinot jump condition. Below are the corresponding residual functions.

Using $e = e(\rho, P)$

If we use $e = e(\rho, P)$ to solve (16) - (17) for ρ_L, P_L, we obtain the following residual function:

$$\mathbf{F_e^s}(\rho, P) := \begin{pmatrix} P - \rho_0 u_0^2 - \rho_0 \frac{P}{\rho} \\ e(\rho, P) - e_0 - \frac{1}{2}u_0^2 \end{pmatrix}. \tag{18}$$

The Jacobian is given by:

$$(\mathbf{F_e^s})'(\rho, P) := \begin{pmatrix} \frac{P}{\rho^2}\rho_0 & 1 - \frac{1}{\rho}\rho_0 \\ \frac{\partial e}{\partial \rho}(\rho, P) & \frac{\partial e}{\partial P}(\rho, P) \end{pmatrix} \tag{19}$$

with determinant

$$\det((\mathbf{F_e^s})')(\rho, P) = \frac{\partial e}{\partial P}(\rho, P)\frac{P}{\rho^2}\rho_0 - \frac{\partial e}{\partial \rho}(\rho, P)(1 - \frac{\rho_0}{\rho}). \tag{20}$$

The inverse of the Jacobian is thus given by:

$$((\mathbf{F_e^s})')^{-1}(\rho, P) = \frac{1}{\frac{\partial e}{\partial P}(\rho, P)\frac{P}{\rho^2}\rho_0 - \frac{\partial e}{\partial \rho}(\rho, P)(1 - \frac{\rho_0}{\rho})} \begin{pmatrix} \frac{\partial e}{\partial P}(\rho, P) & -(1 - \frac{1}{\rho}\rho_0) \\ -(\frac{\partial e}{\partial \rho}(\rho, P)) & \frac{P}{\rho^2}\rho_0 \end{pmatrix}. \tag{21}$$

Using $P = P(\rho, e)$

If we use $P = P(\rho, e)$ to solve (16)-(17), but this time for ρ_L, e_L, we obtain the following residual function:

$$\mathbf{F_p^s}(\rho, e) := \begin{pmatrix} P(\rho, e) - \rho_0 u_0^2 - \rho_0 \frac{P(\rho, e)}{\rho} \\ e - e_0 - \frac{1}{2}u_0^2 \end{pmatrix}. \tag{22}$$

The Jacobian is given by:

$$(\mathbf{F_p^s})'(\rho, e) := \begin{pmatrix} \frac{\partial P}{\partial \rho}(\rho, e) - \rho_0 \left(\frac{\frac{\partial P}{\partial \rho}(\rho, e)\rho - P(\rho, e)}{\rho^2} \right) & \frac{\partial P}{\partial e}(\rho, e) - \rho_0 \frac{\partial P}{\partial e}(\rho, e)\frac{1}{\rho} \\ 0 & 1 \end{pmatrix} \tag{23}$$

with determinant

$$\det((\mathbf{F_p^s})')(\rho, e) = \frac{\partial P}{\partial \rho}(\rho, e) - \rho_0 \left(\frac{\frac{\partial P}{\partial \rho}(\rho, e)\rho - P(\rho, e)}{\rho^2} \right). \tag{24}$$

The inverse of the Jacobian is thus given by:

$$((\mathbf{F_p^s})')^{-1}(\rho, e) = \frac{1}{\frac{\partial P}{\partial \rho}(\rho, e) - \rho_0 \left(\frac{\frac{\partial P}{\partial \rho}(\rho, e)\rho - P(\rho, e)}{\rho^2} \right)} \begin{pmatrix} 1 & -(\frac{\partial P}{\partial e}(\rho, e) - \rho_0 \frac{\partial P}{\partial e}(\rho, e)\frac{1}{\rho}) \\ 0 & \frac{\partial P}{\partial \rho}(\rho, e) - \rho_0 \left(\frac{\frac{\partial P}{\partial \rho}(\rho, e)\rho - P(\rho, e)}{\rho^2} \right) \end{pmatrix}. \tag{25}$$

Remark 2.8. The two variable method just described is not as general as the three variable method described in the previous subsection. The two variable method assumes the problem is in planar geometry and the initial pressure is zero. However, it is still agnostic to the equation of state and the other initial variables. Indeed, as we discuss in more detail later, the two variable and three variable methods can be used in conjunction to find a semi-analytic solution to a realistic problem. To preview, we use the simplified two variable method to find the density and pressure (or energy), and using these values we can find the shock speed (see (14)). With all three variables, we can now change the initial pressure to something nonzero (but small) and use the previous answers as an initial guess. The idea is that we can "walk" our semi-analytic solution up to a case where the initial conditions are realistic.

2.2 Analysis via Self-Similar Transformation

One approach this report utilizes is to conduct error analysis using the self-similar nature of the Noh Problem. As noted earlier, the theoretical approach used to find the analytic solution is to cast the problem in a self-similar variable:

$$\xi = \xi(x,t) = \frac{x}{t}. \tag{26}$$

The solution to Noh Problem, but in self-similar variables, is then given by:

$$\rho(\xi) = \begin{cases} \rho_L & \xi < D \\ \rho_0 \left(1 - \frac{u_0}{\xi}\right)^m & \xi > D \end{cases} \tag{27a}$$

$$P(\xi) = \begin{cases} P_L & \xi < D \\ P_0 & \xi > D \end{cases} \tag{27b}$$

$$u(\xi) = \begin{cases} 0 & \xi < D \\ u_0 & \xi > D \end{cases} \tag{27c}$$

The basic idea for error analysis is as follows: as the simulation runs in time, the shock location moves. As it moves, more and more cells are located *behind* the shock. But, in *self-similar* variables, the shock (which is really a discontinuity) remains stationary. Thus, as time progresses in (x,t) coordinates, we are adding more cells behind the shock in self-similar variables. This refinement will occur both in the shocked and unshocked region, and thus a single simulation will provide multiple levels of refinement that approximate the self-similar solution, and so a *single* simulation provides a convergence study.

A single simulation is run, and a set of times are selected: $0 < t_1, t_2, \ldots, t_n < T_F$. Then, at each time, we take the approximation in the computation domain, $\mathbf{u}(x,t_j)$ and produce the approximation in the self-similar domain, $\mathbf{u}_s^j(\xi)$, via:

$$\mathbf{u}_s^j(\xi) = \mathbf{u}(\xi \cdot t_j, t_j) \tag{28}$$

We then compare $\mathbf{u}_s^j$ to the exact solution given in (27a)-(27c).

2.2.1 Choosing a self-similar domain

Here we formulate how to set up a simulation that provides sufficient information to produce a self-similar analysis. We assume that the computational domain is on $[0, x_R] \times [0, T_F]$, with T_F to be determined. We assume the user knows the shock speed D. (This is another benefit of our solver: it provides the shock speed before running the simulation.) We denote $[\xi_L = 0, \xi_R]$ as the self-similar domain, and $\xi_D = D$ is the shock location in self-similar coordinates. (Recall that the shock in self-similar coordinates is stationary.)

The question now is two-fold: what is an appropriate self-similar domain to accurately capture the hydrodynamics, and what is an appropriate final time so that the simulation domain remains valid when transformed into self-similar coordinates? Put more succinctly, how we do choose ξ_R and T_F wisely? First, using (26), the final time and computational domain, matching the computational domain to the self-similar domain at final time (that is, $\xi([0, x_R] \times T_F) = [0, \xi_R]$), the self-similar domain is determined by:

$$\xi_R = \frac{x_R}{T_F}. \tag{29}$$

Thus, for a valid self-similar domain, we require $\xi_D < \xi_R$. Now, that does not mean this is an appropriate self-similar domain: for example, if ξ_D is very close to ξ_R, then the analysis is not sufficiently capturing the behavior of the unshocked region. On the other hand, if ξ_D is close to 0, then we are not capturing the behavior of the shocked region. The user will have to make a judgement call as to what is an appropriate domain. For example, if we want ξ_D to be in the middle of self-similar domain, then the formula for the final time is:

$$T_F = \frac{x_R}{2D}. \tag{30}$$

Then, once T_R is calculated, ξ_R is simply calculated via $\xi_R = \frac{x_R}{T_F}$, and this is the self-similar domain.

3 Singularity-EoS

In this section, we survey the `singularity-eos` equation of state library Peterson et al. [2022]. We do not go into the details of the library; instead, we only highlight the verification questions pertaining to `singularity`.

The `singularity-eos` equation of state library is an open source library dedicated to providing access to equations of state and mixed cell closure models in a GPU-friendly way. While it contains implementations for Mie-Gruneisen equations of state, it also allows interfaces with tabulated equations of state, all in an efficient, hydrodynamic-algorithmic agnostic way. As previously mentioned, one objective of this project to provide verification for this library. One such verification test, and previewed in the introduction, is to test the library's ability to solve the Noh Problem's jump conditions. This is a *hydro-independent* problem, and so serves as a test that is more extensive than point-queries but not as extensive as running hydro problems. The second such test is to run the more traditional hydro problems—such as the Noh Problem—and ensure that `singularity` is producing the correct solution. Section 7 provides the verification results.

First, `singularity` exists with a set of python bindings. Indeed, everything can be done via python. Our code has a wrapper for `singularity-eos` EoS objects that can be used with pre-set residual functions and Newton solvers. Second, `singularity` does not have calls for internal energy given density and pressure. What it does contain is a call for pressure given density and internal energy (the call is `PressureFromDensityInternalEnergy`). To keep things as modular as possible, we have implemented a python energy inverter that utilizes the `brentq` iterative solver. However, it should be noted that this energy inverter is *highly* sensitive to the EoS and initial conditions. Indeed, for tabulated EoSs, it may not even be able to find the initial energy. For this reason, we highly suggest using the $P = P(\rho, e)$ version of the jump conditions and the corresponding residual given in (12).

Remark 3.1 ($P(\rho, e)$ vs. $e = e(\rho, P)$)**.** There is a benefit to using the $P = P(\rho, e)$ form over the $e = e(\rho, P)$ form. Consider the Mie-Gruneisen equation of state. Its $e = e(\rho, P)$ form is given by:

$$e(\rho, P) = \frac{P - P_\infty}{\rho\Gamma} + e_\infty(\rho) \tag{31}$$

Thus, in order for the energy to be bounded—often a physical requirement—$\rho \to 0$ implies that $P \to P_\infty$. Essentially, as density gets small, the domain for pressure also gets small. Thus, the set of admissible points shrinks.

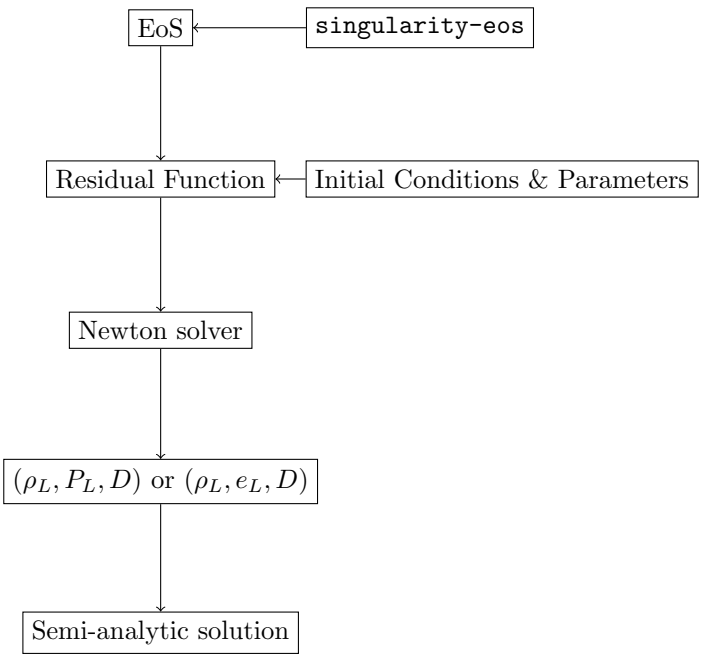
On the other hand, its $P = P(\rho, e)$ from is given by:

$$P(\rho, e) = \rho\Gamma(e - e_\infty(\rho)) + P_\infty(\rho). \tag{32}$$

In this formulation, there is no "squeezing": all restrictions are captured in the formulation, not in the domain.

4 Software

Let us outline some of the software that was developed as an implementation of the above technique. The code is completely python and was designed to be modular. All files can be found in an online repository at `https://re-git.lanl.gov/gerberding/xrage_verification`. The programatic scheme is outlined below:



4.1 EoS

The scripts pertaining to equations of state are found in the `eos_files` directory. It contains (at the time of writing) two scripts: `energy_solver.py` and `eos_library.py`. The main file is the `eos_library` (we will address the energy solver in a moment–it containers wrappers specifically for `singularity-eos`). Each equation of state class is a child class of the parent `equation_of_state`, containing member functions various forms of the EoS and partial derivatives. As of now, there are exact implementations for analytic equations of state: ideal gas (`ideal_gas_eos`), stiffened gas (`stiffened_gas_eos`), Noble-Abel (`noble_abel_eos`), and Carnahan-Starling (`carnahan_starling_eos`).

However, there are two other important classes. The first is the `steinberg` EoS base class (which is a child class of the more general `generic_mie_gruneisen` class). This class is itself a parent class for equations of state stemming from Steinberg et al. [1996]; we include this class because they are a key feature in `singularity`. It thus allows us to test `singularity`'s implementation of Steinberg EoSs, which we do in section 7. These child classes are simple to implement: call the `steinberg` parent class and set the appropriate parameters (for example, found in Steinberg et al. [1996]). We have an example class created for aluminum (6061-T6): `aluminum`.

The second important class is the `derived_singularity_eos` class. This class is the wrapper for `singularity-eos`, taking the information from `singularity` and converting it into information required by the residual functions. This wrapper includes calls for $e = e(\rho, P)$ and $P = P(\rho, e)$; note that all relevant partial derivatives are computed using finite differences.

Let us quickly address the `energy_solver.py` file, as it is required to fully understand the `derived_singularity_eos` class. As of now, `singularity` does not support calls for $e(\rho, P)$. However, it does support calls for $P(\rho, e)$. The energy solver contains a class—`PressureFromDensityEnergyInverter`—that inverts P for e. That is, given ρ, P, it uses $P(\rho, e)$ provided by `singularity` to solve for $e(\rho, P)$.

Returning to the `derived_singularity_eos` class, it uses the energy inverter to compute $e(\rho, P)$, which allows the user to call $e(\rho, P)$. It also uses this inversion to compute finite differences to approximate the partial derivatives. The key point, however, is that this is the *only* wrapping needed for `singularity`. The residual functions only need to accept the wrapped EoS. Any future wrappings for other EoS interfaces can (and should) be done at this stage: write a wrapper as a child class of `equation_of_state`, and the rest of the code remains unchanged.

4.2 Residual functions

The residual functions used for the Newton iteration are found in the `functions` directory in the `noh_functions.py` file. All residual functions are child classes of the parent class `newton_solver_function`. The software is structured so that the Newton solver accepts instances of this class and uses member methods. We have four residual functions corresponding to the residual functions in equations (10) (`generic_noh_function`), (18) (`eos_noh_function`), (12) (`pressure_noh_function`), and section (22) (`simplified_pressure _noh_function`). The `eos_noh_function` and `simplified_pressure_noh_function` assume $m = P_0 = 0$. All of these classes are initialized using an `equation_of_state` object (see above) and a dictionary with the initial conditions of the problem (density, pressure, velocity, symmetry). Once initialized, the function is then passed to the Newton solver.

4.3 Newton solver

The various Newton solvers available can be found in the `solvers` directory. There is only one file–`newton_solvers.py`. The primary class is the `newton_solver` class. The initialization is default, leaving the particulars to the user to set. This class is designed to accept a `newton_solver_function` class using `newton_solver.set_new_function()`. The user then provides an initial guess: `newton_solver.set_new_initial_guess()`. Tolerance is set with `newton_solver.set_new_tolerance()`, and number of allowed iterations with `newton_solver.set_new_max_iteration()`. (The tolerance and iteration max have set defaults; it will throw an error if no function or initial guess is provided.) The key method is `newton_solver.solve()`. This executes the Newton iteration. If it finds a solution, it returns it as a dictionary with the solution, initial guess, number of iterations used, the achieved x error, and achieved residual error. If the iteration runs into problems—nonzero determinant, division by zero, etc.—it throws an error informing the user of the problem and exits the iteration. If it reaches the maximum number of iterations, it throws an error and exits the iteration.

`newton_solver.solve()` has several output options. There is a `verbose = True` option that prints the iteration data to the terminal. There is also an output file option, `output_file = file.txt`, which prints the verbose information to a `file.txt` file in the current directory.

Remark 4.1. Hypothetically, if an user wanted to use the Newton solver for other purposes, doing so would be a simple matter of creating a child class of `newton_solver_function` where `F, F_prime, F_prime_inv` are defined for their specific purpose. The Newton Solver is designed to run *any* Newton iteration—it is not designed with the Noh Problem specifically in mind. Indeed, it is programmed to be agnostic to the problem dimension.

Remark 4.2. A natural question arises: between the 2-and-3 variable functions, which is appropriate for which problems? The answer is simple, with a small aside: the three variable residual functions are appropriate for *all* settings. It is designed to iterate in *any geometry* with *any initial conditions* and with *any equation of state*. The aside is that, if for some reason the three variable function is struggling, the two variable function is a minimal alternative. The user has to change their setting to $m = 0$ and $P_0 = 0$. But the heuristic is that the two variable function has to solve for less, and so perhaps it can solve the simplified problem. Indeed, we encountered this problem when using a Steinberg EoS and tabulated EoS: we struggled with three variable solver, but were able to get a solution to the two variable problem. Using that solution, we were able to find a solution to the three variable problem. We summarize this idea in table 2:

Setting:	Use residual function:
Planar ($m = 0$) with $P_0 = 0$; any EoS	`eos_noh_function` or `simplified_pressure_noh_function`
Any geometry; any initial conditions; any EoS	`generic_noh_function` or `pressure_noh_function`

Table 2: Appropriate settings for each residual function.

4.4 Example Code

Here we provide an example using the above classes based upon $e = e(\rho, P)$. All three instances solve the same problem: planar Noh using ideal gas and standard initial conditions. However, this example shows how to wrap `singularity-eos` as well as how to use the analytic EoSs from the library. All methods produce the correct solution.

```
from functions import generic_noh_function, eos_noh_function
from solvers import newton_solver
from eos_files import ideal_gas_eos, derived_singularity_eos,
from singularity_eos import IdealGas

# Set initial conditions for the Noh Problem
conditions = {'velocity': -1, 'density': 1, 'pressure': 0, 'symmetry': 0}

# Parameters for singularity-eos
gm1 = 5./3. - 1
Cv = 2

# Singularity EoS object
sing_ideal_gas = IdealGas(gm1, Cv)

#Singularity EoS object Wrapper from eos_library.py
sing_eos = derived_singularity_eos(sing_ideal_gas)

#Analytic EoS object from eos_library.py
eos = ideal_gas_eos() # Analytic EoS

#Initial guess for the 2D resdiual function
rho_P_guess = [5, 3]

#Initial guess for the 3D residual function
rho_P_D_guess = [5, 3, 0.5]

# Using analytic EoS, 2D residual function
eos_function = eos_noh_function(conditions, eos)
eos_solver = newton_solver()
eos_solver.set_function(eos_function)
eos_solver.set_new_tolerance(1.0e-08)
eos_solver.set_new_initial_guess(rho_P_guess)
eos_solver.set_new_max_iteration(100)
print(eos_solver.solve(verbose= False, output_file="eos_solution.txt"))

# Using Singularity-EoS Library, 2D residual function
sing_eos_function = eos_noh_function(conditions, sing_eos)
sing_eos_solver = newton_solver()
```

```
sing_eos_solver.set_function(sing_eos_function)
sing_eos_solver.set_new_tolerance(1.0e-08)
sing_eos_solver.set_new_initial_guess(rho_P_guess)
sing_eos_solver.set_new_max_iteration(100)
print(sing_eos_solver.solve(verbose= False, output_file="sing_eos_solution.txt"))

# Using analytic EoS, 3D residual function
function = generic_noh_function(conditions, eos)
solver = newton_solver()
solver.set_function(function)
solver.set_new_initial_guess(rho_P_D_guess)
solver.set_new_tolerance(1.0e-12)
solver.set_new_max_iteration(100)
print(solver.solve(verbose= False, output_file="ideal_gas.txt"))

# Using Singularity-EoS library, 3D residual function
function_1 = generic_noh_function(conditions, sing_eos)
sing_solver = newton_solver()
sing_solver.set_function(function_1)
sing_solver.set_new_initial_guess(rho_P_D_guess)
sing_solver.set_new_tolerance(1.0e-10)
sing_solver.set_new_max_iteration(100)
print(sing_solver.solve(verbose= False, output_file= "sing_ideal_gas.txt"))
```

This script produces the following output:

1. In the terminal:

```
$ python example_newton_solver.py
{'solution': array([4.        , 1.33333333]), 'initial_guess': [5, 3], 'number_of_iterations': 6,
'residual_achieved': 1.7763568394002505e-15, 'error_achieved': 1.1102230246251565e-16}
{'solution': array([4.        , 1.33333333]), 'initial_guess': [5, 3], 'number_of_iterations': 6,
'residual_achieved': 2.2315206618374916e-15, 'error_achieved': 1.1102230246251565e-16}
{'solution': array([4.        , 1.33333333, 0.33333333]), 'initial_guess': [5, 3, 0.5],
'number_of_iterations': 6, 'residual_achieved': 8.899114524108741e-16, 'error_achieved': 0.0}
{'solution': array([4.        , 1.33333333, 0.33333333]), 'initial_guess': [5, 3, 0.5],
'number_of_iterations': 6, 'residual_achieved': 1.355199909595506e-15,
'error_achieved': 5.551115123125783e-17}
```

2. In the current directory:
   - `eos_solution.txt`
   - `sing_eos_solution.txt`
   - `ideal_gas.txt`
   - `sing_ideal_gas.txt`

5 Examples

In this section, we show the versatility and applicability of our approach in solving the Noh Problem. In particular, we give the solution to the jump conditions for multiple equations of state ranging in complexity. When applicable, we provide solutions in multiple dimensions with non-standard initial conditions.

For the purposes of this section, we utilize an exact implementation of the equations of state. That is, we are *not* using `singularity`. Instead, we code the exact equations of state and the exact requisite partial derivatives and use those to execute the Newton iteration.

The tables in each subsection give the initial conditions used, the solution found by the solver, the initial guess, the $x-$error–which denotes the final value of $\|x_{n+1} - x_n\|$, the residual error–which denotes the final value of $\|\mathbf{F}(x_{n+1})\|$, and the number of iterations it took to reach the solution.

5.1 Application to Ideal Gas

The first equation of state we utilized was the polytropic ideal gas equation of state. Its $e = e(\rho, P)$ form is:

$$e(\rho, P) = \frac{P}{\rho(\gamma - 1)} \quad \gamma := \frac{5}{3} \tag{33}$$

We used the $e = e(\rho, P)$ version of our solver, i.e., we used the residual given in equation (10). We selected a tolerance of 1.0E-01. The solution vectors denote the values found for $\big(\rho_L, P_L, D\big)$.

ρ_0	P_0	u_0	m	Solution (ρ_L, P_L, D)	Initial Guess	x-error	Residual Error	# of Iterations
1	0	-1	0	$(4, 1.\bar{3}, 0.\bar{3})$	$(5, 3, 0.5)$	8.90E-16	0.00	6
1	0	-1	1	$(16, 5.\bar{3}, 0.\bar{3})$	$(5, 3, 0.5)$	4.99E-14	0.00	7
1	0	-1	2	$(64, 21.\bar{3}, 0.\bar{3})$	$(5, 3, 0.5)$	1.75E-11	0.00	8
5	3	-2	0	$(15, 33, 1)$	$(5, 3, 0.5)$	1.78E-14	5.55E-16	8
10	0	-2	2	$(6.4\text{E}02, 8.5\bar{3}\text{E}02, 0.\bar{6})$	$(5, 3, 0.5)$	1.14E-13	1.61E-13	47

Table 3: EoS: Solutions to the jump conditions using the ideal gas equation of state (33).

5.2 Application Stiff Gas

Next we used the stiffened-gas equation of state. Its $e = e(\rho, P)$ form is:

$$e(\rho, P) = \frac{P - c_s^2(\rho - \rho_\infty)}{\rho(\gamma - 1)} \quad \gamma := \frac{5}{3}. \tag{34}$$

Observe that, per Table 1, this EoS is only valid in the Noh Problem for $m = 0$. Thus, we only solve with $m = 0$. As in the ideal gas case, we use the residual given in (10). We selected a tolerance of 1.0E-10. The solution vectors denote the values found for $\big(\rho_L, P_L, D\big)$. We used $c_s := \sqrt{\gamma}$ and $\rho_\infty = 1$.

ρ_0	P_0	u_0	m	Solution (ρ_L, P_L, D)	Initial Guess	x-error	Residual Error	# of Iterations
1	0	-1	0	$(1.8931498239, 2.1196329812, 1.1196329812)$	$(3, 1.5, 2)$	4.97E-16	5.24E-16	7
3	1	-2	0	$(8.8082886929, 19.1980390272, 1.0330065045)$	$(10, 10, 5)$	2.22E-16	3.97E-15	10
5	3	- 10	0	$(19.5662418895, 674.6297188364, 3.4325943767)$	$(20, 10, 5)$	1.12E-11	4.54E-15	6
10	10	-10	0	$(38.8230631583, 1356.944387732, 3.4694438773)$	$(20, 10, 5)$	3.55E-13	7.22E-15	19

Table 4: EoS: Solutions to the jump conditions using the stiffened gas equation of state (34).

Remark 5.1. In Ramsey et al. [2017], Burnett et al. [2017], the authors do give a closed form solution for stiff gas. However, one of their assumptions was that $\rho_0 = \rho_\infty$. Our solver does not rely on such an assumption (see the second row of table 4: $\rho_0 \neq \rho_\infty$). Thus, even in equations of state that have been heavily studied, our solver expands previous work and provides a wider range of verification capabilities.

5.3 Application to Noble-Able

Next, we apply our approach to the Noble-Able equation of state, given by:

$$e(\rho, P) = \frac{P(1 - b\rho)}{\rho(\gamma - 1)}; \quad \gamma := \frac{5}{3} \tag{35}$$

This equation of state is admissible for the Noh Problem in all geometries; however, solving in cylindrical and spherical geometries can prove challenging. For our case, we used $b = 0.01$; the residual given in (10) was used. The solution vectors denote the values found for $\big(\rho_L, P_L, D\big)$.

ρ_0	P_0	u_0	m	Solution (ρ_L, P_L, D)	Initial Guess	x-error	Residual Error	# of Iterations
1	0	-1	0	$(3.8834951456, 1.3468013468, 0.3468013468)$	$(5, 3, 0.5)$	4.47E-016	2.22E-16	6
1	1	-1	0	$(1.8808720844, 3.1352386092, 1.1352386092)$	$(10, 10, 5)$	0.00	1.11E-16	7
1	0	-1	1	$(13.0263855224, 4.992466432, 0.3832579977)$	$(20, 15, 5)$	5.62E-15	1.78E-15	38
1	0	-1	2	$(29.8879859818, 14.2096360138, 0.4754296935)$	$(20, 15, 5)$	1.12E-14	1.08E-14	54
5	0	-3	2	$(57.9139303375, 412.8249380511, 2.3760831268)$	$(20, 15, 5)$	1.14E-13	5.73E-14	77

Table 5: EoS: Solutions to the jump conditions using the Noble-Able equation of state (35).

5.4 Application to Carnahan-Starling

Now we apply our method to the Carnahan-Starling equation of state:

$$e(\rho, P) = \frac{P}{\rho Z(\rho)(\gamma - 1)}; \quad Z(\eta) := \frac{1 + \eta + \eta^2 - \eta^3}{(1 - \eta)^3}; \quad \eta(\rho) = b\rho; \quad \gamma := \frac{5}{3} \tag{36}$$

Solutions using (36) have been studied in the past; see Burnett et al. [2017]. However, only the planar case has been analyzed, and even then only an implicit solution was derived, thereby requiring a solver. We used $b = 0.01$ and the residual given in (10).

ρ_0	P_0	u_0	m	Solution (ρ_L, P_L, D)	Initial Guess	x-error	Residual Error	# of Iterations
1	0	-1	0	$(3.5918818886, 1.3858200501, 0.3858200501)$	$(5, 3, 0.5)$	8.49E-11	5.15E-11	64
1.5	0.2	-1	0	$(4.1633168433, 2.5448112382, 0.5632074921)$	$(4, 3, 0.5)$	4.85E-11	5.77-13	8
1	0	-1	1	$(9.2359068339, 4.5294847, 0.4904212203)$	$(4, 3, 0.5)$	8.17E-11	2.98E-13	11
1.1	0	-1.05	1	$(9.8192441455, 5.4462459744, 0.5282383095)$	$(9.9, 5.44, 0.53)$	9.65E-11	2.92E-12	136

Table 6: EoS: Solutions to the jump conditions using the Carnahan-Starling equation of state (36).

Remark 5.2. In general, Carnahan-Starling appear to be *extremely* sensitive to the initial guess. The last row in table 6 shows an initial guess quite close to the true solution; 136 iterations were still required to converge. Indeed, attempting to increase the velocity to -1.06 would not converge with three thousand iterations with the same initial guess.

5.5 Application to Steinberg Equations of State

Next, we apply our method to following class of Steinberg (sometimes called Mie-Gruneisen or simply Gruneisen) equations of state. These EoSs are designed to model properties of more practical materials (such as aluminum). We refer the reader to Steinberg et al. [1996] for further details. These equations of state are of particular interest because `singularity-eos` utilizes them. As far as we are aware, there has been no attempt to derive a semi-analytic solution to the Noh Problem when a Steinberg EoS is used. Following the documentation for `singularity-eos` Security [2024], the equations take the following $P = P(\rho, e)$ form:

$$P(\rho, e) = P_H(\rho) + \rho\Gamma(\rho)(e - e_H(\rho)) \tag{37}$$

with:

$$\eta(\rho) = 1 - \frac{\rho_0}{\rho}; \quad \Gamma(\rho) = \begin{cases} \Gamma_0 & \eta \leq 0 \\ \Gamma_0(1-\eta) + b\eta & 0 \leq \eta < 1 \end{cases}; \quad e_H(\rho) = \begin{cases} 0 & \rho < \rho_0 \\ \frac{\eta(P_H(\rho)+P_0)}{2\rho_0} & 0 \leq \eta < 1 \end{cases};$$
$$P_H(\rho) = \rho_0 + c_0^2\eta \begin{cases} \rho & \rho < \rho_0 \\ \frac{\rho_0}{(1-s_1\eta-s_2\eta^2-s_3\eta^3)^2} & \rho \geq \rho_0 \end{cases}. \tag{38}$$

For our case, we tested using the aluminum (6061-T6) material found in Steinberg et al. [1996]. The parameters we used were (note that these are paramters for the above equation and not the initial conditions): $\rho_0 = 2.703$, $P_0 = 0$, $\Gamma_0 = 1.97$, $b = 0.48$, $c_0 = 0.524 \times 10^6$, $s_1 = 1.4$, $s_2 = s_3 = 0$. (We chose the scaling for c_0 to match the scaling in `singularity-eos`.) For this problem, we used the $P = P(\rho, e)$ formulation in the simplified case; that is, we used the residual function given in equation (22). Also note that this problem can only be posed in planar geometry, i.e. $m = 0$, per table 1. We present two case: one with zero initial pressure–which allows us to solve the easier problem–and one with nonzero initial pressure. We begin with the two variable case.

Since we solving simplified problem, we do not solve for the shock speed, and thus our solution only consists of (ρ_L, e_L).

ρ_0	P_0	u_0	m	Solution (ρ_L, e_L)	Initial Guess	x-error	Residual Error	# of Iterations
2.7	0	$-1.5 \times 0.524 \times 10^3$	0	$(2.7040480683, 155174098.23)$	$(3, 290525)$	8.49E-11	5.15E-11	11

Table 7: EoS: Solutions to the jump conditions using the Steinberg equation of state (37) and simplified pressure residual (22).

Using this information, we are able to increase the complexity of the problem. With ρ_L, e_L, we compute P_L using the EoS. Then, using equation (14), we can compute the shock speed. Now that we know the shock speed, we can modify the initial pressure to a nonzero value and use (ρ_L, P_L, D) as an initial guess for the new iteration. Indeed, using this approach we were able to produce the following solution:

ρ_0	P_0	u_0	m	Solution (ρ_L, e_L, D)	Initial Guess	x-error	Residual Error
2.7	20	$-1.5 \times 0.524 \times 10^3$	0	$(2.7040480683, 15517410.200, 52425.005320)$	$(2.7, 155174098, 524230)$	3.90E-09	8.10E-06

Table 8: EoS:Solutions to the jump conditions using the Steinberg equation of state (37) and pressure residual (12).

One can see a (small) difference in the energy and shock speed results due to the nonzero initial pressure.

Remark 5.3 (Solving non-zero initial pressure problems)**.** These results do represent a proto-type method for solving the Noh Problem with more complicated equations of state. Begin by solving the simplified problem. If this can be done, then the user can use equation (14) to determine the shock speed. Once the shock speed is known, then the user can move to general problem, i.e., use (12) or (10) and change the initial pressure to the practical value.

Remark 5.4 (Computer precision and scales)**.** We did run into a computer precision problem that users should be aware of. If we set the tolerance too low, the iteration would find a fix point ($x_n = x_{n+1}$); however, because the quantities can vary wildly in scale (density is of order two, energy order eight, and shock speed order 5), computer precision will begin to produce errors in the x-error and residual error calculations. Thus, the solver will find a fixed point, but will not realize it's reached the solution.

5.6 Application to Tabulated Equations of State

Our final application involves tabulated equations of state. These EoSs do not have a closed equation form; instead, they are called from a table. To execute this problem, we used `singularity-eos` in a build that incorporated the `EOSPAC` library. Again, to the best of our knowledge, no one has attempted to create a solution to the Noh Problem using a tabulated equation of state. We used the table for aluminum, number 3720. We used the pressure-based residual (12).

ρ_0	P_0	u_0	m	Solution (ρ_L, e_L, D)	Initial Guess	x-error	Residual Error
2.7	0	-786	0	(2.7039485211, 290525.89791, 537467.05202)	$(2.7, 3.0 \times 10^5, 1886.924)$	0.00	2.38E-07
2.7	15	-786	0	(2.7039485211, 290528.54944, 537467.05203	$(2.7, 3.0 \times 10^5, 1886.924)$	0.00	2.38E-07

Table 9: EoS: Solutions to the jump conditions using the aluminum SESAME data and pressure residual (12).

Remark 5.5 (Choosing a good initial guess)**.** Our choice of initial guess was not completely arbitrary. We used information from the Steinberg equation of state (see preivous section) to calculate initial guesses. If we let (ρ_g, e_g, D_g) denote our initial guess, we used $D_g = c_0 + s_1|u_0| - u_0$ and $\rho_g = \rho_0 \frac{D_g - u_0}{D_g}$. The energy guess was arbitrary, but the scale was chosen to match the D_g scale.

6 ExactPack

In this section, we briefly discuss our contribution to `ExactPack` Thrussel et al. [2024], an open source library of exact solutions to a variety of problems. One outcome of this project is the inclusion of the black box EoS Noh solver into the package. This inclusion expands the verification capabilities of `ExactPack`: users can now run verification tests with *any* equation of state, whereas before the Noh Problem in `ExactPack` was hardcoded using ideal gas.

At the time of this writing, the solver is included under `exactpack/solvers/nohblackboxeos`. That directory contains the solver file (`blackboxnoh.py`) and the `solution_tools` subdirectory, which contains the residual function classes and the Newton solver class. The classes (based on the code outlined above) have been refactored to adhere `ExactPack` conventions, including wrapping the `ExactSolver` base class. There is an example in `exactpack/examples` called `nohblackbox.py`.

The long term term goal is to integrate `singularity` into `ExactPack`. Such a combination would create immense verification capabilities: access to a host of equations of state and a solver that can produce semi-analytic solutions.

7 Verification Results

In this section we provide our verification results. We ran two separate tests. The first was solving the jump conditions (5a)-(5c) using `singularity-eos`. The second is traditional hydrodynamics verification, where we interfaced `singularity` with `xRAGE` and performed error analysis.

7.1 Singularity-eos: Solving the jump conditions

In this section, we present verification results for `singularity` based on the jump conditions problem discussed in sections 2 and 3. Our goal is to solve equations (5a)-(5c) using `singularity` as the EoS interface.

To test `singularity`, we use both the $e = e(\rho, P)$ formulation of the jump conditions (that is, use the residual given in (10)) and the $P = P(\rho, e)$ formulation (residual (12)). We compare the solution computed by `singularity` to exact solutions or to solutions computed using an algebraic implementation where all quantities are computed exactly.

Note that when comparing `singularity` to an algebraic EoS implementation, we are testing several components. First, we are testing the finite difference approach used to compute derivatives with `singularity`. Secondly, especially in the $e = e(\rho, P)$ framework, we test the energy inversion process used to compute $e(\rho, P)$. For `singularity`, this is an important test as the inversion is used whenever $e(\rho, P)$ is called, including in finite differences. Thus we are testing the energy inversion *and* finite differences at the same time and seeing if their errors compound in any serious way. In the $P = P(\rho, e)$ framework, the energy inversion is used only to calculate the initial energy; however, it is still critical to test this step, and we can measure how much error (if any) a small difference in initial energy makes in the Newton iteration.

Let us quickly describe the tables. The first column always provides the initial conditions. The column for `singularity` provides the solver information obtained using `singularity`. The column for "Algebraic EoS" is the solver information when the exact algebraic EoS is used (that is, we used the `eos_library.py` files). "x-error" denotes the final value of $\|x_{n+1} - x_n\|$ and "Residual error" denotes the final value of $\|\mathbf{F}(x_{n+1})\|$. "# of Iterations" denotes the number of iterations used to reach the solution. Note that, in each case, we used the same initial guess for both the `singularity`-interfaced solver and the algebraic EoS solver.

7.1.1 Using $e = e(\rho, P)$

In this section, we test `singularity`'s ability to solve the jump conditions when the $e = e(\rho, P)$ framework is used, i.e., using residual (10). These tests examine the effect of the energy inversion approach to calculating $e = e(\rho, P)$ in `singularity` and using finite differences to calculate the partial derivatives. We test using the ideal gas (33) and stiffened gas (34) equations of state.

Ideal Gas

We begin with the ideal gas equation of state. Table 10 reports the initial conditions and the corresponding exact solution.

ρ_0	P_0	u_0	m	Exact Solution (ρ_L, P_L, D)
1	0	-1	0	$[4, 1\frac{1}{3}, \frac{1}{3}]$
1	1	-1	0	$[1.89314982, 3.11963298, 1.11963298]$
2	0	-1	0	$[8, 2\frac{2}{3}, \frac{1}{3}]]$
1	0	-2	0	$[8, 2\frac{2}{3}, \frac{1}{3}]]$
10	3.3	-6.3	0	$[39.0996617, 536.59333821, 2.16497362]$

Table 10: Exact solution for to the jump conditions (5a)- (5c) with ideal gas. Solution in terms of ρ_L, P_L, D.

Table 11 reports the solution found using `singularity` and with our algebraic implementation. The first column reports the initial conditions. The second column reports the solution information found using `singularity-eos`. The third column reports the solution information found using an algebraic EoS implementation. "Solution Err." denotes the error when compared to the solutions in table 10.

ρ_0	P_0	u_0	m	`singularity-eos`				Algebraic EoS			
-	-	-	-	x-error	Residual Error	# of Iter.	Solution Err.	x-error	Residual Error	# of Iter.	Solution Err.
1	0	-1	0	5.00E-16	5.00E-16	6	4.44E-16	5.00E-16	2.48E-16	6	4.44E-16
1	1	-1	0	1.13E-14	2.22E-16	9	2.22E-16	1.22E-14	6.66E-16	9	8.88E-16
2	0	-1	0	4.22E-14	8.95E-16	6	8.88E-16	4.31E-14	9.99E-16	6	0.00
1	0	-2	0	9.93E-16	0.00	7	2.22E-15	9.99E-16	0.00	7	2.22E-15
10	3.3	-6.3	0	1.15E-13	5.41E-15	7	2.42E-13	1.14E-13	1.14E-13	7	2.27E-13

Table 11: EoS: Ideal gas, $e = e(\rho, P)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS and against exact solution. First column: initial conditions. Second column: results for singularity-eos. Third column: results for algebraic eos.

Second, we tested `singularity-eos` still with ideal gas, but in higher geometries ($m = 1, 2$). As it is tedious to analytically derive a solution, we test only against an algebraic implementation of the EoS. The first column gives the initial conditions, the second the results found using `singularity-eos`, and the third column the results found using an algebraic implementation.

ρ_0	P_0	u_0	m	singularity-eos				Algebraic EoS			
-	-	-	-	Solution ($[\rho_L, P_L, D]$)	x-error	Residual Error	# of Iter.	Solution ($[\rho_L, P_L, D]$)	x-error	Residual Error	# of Iter.
1	0	-1	1	$[16., 5.\overline{3}, 0.\overline{3}]$	4.63E-14	1.78E-15	7	$[16., 5.\overline{3}, 0.\overline{3}]$	4.63E-14	1.78E-15	7
2	0	-1	1	$[32., 10.\overline{6}, 0.\overline{3}]$	1.78E-15	3.55E-15	8	$[32., 10.\overline{6}, 0.\overline{3}]$	0.00	0.00	8
1	0	-2	1	$[16., 21.\overline{3}, 0.\overline{6}]$	3.97E-15	1.78E-15	6	$[16., 1.\overline{3}, 0.\overline{6}]$	1.78E-15	1.78E-15	6
1	0	-1	2	$[64., 21.\overline{3}, 0.\overline{3}]$	2.93E-12	2.13E-14	7	$[64., 21.\overline{3}, 0.\overline{3}]$	2.99E-12	3.55E-15	7
2	0	-1	2	$[128., 42.\overline{6}, 0.\overline{3}]$	4.91E-11	1.59E-14	7	$[128., 42.\overline{6}, 0.\overline{3}]$	4.83E-11	1.42E-14	7
1	0	-2	2	$[64., 85.\overline{3}, 0.\overline{6}]$	6.50E-11	2.13E-14	7	$[64., 85.\overline{3}, 0.\overline{6}]$	6.33E-11	2.56E-14	7
6	0	-5.2	1	$[96., 865.28, 1.7\overline{3}]$	2.28E-13	1.14E-13	8	$[96., 865.28, 1.7\overline{3}]$	2.28E-13	1.15E-13	8

Table 12: EoS: Ideal gas, $e = e(\rho, P)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS. First column: initial conditions. Second column: results for singularity-eos. Third column: results for algebraic EoS.

Stiff Gas

Next we test using stiff gas. In the formulation given in (34), we set the paramters as $c_s = \sqrt{\gamma},\ \rho_\infty = 1$. Now `singularity` uses a different form of stiff gas, given by:

$$e(\rho, P) = \frac{P + \gamma P_\infty}{(\gamma - 1)\rho} + q. \tag{39}$$

To ensure that these matched, we used parameters $P_\infty = 1,\ q = -\frac{\gamma}{(\gamma-1)}$. (The reader may verify that these give rise to the same EoS.) Note that, per table 1, the jump conditions with the stiff gas EoS are only meaningful in $m = 0$. We vary the initial conditions to stress both implementations.

ρ_0	P_0	u_0	m	singularity-eos				Algebraic EoS			
-	-	-	-	Solution ($[\rho_L, P_L, D]$)	x-error	Residual Error	Iter.	Solution ($[\rho_L, P_L, D]$)	x-error	Residual Error	Iter.
1	0	-1	0	[1.8931498239, 2.1196329812, 1.1196329812]	4.97E-16	5.24E-16	7	[1.8931498239, 2.1196329812, 1.1196329812]	4.97E-16	2.48E-16	7
2.5	2	-3	0	[7.6742346142, 35.3711730709, 1.4494897428]	7.62E-15	7.11E-15	6	[7.6742346142, 35.3711730709, 1.4494897428]	7.46E-15	7.17E-15	6
5	3	-10	0	[19.5662418895, 674.6297188364, 3.4325943767]	1.14E-13	2.78E-15	8	[19.5662418895, 674.6297188364, 3.4325943767]	1.14E-13	1.14E-13	8
6	10	-10	0	[22.8601317956, 823.5214625627, 3.5586910427]	1.14E-13	1.14E-13	7	[22.8601317956, 823.5214625627, 3.5586910427]	0.00	1.33E-15	6

Table 13: EoS: Stiff Gas, $e = e(\rho, P)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS. First column: initial conditions. Second column: results for singularity-eos. Third column: results for algebraic EoS.

These results demonstrate two critical things. First, that an energy inversion approach used to calculate $e = e(\rho, P)$ does not appear to introduce any serious errors. Second, that using finite differences to compute partial derivatives also does not appear to introduce any serious errors. Indeed, the fact that $\frac{\partial e}{\partial \rho}$, $\frac{\partial e}{\partial P}$ are computed using *both* the energy inversion and finite differences, and yet `singularity` is able to perform just as well as the algebraic implementation (which uses exact partial derivatives) is compelling evidence that `singularity` is able to call equations of state accurately. In particular, `singularity` is accurate even when the particular form of the EoS (in this case $e = e(\rho, P)$) is not directly implemented.

7.1.2 Using $P = P(\rho, e)$

In this section, we test `singularity`'s ability to solve the jump conditions using the $P = P(\rho, e)$ framework. We test several EoS, namely ideal gas and–far more interesting–a Steinberg EoS. This test examines the effects of the energy inversion on the initial energy and the finite difference calculations to approximate the partial derivatives $\frac{\partial P}{\partial \rho}$ and $\frac{\partial P}{\partial e}$.

Ideal Gas

We begin with ideal gas. In planar geometry, it is possible to analytically derive the solution to the jump conditions given the initial conditions. We report those solutions we investigate in table 10. In higher geometries, we do not calculate the exact solution, and instead compare to implicit solution found using an algebraic implementation of the EoS. 'Solution Err." denotes the error when compared to the solutions in table 14.

ρ_0	P_0	u_0	m	Exact Solution (ρ_L, e_L, D)
1	0	-1	0	$[4, 0.5, \frac{1}{3}]$
1	1	-1	0	$[1.89314982, 2.4717797887, 1.11963298]$
2	0	-1	0	$[8, 0.5, \frac{1}{3}]$
1	0	-2	0	$[4, 2, \frac{2}{3}]$
10	3.3	-6.3	0	$[39.0996617, 20.5856002928, 2.16497362]$

Table 14: Exact solution for to the jump conditions (5a)- (5c) with ideal gas.

In this table we report the solution error of the iterative solver found using `singularity` and using an algebraic EoS.

ρ_0	P_0	u_0	m	singularity-eos				Algebraic EoS			
-	-	-	-	x-error	Residual Error	# of Iter.	Solution Err.	x-error	Residual Error	# of Iter.	Solution Err.
1	0	-1	0	5.81E-11	0.00	6	6.66E-16	5.81E-11	0.00	6	6.66E-16
1	1	-1	0	6.29E-11	1.12E-11	14	2.43E-11	6.29E-11	1.12E-11	14	2.43E-11
2	0	-1	0	8.88E-16	0.00	8	1.11E-15	8.88E-16	0.00	8	1.11E-15
1	0	-2	0	0.00	0.00	7	1.33E-15	0.00	0.00	7	1.33E-15
10	3.3	-6.3	0	3.25E-12	1.19E-13	11	9.95E-14	3.25E-12	1.19E-13	11	9.95E-14

Table 15: EoS: Ideal gas, $P = P(\rho, e)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS and against exact solution. First column: initial conditions. Second column: results for singularity-eos. Third column: results for algebraic eos.

To add more complexity, we increase the geometry of the problem.

ρ_0	P_0	u_0	m	singularity-eos			Algebraic EoS			
-	-	-	-	Solution ($[\rho_L, P_L, D]$)	x-error	Residual Error	Iter.	Solution ($[\rho_L, P_L, D]$)	x-error	Residual Error	Iter.
1	0	-1	1	[16., 0.5, $0.\overline{3}$]	6.68E-13	0.00	6	[16., 0.5, $0.\overline{3}$]	6.73E-13	1.78E-15	6
2	0	-1	1	[32., 0.5, $0.\overline{3}$]	7.11E-15	0.00	7	[32., 0.5, $0.\overline{3}$]	1.07E-14	0.00	7
1	0	-2	1	[16., 2, $0.\overline{6}$]	1.86E-15	1.78E-15	6	[16., 2, $0.\overline{6}$]	3.60E-15	1.78E-15	6
1	0	-1	2	[64., 0.5, $0.\overline{3}$]	7.11E-15	0.00	7	[64., 0.5, $0.\overline{3}$]	7.11E-15	0.00	7
2	0	-1	2	[128., 0.5, $0.\overline{3}$]	1.42E-14	0.00	7	[128., 0.5, $0.\overline{3}$]	1.42E-14	0.00	7
1	0	-2	2	[64., 2, $0.\overline{6}$]	1.42E-14	0.00	7	[64., 2, $0.\overline{6}$]	7.11E-15	0.00	7
6	0	-5.2	1	[96., 13.52, $1.7\overline{3}$]	8.53E-14	1.15E-13	6	[96., 13.52, $1.7\overline{3}$]	8.53E-14	1.14E-13	6

Table 16: EoS: Ideal gas, $P = P(\rho, e)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS. First column: initial conditions. Second column: results for singularity-eos. Third column: results for algebraic EoS.

Stiff Gas

Next we test using stiff gas. We use the same parameters as above. Again, we only test with $m = 0$ and vary the initial conditions.

ρ_0	P_0	u_0	m	singularity-eos				Algebraic EoS			
-	-	-	-	Solution ($[\rho_L, e_L, D]$)	x-error	Residual Error	Iter.	Solution ($[\rho_L, e_L, D]$)	x-error	Residual Error	Iter.
1	0	-1	0	[1.8931498239, 0.5, 1.1196329812]	8.02E-11	2.22E-16	7	[1.8931498239, 0.5, 1.1196329812]	8.02E-11	4.44E-16	7
2.5	2	-3	0	[7.6742346142, 4.7393876913, 1.4494897428]	8.22E-11	6.77E-12	12	[7.6742346142, 4.7393876913, 1.4494897428]	8.22E-11	4.44E-16	7
5	3	-10	0	[19.5662418895, 49.3466746953, 3.4325943767]	9.33E-12	1.63E-13	10	[19.5662418895, 49.3466746953, 3.4325943767]	9.33E-12	2.56E-13	10

Table 17: EoS: Stiff Gas, $P = P(\rho, e)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS. First column: initial conditions. Second column: results for singularity-eos. Third column: results for algebraic EoS.

Steinberg EoS

Of particular interest in the use of Steinberg EoSs (37). These EoS provides a strong test because the partial derivatives are complicated. Indeed, the algebraic implementation utilizes several applications of the chain rule. Because of the complexity, we only test with two initial conditions (indeed, *finding* an initial condition that admits a solution is not a simple task). The first set uses $P_0 = 0$, while the second uses $P_0 = 20$.

EoS Interface	Solution (ρ_L, e_L, D)	x-error	Residual Error	# of iterations
Singularity-eos	2.7040480683, 155174098.23, 524250.05318	1.54E-08	6.20E-06	4
Algebraic	2.7040480683, 155174098.23, 524250.05318	3.78E-08	2.86E-06	5

Table 18: EoS: Steinberg EoS, $P = P(\rho, e)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS. First column: EoS interface. Second column: Solution. Third column: x-error. Fourth Column: Residual error. Fourth column: Number of iterations. The parameters we used were: $\rho_{ref} = 2.703$, $P_{ref} = 0$, $\Gamma_{ref} = 1.97$, $b = 0.48$, $c_0 = 0.524 \times 10^6$, $s_1 = 1.4$, $s_2 = s_3 = 0$. The initial conditions for this problem were $\rho_0 = 2.7$, $P_0 = 0$, $u_0 = -1.5 \times 0.524 \times 10^3$. Initial guess: $(\rho_L, e_L, D) = (2.7, 155174098.0, 524230)$

To add more complexity to the problem, we make the initial pressure nonzero; `singularity` performed just as well as the algebraic interface.

EoS Interface	Solution (ρ_L, e_L, D)	x-error	Residual Error	# of iterations
Singularity-eos	2.7040480683, 1.55174102.00, 524250.05320	2.39E-09	5.01E-06	6
Algebraic	2.7040480683, 1.55174102.00, 524250.05320	3.90E-09	8.11E-06	6

Table 19: EoS: Steinberg EoS, $P = P(\rho, e)$ formulation. Comparison of singularity-eos against algebraic implementation of EoS. First column: EoS interface. Second column: Solution. Third column: x-error. Fourth Column: Residual error. Fourth column: Number of iterations. The parameters we used were: $\rho_{ref} = 2.703$, $P_{ref} = 0$, $\Gamma_{ref} = 1.97$, $b = 0.48$, $c_0 = 0.524 \times 10^6$, $s_1 = 1.4$, $s_2 = s_3 = 0$. The initial conditions for this problem were $\rho_0 = 2.7$, $P_0 = 20$, $u_0 = -1.5 \times 0.524 \times 10^3$. Initial guess: $(\rho_L, e_L, D) = (2.7, 155174098.0, 524230)$

These results demonstrate that using finite difference to calculate $\frac{\partial P}{\partial \rho}$ and $\frac{\partial P}{\partial e}$ does not introduce any serious errors. They also demonstrate the using an energy inversion to calculate e_0 neither produces serious errors (this was also tested in the $e = e(\rho, P)$ tests, but these results further confirm).

In general, the above results—both $e = e(\rho, P)$ and $P = P(\rho, e)$—demonstrate that `singularity-eos` is just as capable of solving the Noh Problem jump conditions as a hard-coded, exact EoS interface. Any approximation used in conjunction with `singularity`, i.e., the energy inverter and finite differences, do not produce any significant errors. In summary: `singularity` is just as accurate and effective at solving these jump conditions as an exact EoS implementation, thusly demonstrating its accuracy, ability to invert an EoS, and finite difference capability.

7.2 Hydrodynamic Problems

The goal is to verify `singularity-eos` through the use of hydrodynamics code, namely `xRAGE`. In particular, we want to ensure that `singularity` is producing similar results to when `xRAGE` is using other EoS implementations (such as P-T tables). All error are relative and calculated using the l^1-norm.

7.3 Analysis Using Self-Similar Solution

In this section, we present verification results that are based on the self-similar analysis presented in section 2.2. In all cases, the numerical solution was compared to a semi-analytic solution produced with the technique outline in section 2.

7.3.1 Ideal Gas

Here, we present verification results for when the ideal gas equation of state 33 is used. We compare results for when three EoS interfaces are used: `singularity`, P-T tables, and `xRAGE`'s internal implementation of ideal gas. The problem we ran was the planar Noh Problem ($m = 0$) with initial conditions $\rho_0 = 1,\ u_0 = -1,\ P_0 = 0$. The computational domain was $[0, 2]$ with 8,000 degrees of freedom and a final time of $T = 1.5$. The self-similar domain was $[0, 2 \times \xi_D]$. We report the time from which the data is taken, the number of degrees of freedom *in self-similar coordinates*, the density error, and the convergence rate for each interface.

Time	#Dofs	Singularity Error	Rate	xRAGE Internal Error	Rate	P-T Table Error	Rate
0.25	668	1.69e-03	–	1.69e-03	–	1.73e-03	–
0.5	1335	6.90e-04	1.29	6.90e-04	1.29	7.31e-04	1.24
0.75	2003	4.76e-04	0.88	4.76e-04	0.88	5.03e-04	0.92
1.0	2670	4.23e-04	0.34	4.23e-04	0.34	4.66e-04	0.39
1.25	3337	2.77e-04	1.01	2.77e-04	1.01	3.18e-04	0.83
1.5	4005	2.40e-04	0.72	2.40e-04	0.72	2.69e-04	0.84

Table 20: Self-similar convergence table of the Noh Problem using different EoS interfaces for ideal gas.

7.3.2 Stiff Gas

Here, we present verification results for when the stiffened gas equation of state 34 is used. The parameters we selected were $\rho_\infty = 1$ and $c_s = \sqrt{\gamma}$. We compare results for when two EoS interfaces are used: `singularity` and P-T tables. The problem we ran was the planar Noh Problem ($m = 0$) with initial conditions $\rho_0 = 1,\ u_0 = -1,\ P_0 = 0$. The computational domain was $[0, 2]$ with 8,000 degrees of freedom and a final time of $T = 1.0$. The self-similar domain was $[0, 2]$. We report the time from which the data is taken, the number of degrees of freedom *in self-similar coordinates*, the density error, and the convergence rate for each interface.

Time	#Dofs	Singularity Error	Rate	P-T Table Error	Rate
0.2	1062	4.02E-04	–	4.02E-04	–
0.4	3204	2.08E-04	0.95	2.08E-04	0.95
0.6	4806	1.39E-04	0.99	1.38E-04	1.01
0.8	6407	1.05E-04	0.97	1.05E-04	0.95
1.0	8010	8.69E-05	0.84	8.67E-05	0.86

Table 21: Self-similar convergence table of the Noh Problem using different EoS interfaces for stiff gas.

7.4 Error Analysis

In this section, we present numerical results for the Noh Problem that use different interfaces for the EoS. The analysis here is the standard one, where the simulation is run up to a final time with different levels of spatial refinement, and the error is computed by using an analytic or semi-analytic solution.

7.4.1 Ideal Gas

The problem we ran was the planar Noh Problem ($m = 0$) with initial conditions $\rho_0 = 1,\ u_0 = -1,\ P_0 = 0$. The computational domain was $[0, 2]$ with with final time of $T = 1.25$. The second column compares the errors between the numerical solution computed using `singularity` and the numerical solution computed using `xRAGE`'s internal ideal gas implementation. The third column is the error between the numerical solution computed using `singularity` and the semi-analytic solution. The sixth column is the error between the numerical solution computed using `xRAGE`'s internal ideal gas implementation and the semi-analytic solution. First order convergence is observed in both cases.

# Dofs	Sing. vs. Internal	Sing. vs. Exact	rate	Internal vs. Exact	rate
2000	3.29e-08	7.61e-04	–	7.61e-04	–
4000	2.75e-08	3.40e-04	1.16	3.40e-04	1.16
8000	3.72e-08	1.71e-04	0.99	1.71e-04	0.99
16000	3.98e-08	8.59e-05	0.99	8.59e-05	0.99

Table 22: Convergence table for the planar Noh Problem with ideal gas.

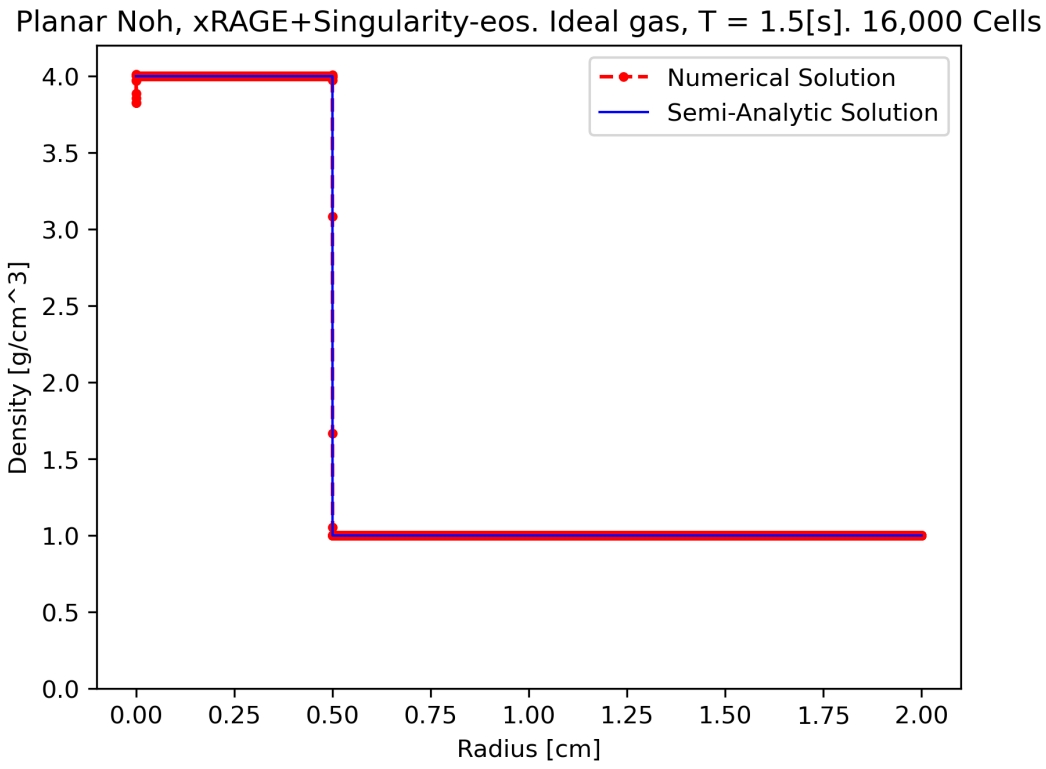


Figure 1: The Planar Noh Problem with the Ideal Gas Equation of State (33) with Singularity-eos library. Semi-analytic solution provided for comparison.

We observe that the first order results are as expected, and serve as verification of both `singularity` as a viable EoS interface and as `xRAGE` as being able to solve this problem.

7.4.2 Stiff Gas

Next we performed a convergence study with the stiff gas equation of state (34). We ran the planar Noh Problem with initial conditions $\rho_0 = 1.01, P_0 = 0.1, u_0 = -1$ on [0,2] with final time $T = 1$. The EoS parameters we selected were $\rho_\infty = 1, c_s = \sqrt{\gamma}$. These results were all computed a semi-analytic solution found using the method detailed in section 2.

# Dofs	Singularity Error	rate	P-T Tables Error	rate
2000	2.97 E-04	–	2.97E-04	–
4000	1.59 E-04	0.90	1.59E-04	0.90
8000	8.69 E-05	0.87	8.67E-05	0.87
16000	5.27 E-05	0.75	5.25E-05	0.72

Table 23: Convergence table for the planar Noh Problem with Stiff gas (34).

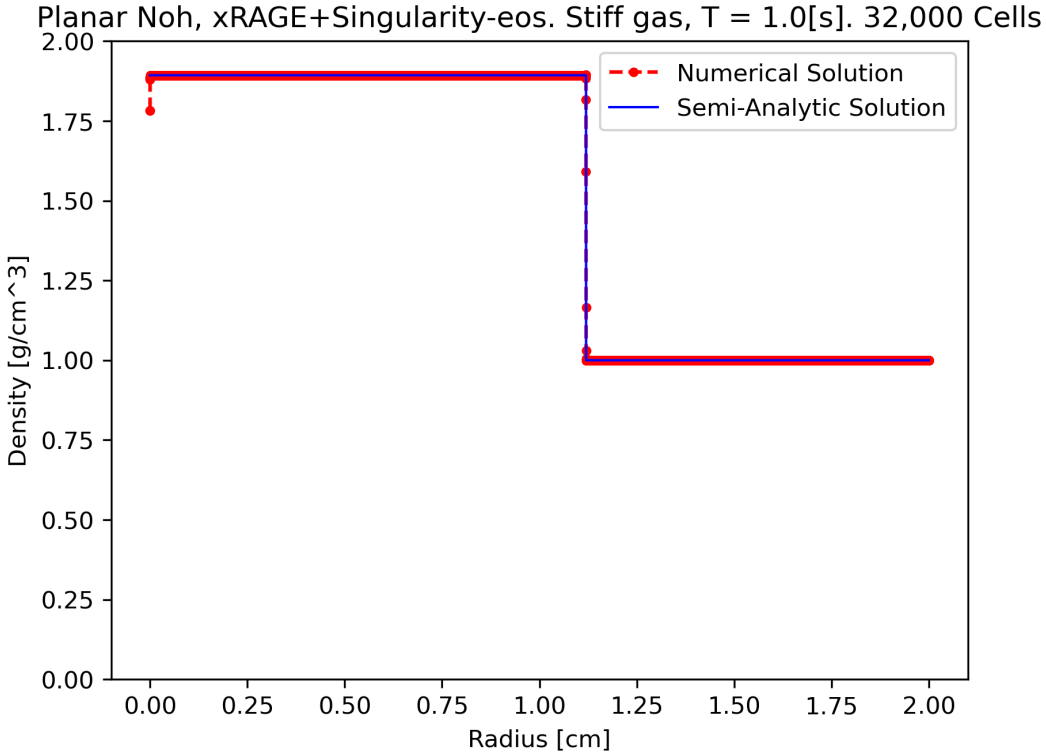


Figure 2: The Planar Noh Problem with the Stiff Gas Equation of State (34) with Singularity-eos library. Semi-analytic solution provided for comparison.

The low convergence rates are not as optimal as expected and warrant further investigation. However, the error is still low and `singularity` is delivering similar errors as the P-T tables. Thus, we may conclude that `singularity`'s implementation is just as accurate as previous EoS interfaces.

8 Conclusion

This report serves several purposes: to expand verification capabilities with regards to equations of state and initial conditions, and to perform verification for the `singularity-eos` equation of state library.

With regards to the first purpose, we detailed a new methodology for the Noh Problem that is agnostic with respect to the equation of state, initial conditions, and other parameters. Using the jump conditions, we are able find the shocked variables and use their values to construct the solution. We detailed two possible approaches that utilize the $e = e(\rho, P)$ and $P = P(\rho, e)$ formulations. We then used the method to solve the Noh Problem for a host of equations of state—ranging from ideal up to tabulated data—on a variety of non-standard initial conditions. The results demonstrate that our method is capable of solving the Noh Problem for many EoSs in all geometries (when permitted). For traditional EoS, such as ideal gas, stiff gas, and Noble-Able, we were able to extend previous work by removing assumptions on the initial conditions, geometry, and other EoS parameters. Of particular interest is our ability to use significantly more complicated equations of state, such as Steinberg-type EoS and `EOSPAC` data structures. As far as we know, this is the first time a solution to the Noh Problem has been obtained using these equations of state. Such progress is important, as these equations of state are used for more realistic problems. As such, we now have the foundation for performing verification on realistic problems, a capability that has been wanting.

With regards to the second purpose, we were able to apply an intermediary verification test (that is, more intensive that point-queries, but not as complicated a hydrodynamics problem) to `singularity` in the form of solving the jump conditions. We tested the accuracy of `singularity`, the reliability of using an energy

inverter, and the feasibility of using finite differences to compute partial derivatives. The results are promising: `singularity` appears to perform just as well as a hard-coded, exact algebraic implementation of equations of state. We can conclude that using EoS inversions (for example, using $P = P(\rho, e)$ to compute $e = e(\rho, P)$) is a viable alternative when the explicit form is not available. Furthermore, finite differences also provide a possible way to calculate partial derivatives is exact forms are not available (we particularly refer to the `EOSPAC` results).

In the context of traditional hydrodynamic verification, `singularity` demonstrated excellent performance in the Noh Problem with ideal gas. It performed just as well as previous EoS interfaces. For the stiff gas EoS, `singularity` also performed just as well as P-T tables.

Future research includes further analysis of the robustness of the algorithm and if particular forms of the EoS give rise to more forgiving Newton iterations. More error analysis ought to be performed, particularly in higher dimensions (when possible). Of particular interest is performing error analysis using a Steinberg and EOSPAC EoS. In the more mathematical vein, generalization of the Noh Problem, where we can remove a lot of the restrictions on the equations of state, would further expand verification capabilities.

Acknowledgements

This work was funded by the Eulerian Applications Project (EAP) and the Cross Cutting Capabilities Project (XCAP) under the Advanced Simulation and Computing (ASC) Program at LANL. Additionally, this work would not have been possible without the guidance and support of the project mentors, Jeff H. Peterson and Eric J. Tovar; SG extends his sincere gratitude and appreciation.

Seth J. Gerberding *is a fifth year mathematics Ph.D. student at Texas A&M university studying numerical analysis. He enjoys chess, reading, rock climbing (specifically bouldering), and telling engineers they need to learn more linear algebra (which has not made him many friends).*

References

Roy A. Axford. Solutions of the noh problem for various equations of state using lie groups. *Laser and Particle Beams*, 18(1):93–100, 2000. doi: 10.1017/S026303460018111X.

Sarah C. Burnett, Kevin G. Honnell, Scott D. Ramsey, and Robert L. Singleton Jr. Verification studies for the noh problem using non-ideal equations of state and finite strength shocks. *Journal of Verification, Validation and Uncertainty Quantification*, 2017.

Seth Gerberding and Benjamin Musick. Verification & validation of a new equation of state framework in xrage. Technical Report LA-UR-23-29866, Los Alamos National Laboratories, Los Alamos, NM, 2023.

Michael Gittings, Robert Weaver, Michael Clover, Thomas Betlach, Nelson Byrne, Robert Coker, Edward Dendy, Robert Hueckstaedt, Kim New, W Rob Oakes, Dale Ranta, and Ryan Stefan. The rage radiation-hydrodynamic code. *Computational Science & Discovery*, 1(1):015005, nov 2008. doi: 10.1088/1749-4699/1/1/015005. URL `https://dx.doi.org/10.1088/1749-4699/1/1/015005`.

W.F Noh. Errors for calculations of strong shocks using an artificial viscosity and an artificial heat flux. *Journal of Computational Physics*, 72(1):78–120, 1987. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(87)90074-X. URL `https://www.sciencedirect.com/science/article/pii/002199918790074X`.

Jeffrey Hammett Peterson, Jonah Maxwell Miller, Anna Mataleena Pietarila Graham, Daniel Alphin Holladay, Christopher Michael Mauney, Richard Felix Berger, and Karen Chung-Yen Tsai. Singularity-eos xcap report. *Los Alamos Natl. Lab.*, 12 2022. doi: 10.2172/1907763. URL `https://www.osti.gov/biblio/1907763`.

S. D. Ramsey, Z. M. Boyd, and S. C. Burnett. Solution of the noh problem using the universal symmetry of the gas dynamics equations. *Shock Waves*, 27(3):477–485, 2017. doi: 10.1007/s00193-016-0670-z. URL `https://doi.org/10.1007/s00193-016-0670-z`.

Triad National Security. Singularity-eos. `https://lanl.github.io/singularity-eos/main/index.html`, 2024. Accessed: 2024-07-30.

Daniel J Steinberg et al. *Equation of state and strength properties of selected materials*. Lawrence Livermore National Laboratory Livermore, 1996.

Jasper Thrussel, Scott Doebling, Robert Singleton, and Nathan Woods. Exactpack. https://github.com/lanl/ExactPack, 2024.

A. L. Velikovich and J. L. Giuliani. Solution of the noh problem with an arbitrary equation of state. *Phys. Rev. E*, 98:013105, Jul 2018. doi: 10.1103/PhysRevE.98.013105. URL https://link.aps.org/doi/10.1103/PhysRevE.98.013105.