blastFoam Theory and User Guide

Version 3.0.0

Jeff Heylmun, Peter Vonk
and Timothy Brewer
Synthetik Applied Technologies
info@synthetik-technologies.com

https://github.com/synthetik-technologies/blastFoam

April 13, 2020

Abstract

blastFoam is an open-source toolbox for simulating detonations based on the OpenFOAM®[1] framework [OpenCFD Ltd., 2018b]. *blastFoam* provides solutions to highly compressible systems including single- and multi-phase compressible flow based on the work of Zheng et al. [2011] and Shyue [2001]. *blastFoam* provides implementations of the essential numerical methods (e.g. 2nd and 3rd order schemes), equations of state (e.g. ideal gas, stiffened gas, Jones-Wilkins-Lee, etc.), run-time selectable flux schemes (e.g. HLL, HLLC, AUSM+, Kurganov/-Tadmor) and high-order explicit time integration (e.g. 2nd, 3rd, and 4th order). *blastFoam* provides activation and explosive burn models to simulate the initiation and expansion of energetic materials, as well as afterburn models to simulate under-oxygenated explosives which exhibit delayed energy release.

[1] *DISCLAIMER: This offering is not approved or endorsed by OpenCFD Limited, producer and distributor of the OpenFOAM software via www.openfoam.com, and owner of the OPENFOAM and OpenCFD trade marks.*

Contents

1 Introduction

blastFoam is an opensource library based on the OpenFOAM C++ library [OpenCFD Ltd., 2018b] to simulate strongly compressible systems of equations with detonations. Along with the main solvers for simulating fluid flow, additional utilities have been added to simplify case setup, allow adaptive mesh refinement, and case post-processing.

The purpose of this guide is to serve as a reference on the governing equations of *blastFoam* and the models that have been implemented. The guide also provides a brief description of the capabilities of the solver and the new utilities that have been introduced on top of those available in the standard OpenFOAM-7.

1.1 System Requirements

blastFoam currently builds against OpenFOAM-7[2]. The following dependencies are required to successfully compile and run *blastFoam*:

1. A working installation of OpenFOAM-7 complete with libraries and headers produced by OpenFOAM-7, no other packages are required to run *blastFoam* applications.

2. In order to create and view the postscript plots (e.g. *.eps files) created in the validation cases, *gnuplot* and an eps viewer must be installed. *gnuplot* is available on all platforms. EPS Viewer can be used on Windows and *ghostview* can be used on linux. On macOS, *Preview* opens eps files natively.

1.2 Downloading

The un-compiled source code can be obtained at: `https://github.com/synthetik-technologies/blastfoam`. Download or clone the source in the following location: `$HOME/OpenFOAM/blastfoam`

```
mkdir -p $HOME/OpenFOAM # create the OpenFOAM directory
cd $HOME/OpenFOAM # go to the £HOME/OpenFOAM directory
git clone https://github.com/synthetik-technologies/blastfoam
```

[2] `https://github.com/OpenFOAM/OpenFOAM-7`

1.3 Compilation

After cloning the the *blastFoam* source code from https://github.com/synthetik-technologies/blastfoam (see previous section), similar to OpenFOAM, add the following line to `$HOME/.bashrc` file:

```
source ~/OpenFOAM/blastfoam/etc/bashrc
```

and run:

```
source ~/.bashrc
```

in any open terminals to update the BASH environment. This is used to specify the local directories included during *blastFoam* compilation, as well as make future developments easier and less intrusive. The default path assumes that the *blastFoam* directory is located in `$HOME/OpenFOAM/blastfoam`. If this is not the case, the location will need to be changed in the `etc/bashrc` file by setting `BLAST_DIR` to the correct location.

Next run the `./Allwmake` command from the top `blastfoam` directory to compile the libraries and applications. Finally, ensure that OpenFOAM-7 and *blastFoam* have been installed and that the environment has been correctly setup by running one of the tutorial or validation cases.

Summary of steps to download, configure and compile *blastFoam*:

```
# 1. create the OpenFOAM directory
mkdir -p $HOME/OpenFOAM

# 2. go to the £HOME/OpenFOAM directory
cd $HOME/OpenFOAM

# 3. # clone the blastFoam repository
git clone https://github.com/synthetik-technologies/blastfoam

# go to the blastfoam directory
cd $HOME/OpenFOAM/blastfoam

# Append the etc/bashrc to your .bashrc file
echo "source $HOME/OpenFOAM/blastfoam/etc/bashrc" >> $HOME/.bashrc
```

```bash
# Load and set the bash environment to compile blastFoam
source $HOME/.bashrc

# Compile blastFoam
./Allwmake
```

1.4 Executable

The applications to solve these equations are executed by running the *blastFoam*, *blastReactingFoam*, or *blastXiFoam* commands. The executables are stored within the `$FOAM_USER_BIN` directory, and can be run from any directory.

1.5 Getting Help

This guide will attempt to cover the major points of the solver, but for more in depth question on the equations or models, please see the references listed in the class header files (*.H). Please report bugs using the issues tab on the GitHub page: https://github.com/synthetik-technologies/blastfoam/issues

2 Governing equations

The simulation of highly compressible materials, governed by a unique equation of state (EOS), is modeled using the Navier-Stokes equations. **U** is the vector of conservative variables, volume fraction (for multiphase simulations), mass, momentum, and energy, **F** are the fluxes corresponding to the respective conservative variables, and **S** is a vector of source terms.

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F} = \mathbf{S} \tag{1}$$

For a single phase with multiple species, the conservative variables and fluxes are defined as

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho Y_i \\ \rho \mathbf{u} \\ \rho E \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \rho \mathbf{u} \\ \rho Y_i \mathbf{u} \\ \rho \mathbf{u} \otimes \mathbf{u} + p\mathbf{I} \\ (\rho E + p)\mathbf{u} \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 0 \\ 0 \\ \dot{\mathrm{M}} + \dot{\mathrm{M}}_v \\ \dot{\mathrm{E}} + \dot{\mathrm{E}}_v \end{pmatrix}, \tag{2}$$

where Y_i is the mass fraction of specie i. And for multiple phases

$$\mathbf{U} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \alpha_1 \rho_1 \\ \vdots \\ \alpha_n \rho_n \\ \rho \mathbf{u} \\ \rho E \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \alpha_1 \mathbf{u} \\ \vdots \\ \alpha_n \mathbf{u} \\ \alpha_1 \rho_1 \mathbf{u} \\ \vdots \\ \alpha_n \rho_n \mathbf{u} \\ \rho \mathbf{u} \otimes \mathbf{u} + p\mathbf{I} \\ (\rho E + p)\mathbf{u} \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} \alpha_1 \nabla \cdot \mathbf{u} \\ \vdots \\ \alpha_n \nabla \cdot \mathbf{u} \\ 0 \\ \vdots \\ 0 \\ \dot{\mathrm{M}} + \dot{\mathrm{M}}_v \\ \dot{\mathrm{E}} + \dot{\mathrm{E}}_v \end{pmatrix}, \tag{3}$$

where ρ is the mixture density, $\mathbf{u}$ the mixture velocity, E the total energy, p the pressure, and ρ_i and α_i are the density and volume fraction of each phase. The source terms, $\dot{\mathrm{M}}$ and $\dot{\mathrm{E}}$ are the momentum and energy sources (including gravitational acceleration), and $\dot{\mathrm{M}}_v$ and $\dot{\mathrm{E}}_v$ are viscous terms.

The sum of all volume fractions is defined to be one, i.e.

$$\sum_i \alpha_i = 1 \tag{4}$$

When only two phases are used, the second volume fraction is defined

$$\alpha_2 = 1 - \alpha_1. \tag{5}$$

Additionally, the mixture density is defined by

$$\rho = \sum_i \alpha_i \rho_i. \tag{6}$$

When all phases are considered inviscid, $\dot{\mathrm{M}}_v = \dot{\mathrm{E}}_v = 0$. However, when viscosity is included,

$$\dot{\mathrm{M}}_v = \nabla \cdot \left(\rho\nu \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T + \frac{2}{3}(\nabla \cdot \mathbf{u})\mathbf{I} \right] \right), \tag{7}$$

and

$$\dot{\mathrm{E}}_v = \nabla \cdot \left(\rho\nu \left[\nabla \mathbf{u} + (\nabla \mathbf{u})^T + \frac{2}{3}(\nabla \cdot \mathbf{u})\mathbf{I} \right] \cdot \mathbf{u} \right) + \nabla \cdot (\alpha \nabla e), \tag{8}$$

where in ν is the viscosity, α is the thermal diffusivity, and $\mathbf{I}$ is the identity matrix.

The five equation model is used for multiphase systems in which a single shared velocity, shared internal energy, and temperature is used, and transports individual phase masses and volume fractions. Pressure is defined using a specified EOS, where the mixture's internal energy, densities, and volume fraction are used to calculate the total pressure.

3 Turbulence

In addition to the hyperbolic flux terms, viscous effects can be included in simulations. This includes the use of turbulence models such as Reynolds averaged Navier-Stokes (RANS) or Large Eddy simulations (LES) models. All single phase turbulence models standard to OpenFOAM including k-epsilon, k-omega, Smagorinsky, as well as many others are available. The specified model introduces additional transport equations, as well as modifications to transport coefficients like viscosity (ν) and thermal diffusivity (α) which modify the viscous terms in the conservation equations. When the maximum viscosity is zero, the `constant/turbulenceProperties` does not need to be included, however if any phase has viscous terms, this dictionary needs to be included along with the appropriate turbulence model (laminar is the same as no turbulence model). The available turbulence models are listed below, and these can also be found using the "-listTurbulenceModels" option when running `blastFoam`.

Turbulence models: `LES, RAS, laminar` (e.g. no model)
laminar models:

- `Giesekus`
- `Maxwell`
- `Stokes (default)`
- `generalizedNewtonian`

RAS models:

- `LRR`
- `LaunderSharmaKE`
- `RNGkEpsilon`
- `SSG`
- `SpalartAllmaras`
- `buoyantKEpsilon`
- `kEpsilon`
- `kOmega`
- `kOmegaSST`
- `kOmegaSSTLM`
- `kOmegaSSTSAS`
- `realizableKE`
- `v2f`

LES models:

- `DeardorffDiffStress`
- `Smagorinsky`
- `SpalartAllmarasDDES`
- `SpalartAllmarasDES`
- `SpalartAllmarasIDDES`
- `WALE`
- `dynamicKEqn`
- `dynamicLagrangian`
- `kEqn`
- `kOmegaSSTDES`

NOTE: Many of the turbulence models require additional field (`k`, ε, ω, etc.) to be specified in the initial conditions folder.

4 Fluid thermo models

To calculate the pressure and temperature given the conservative quantities (mass, momentum, and energy) three components are required: an equation of state, a thermodynamic model, and a transport model. These models are defined in the `phaseProperties` file located in the `constant` directory of each blastFoam case. Each phase within the `phaseProperties` file has several dictionary entries, and these models are specified in the `thermoType` sub-dictionary; the molecular weight must also be defined in the `specie` sub-dictionary.

```
phase_name
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      tabulated;
        equationOfState tabulated;
    }
    specie
    {
        molWeight   28.97;
    }
    transport
    {
        ...
    }
    thermodynamics
    {
       ...
    }
    equationOfState
    {
        ...
    }
}
```

4.1 Thermodynamic models

The thermodynamic model is used to calculate the internal energy based solely on the thermal contribution:

$$e_t = \int Cv(T)\mathrm{dT}. \tag{9}$$

An additional contribution from the equation of state is given by:

$$e_{eos}(\rho_i, T) = \int \left(\frac{\partial p(\rho_i, T)}{\partial T} \right), \mathrm{d}v_i \tag{10}$$

where $v_i = 1/\rho_i$ is the specific volume. The total internal energy is therefore

$$e = e_T(T) + e_{eos}(\rho_i, T) \tag{11}$$

Because the internal energy is tracked as a primitive quantity, the temperature is solved using a Newton-Raphson root finding method given the local internal enery and density.

Either internal energy or enthalpy-based coefficients can be used, but only internal energy can be used to define the total energy. The conversion from an enthalpy base to an internal energy base is CONDUCTED using the specified equation of state

$$C_v = C_p - CpMCv, \tag{12}$$

and

$$e = h - \frac{p_i}{\rho_i}, \tag{13}$$

where $CpMCv$ is determined from the equation of state using

$$CpMCv = \frac{\partial(p_i v_i)}{\partial T}. \tag{14}$$

For an ideal gas, $CpMCv = \bar{R}/W_i$ where $\bar{R}$ is the ideal gas constant (8314 [J/mol/K]) and W_i is the molecular weight in [kg/kmol]. Equations of state that have a pressure with a non-linear dependence on temperature will deviate from this.

4.1.1 eConst

A constant specific heat at constant volume is specified using the eConst thermodynamic model.

Variable	Description
C_v	Specific at constant volume [J/kg/K]

4.1.2 hConst

A constant specific heat at constant pressure is specified using the hConst thermodynamic model.

Variable	Description
C_p	Specific at constant pressure [J/kg/K]

4.1.3 JANAF

The JANAF thermodynamic model can be used in which the enthalpy is represented by a temperature based polynomial [OpenCFD Ltd., 2018a]. Seven coefficients must be specified for both the low temperature and high temperature states, as well as the high, low, and switching temperatures. Currently, this model is only available for ideal gases.

Variable	Description
highCpCoeffs	High enthalpy coefficients
lowCpCoeffs	Low enthalpy coefficients
Tlow	Minimum temperature [K]
Tcommon	Switching temperature [K]
Thigh	Maximum temperature [K]

4.2 Equation of states

In order to more easily handle a variety of equations of state, both temperature-based equations of state and internal energy based equations of state (e.g. in Mie-Grüneisen form) can be used.

4.2.1 Internal energy based (e.g. in Mie-Grüneisen form)

The Mie-Grüneisen form uses two contributions to calculate the pressure, and ideal gas term contribution, and a deviation term.

$$p = (\Gamma_i - 1)\rho_i e - \Pi_i, \tag{15}$$

Γ_i, also called the Grüneisen coefficient, and Π_i is the pressure deviation from an ideal gas and is specific to the equation of state. Using these quantities, the speed of sound is written as

$$c_i^2 = \frac{\Gamma_i p + \Pi_i}{\rho_i} - \delta_i(\Gamma_i - 1), \tag{16}$$

where δ_i is defined

$$\delta_i = -\frac{[p + \Pi_i]\Gamma_i'}{[\Gamma_i - 1]^2} + \frac{\Pi_i'}{\Gamma_i - 1}, \tag{17}$$

and where Γ_i' and Π_i' are the derivatives with respect to the phase density.

4.2.1.1 Ideal gas

For an ideal gas, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i. \tag{18}$$

The functions as defined to represent an ideal gas in Mie-Grüneisen form are:

$$\Gamma_i = \gamma_i, \tag{19}$$

$$\Pi_i = 0, \tag{20}$$

$$\delta_i = 0. \tag{21}$$

Variable	Description
γ	Specific heat ratio [ ]

4.2.1.2 Stiffened gas

For a material obeying the stiffened EOS, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i a_i, \tag{22}$$

where a_i and γ_i are material properties [Zheng et al., 2011]. In the Mie-Grüneisen form

$$\Gamma_i = \gamma_i, \tag{23}$$

$$\Pi_i = \gamma_i a_i, \tag{24}$$

and

$$\delta_i = 0. \tag{25}$$

Variable	Description
a	Reference pressure [Pa]
γ	Specific heat ratio [ ]

4.2.1.3 Tait

For a material obeying the Tait EOS, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i (b_i - a_i), \tag{26}$$

where a_i, b_i, and γ_i are material properties [Zheng et al., 2011]. In the Mie-Grüneisen form

$$\Gamma_i = \gamma_i, \tag{27}$$

and

$$\Pi_i = \gamma_i (b_i - a_i), \tag{28}$$

and

$$\delta_i = 0. \tag{29}$$

Variable	Description
a	Bulk modulus [Pa]
b	Reference pressure [Pa]
γ	Specific heat ratio [ ]

4.2.1.4 Van der Waals

For a gas described by the generalized van der Waals equation of state [Zheng et al., 2011], the pressure is defined as

$$p_i = \frac{\gamma_i - 1}{1 - b_i\rho_i}(\rho_i e_i + a_i\rho_i^2) - (a_i\rho_i^2 + c_i). \tag{30}$$

Putting the pressure in the Mie-Grüneisen form, we obtain

$$\Gamma_i = \frac{\gamma_i - 1}{1 - b_i\rho_i} + 1, \tag{31}$$

$$\Pi_i = \left[1 - \frac{\gamma_i - 1}{1 - b_i\rho_i}\right] a_i\rho_i^2 + \left[\frac{\gamma_i - 1}{1 - b_i\rho_i} + 1\right] c_i, \tag{32}$$

and

$$\delta_i = -b_i\frac{p_i + a_i\rho_i^2}{\gamma_i - 1} + \left(\frac{1 - b_i\rho_i}{\gamma - 1} - 1\right) 2a_i\rho_i. \tag{33}$$

Variable	Description
a	Attraction between particles [kg^{-1} m^5 s^{-2}]
b	Specific volume excluded due to particle volume [kg^{-1} m^3]
c	Reference pressure [Pa]
γ	Specific heat ratio [ ]

4.2.1.5 Landau, Stanyukovich, Zeldovich, and Kompaneets (LSZK)

The LSZK equation of state is a simple model for describing an ideal explosive [Lutzky, 1964, Needham, 2018]. It takes a form similar to a stiffened gas or Tait EOS, with

$$\Gamma_i = \gamma_i, \tag{34}$$

$$\Pi_i = a_i \rho_i^{b_i}, \tag{35}$$

and

$$\delta_i = 0. \tag{36}$$

Variable	Description
a	Model constant [kg^{1-b}s^{-2}m^{3b-1}]
b	Pressure exponent []
γ	Specific heat ratio [ ]

4.2.1.6 Jones Wilkins Lee (JWL)

The more complicated JWL EOS is often used to define energetic materials [Zheng et al., 2011], and has a pressure deviation given by

$$\Pi_i = A_i e^{-R_{1,i}V}\left(1.0 - \frac{\omega_i}{R_{1,i}V}\right) + B_i e^{-R_{2,i}V}\left(1.0 - \frac{\omega_i}{R_{2,i}V}\right) \tag{37}$$

where $V = \rho_{0,i}/\rho_i$.

$$\Gamma_{r,i} = \omega_i + 1, \tag{38}$$

and

$$\delta_{r,i} = \tag{39}$$

$$A_i e^{-\frac{R_{1,i}\rho_{0,i}}{\rho_i}}\left[\omega_i\left(\frac{1}{R_{1,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{1,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\omega_i} \tag{40}$$

$$+B_i e^{-\frac{R_{2,i}\rho_{0,i}}{\rho_i}}\left[\omega_i\left(\frac{1}{R_{2,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{2,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\omega_i}. \tag{41}$$

NOTE: The JWL model can also be used to describe the unreacted state if a detonating model is used with the "solidJWL" type.

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
R_1	Model coefficient [ ]
R_2	Model coefficient [ ]
ρ_0	Unreacted or reference density [kg m^{-3}]
ω	Grüneisen coefficient [ ]

4.2.1.7 Cochran-Chan

The Cochran Chan EOS can be used to describe a solid material [Zheng et al., 2011], and has a reference pressure given by

$$p_{ref,i} = A_i V_i^{1-\mathcal{E}_{1,i}} - B_i V_i^{1-\mathcal{E}_{2,i}}. \tag{42}$$

The functions used to defined the Mie-Grüneisen equations are

$$\Gamma_i = \Gamma_{0,i} + 1, \tag{43}$$

$$\Pi_i = \Gamma_{0,i}\rho_i \left(-\frac{A_i}{(\mathcal{E}_{1,i}-1)\rho_{0,i}} V_i^{1-\mathcal{E}_{1,i}} + \frac{B_i}{(\mathcal{E}_{2,i}-1)\rho_{0,i}} V_i^{1-\mathcal{E}_{2,i}} \right) - p_{ref,i}, \tag{44}$$

and

$$\delta_i = \tag{45}$$

$$\frac{A_i}{\mathcal{E}_{1,i}} \left[\mathcal{E}_{1,i} V_i^{-\mathcal{E}_{1,i}} \frac{\mathcal{E}_{1,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}} \right] \frac{1}{\Gamma_{0,i}} \tag{46}$$

$$+\frac{B_i}{\mathcal{E}_{2,i}} \left[\mathcal{E}_{2,i} V_i^{-\mathcal{E}_{2,i}} \frac{\mathcal{E}_{2,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}} \right] \frac{1}{\Gamma_{0,i}}. \tag{47}$$

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
$\mathcal{E}_1$	Model coefficient [ ]
$\mathcal{E}_2$	Model coefficient [ ]
ρ_0	Reference density [kg/m^3]
Γ_0	Grüneisen coefficient [ ]

4.2.1.8 Doan-Nickel

The Doan-Nickel EOS includes changes in the composition of air at high temperatures and the changing specific heat ratio due to dissociation of molecules Doan and Nickel [1963]. Due to the way this model is defined, it can currently only be used with inviscid fluids and a constant specific heat at constant volume (eConst). The specific heat is a function of the density and internal energy and is defined as

$$\Gamma_i = 1+\left[0.161 + 0.255e^{\tilde{e}/e_1} f_1 + 0.280e^{\tilde{e}/e_2}(1-f_1) + 0.137e^{\tilde{e}/e_3} f_2 + 0.050 f_3\right] \left(\frac{\rho_i}{\rho_{0,i}}\right)^{\alpha(\rho_i,\tilde{e})} \tag{48}$$

where $\tilde{e} = e \cdot 10^{-6}$. $e_1 = 4.46$, $e_2 = 6.63$, and $e_3 = 25.5$ are the internal energies at which oxygen dissociates, everything is dissociated, and electrons contribute, respectively. $\rho_{0,i}$ is the density of air at sea-level (1.293 [kg/m^3], and

$$\alpha(\rho_i, \tilde{e}) = [0.048 f_1 + 0.032(1-f_1)(1-f_2)] \log_{10}(\tilde{e}) + 0.045 f_3, \tag{49}$$

with

$$f_1 = \frac{1}{\exp(\frac{\tilde{e}-e_{O_2->2O}}{\Delta e_{O_2->2O}}) + 1}, \tag{50}$$

$$e_{O_2->2O} = 8.5 + 0.357\log_{10}\left(\frac{\rho_i}{\rho_{0,i}},\right) \tag{51}$$

$$\Delta e_{O_2->2O} = 0.975\left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.05}, \tag{52}$$

$$f_2 = \frac{1}{\exp(\frac{e_{N_2->2N}-\tilde{e}}{\Delta e_{N_2->2N}}) + 1}, \tag{53}$$

$$e_{N_2->2N} = 45.0\left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.0157}, \tag{54}$$

$$\Delta e_{N2->2N} = 4.0\left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.085}, \tag{55}$$

$$f_3 = \frac{1}{\exp(\frac{e_e-\tilde{e}}{\Delta e_e}) + 1}, \tag{56}$$

$$e_e = 160.0, \tag{57}$$

and

$$\Delta e_e = 6.0. \tag{58}$$

Due to the complexity of the model, and the fact that it was *specifically* developed as a model for air, all coefficients are hard-coded and cannot be changed. This equation of state should only be used to represent standard air.

4.2.2 Temperature-based

While it is preferable to use an internal energy based equation of state for compressible flows, some equations of state cannot be converted, and therefore use the local temperature to define the thermodynamic pressure. Generally the Grüneisen coefficient used to calculate the shared pressure is defined as

$$\Gamma_i = \frac{v_i}{C_v}\frac{\partial p}{\partial T}, \tag{59}$$

where C_v is the specific heat at constant volume, and the speed of sound is defined

$$c_i^2 = v_i^2 \left[\left(\frac{\partial p_i}{\partial T} \right)^2 * \frac{T}{C_v} - \frac{\partial p}{\partial v_i} \right]. \tag{60}$$

Unless otherwise specified, these relations are used to calculate the relevant quantities.

4.2.2.1 Jones Wilkins Lee C-form (JWLC)

The C-form of the JWL equation of state [Souers et al., 2000] does not have a dependence on the temperature or internal energy and has a pressure defined as

$$p_i = A_i e^{-R_{1,i}V} + B_i e^{-R_{2,i}V} + \frac{C_i}{V^{1+\omega_i}}, \tag{61}$$

the speed of sound if defined

$$c_i^2 = \frac{V^2}{\rho_{0,i}} \left(A_i e^{-R_{1,i}V} + B_i e^{-R_{2,i}V} + \frac{C_i(1+\omega_i)}{V^{2+\omega_i}} \right). \tag{62}$$

The Grüneisen coefficient is the same as the standard JWL equation of state.

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
C	Model coefficient [Pa]
R_1	Model coefficient [ ]
R_2	Model coefficient [ ]
ρ_0	Unreacted or reference density [kg m^{-3}]
ω	Grüneisen coefficient [ ]

4.2.2.2 Becker-Kistiakowsky-Wilson (BKW)

The BKW equation of state is another equation of state that is commonly used to simulate explosive materials [Mader, 1963]. Unlike the JWL equation of state, the BKW models uses the temperature, rather than the internal energy, to define the pressure.

$$p_i = RT\rho_i \left(1 + xe^{\beta x}\right) \tag{63}$$

where

$$x = \frac{\kappa k \rho_i}{W_i (T+\theta)^a} \tag{64}$$

and W_i is the molecular weight in [kg/kmol].

Variable	Description
β	Model coefficient [ ]
a	Temperature exponent [ ]
Θ	Model coefficient [K]
κ	Model coefficient [K^a]
k	Specie co-volume [m^3/kmol]

4.2.2.3 Benedict-Webb-Rubin (BWR)

The BWR equation of state was developed by [Benedict et al., 1942] to describe real gases using a set of eight coefficients that must be experimentally determined. The pressure is defined as

$$\begin{aligned} p_i = & \\ & RT\rho_i + (B_0RT - A_0 - \frac{C_0}{T^2})\rho_i^2 + (bRT - a)\rho_i^3 + a\alpha\rho_i^6 \\ & +c\frac{\rho_i^3}{T^2}(1 + \gamma\rho_i^2)e^{-\gamma\rho_i^2} \end{aligned} \tag{65}$$

Variable	Description
A_0	Model coefficient
B_0	Temperature exponent
C_0	Model coefficient
a	Model coefficient
b	Model coefficient
c	Model coefficient
α	Model coefficient
γ	Specific heat ratio [ ]

4.2.2.4 Murnaghan

The Murnaghan (or Munahan) equation of state is used for solid material and has no temperature dependence [Souers et al., 2000]. The pressure is a function of only the density and is defined as

$$p_i = p_{ref} + \frac{1}{\kappa n}\left(\frac{\rho_i}{\rho_{0,i}} - 1\right)^n. \tag{66}$$

Variable	Description
p_{ref}	Reference pressure [Pa]
ρ_0	Reference density [kg/m^3]
κ	Model coefficient [Pa^{-1}]
n	Model coefficient [ ]
Γ	Grüneisen coefficient [ ]

4.2.2.5 Birch-Murnaghan (2nd order)

The second order Birch-Murnaghan equation of state is the second order expansion of the Birch-Murnaghan model. The pressure is defined as

$$p_i = p_{ref} + \frac{3K_0}{2K_0'}\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{7/3} - \left(\frac{\rho_i}{\rho_{0,i}}\right)^{5/3}\right] \tag{67}$$

Many coefficients can be found for this in Zheng and Zhao [2016].

Variable	Description
p_{ref}	Reference pressure [Pa]
ρ_0	Reference density [kg/m^3]
K_0	Bulk modulus [Pa]
Γ	Grüneisen coefficient [ ]

4.2.2.6 Birch-Murnaghan (3rd order)

The Third order Birch-Murnaghan equation of state is the third order expansion of the Birch-Murnaghan see Zheng and Zhao [2016]. The pressure is defined as

$$p_i = p_{ref} + \frac{3K_0}{2K_0'}\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{7/3} - \left(\frac{\rho_i}{\rho_{0,i}}\right)^{5/3}\right]\left\{1 + 0.75(K_0' - 4)\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{2/3} - 1\right]\right\} \tag{68}$$

Many coefficients can be found for this in Zheng and Zhao [2016].

Variable	Description
p_{ref}	Reference pressure [Pa]
ρ_0	Reference density [kg/m^3]
K_0	Bulk modulus [Pa]
K_0'	Change in bulk modulus w.r.t. pressure [ ]
Γ	Grüneisen coefficient [ ]

4.3 Tabulated equation of state and thermodynamic model

In addition to the previously mentioned equations of state and thermodynamic models, tabulated models may also be used. This is useful when simulating real gasses, and high temperatures and pressures when standard models are no longer sufficient. Any combination of pressure and temperature tables can be used, as long as they are a function of density and internal energy (x and y, respectively). The internal energy is initialized using a reverse look up given a pressure and a density. Modifiers can also be used such as: exponential, natural log, and log base 10. The minimum values of density and internal energy, as well as the spacing, both in the modified values (i.e. $\log(\rho_{min})$ for a density in natural log units). The number of rows and columns must also be specified (nRho and ne). All necessary derivatives are calculated using the finite difference method. An example is shown below:

```
phase
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      tabulated;
        equationOfState tabulated;
    }
    specie
    {
        molWeight   28.97;
    }
    thermodynamics
    {
        file    "T.csv";
        mod     ln;     // log(p_i)=p_table

        nRho    7;      // Number of columns
        rhoMod  log10;  // log_10(rho_i)=minRho+i*dRho
        minRho  -3.0;   // 0.001 kg/m^3
        dRho    1.0;

        ne      40;     // Number of rows
        eMod    ln;     // log(e)=mine+j*de
        mine    11.8748;
        dRho    1.0;
    }
    equationOfState
    {
        file     "p.csv";
        mod      ln;      // log(p_i)=p_table

        nRho     7;       // Number of columns
        rhoMod   log10;   // log_10(rho_i)=minRho+i*dRho
        minRho   -3.0;    // 0.001 kg/m^3
        dRho     1.0;

        ne       40;      // Number of rows
        eMod     ln;      // log(e)=mine+j*de
        mine     11.8748;
        dRho     1.0;
    }
}
```

Variable	Description
file	Name of file
mod	Modifier of the data (log10, ln, exp)
nRho	Number of columns
ne	number of rows
rhoMod	Density modifier
eMod	internal energy modifier
dRho	Density spacing in modified units
de	internal energy spacing in modified units

4.4 Transport

The transport models are used to define quantities such as viscosity and thermal diffusion. Only Newtonian viscosity models are currently available.

4.4.1 Constant transport (const)

Constant values for viscosity (μ) and Prandtl number are used. These values are used to calculate the thermal diffusion coefficient using $\alpha_T = \mu/Pr$.

Variable	Description
μ	Dynamic viscosity [Kg/m/s]
Pr	Prandtl number [ ]

4.5 Basic thermodynamic model

The `basicThermoFluid` model (type `basic`) uses a transport model, thermodynamic model, and equation of state to describe a phase. The basic model should be used for non-reacting phases.

4.6 Detonating thermodynamic model

In order to better describe a detonating material, a detonating thermodynamic model can be used. This model transitions between an unreacted state and a reacted state using an activation model. Each state (unreacted and reacted) is described by its own basic thermodynamic model, allowing for different models and coefficients to be used for values such as viscosity or specific heats. In addition to the activation model, afterburn models can also be used to add additional energy into the system after the initial detonation reaction has taken place.

Some standard combinations of unreacted and reacted equations of states include the `JWL++` (`Murnaghn` and `JWLC`), and the `blendedJWL` (`solidJWL` and `JWL`).

NOTE: Whilst it will not give the correct temperature, using an ideal gas is a simple option if the solid properties of the unreacted material are not known.

```
phase1
{
    type            detonating;
    reactants
    {
        ...
    }
    products
    {
        ...
    }
    activationModel linear;
    initiation
    {
        E0      9.0e9;
        points  ((0 0 0) (0.1 0.1 0.1));
        vDet    5000;
    }

    afterburnModel Miller;
    MillerCoeffs
    {
        Q0      1e7;
        m       0.1667;
        n       0.5;
        a       0.0195e6;
        pMin    1.1e5;
    }
}
```

4.6.1 Activation Models

In order to accurately model the release of energy, different models can be used to approximate the speed at which the solid reactants become the product gases. The currently available models include the linear model, a pressure based models, and finally an Arrhenius rate reaction. Theses models all use a reaction progress variable, λ_i, to describe the state of individual cell activation. Additionally, in order to add numerical stability, the initial energy is defined using the unreacted equation of state, and the reaction energy, E_0, is added based on the time rate of change of the reaction progress variable, i.e.

$$\dot{\mathrm{E}} = \frac{\mathrm{E}_0 \rho_i}{\rho_{0,i}} \frac{\mathrm{d}\lambda}{\mathrm{dt}} \tag{69}$$

The phase properties are calculated using

$$x_i = \lambda_i^l x_{r,i} + (1 - \lambda_i^l) x_{u,i)} \tag{70}$$

where $x_{r,i}$ is the reacted phase quantity, $x_{u,i}$ is the unreacted phase quantity, and l is the exponent used to blend the unreacted and reacted states (lamdaPow). By default l is one. The advection scheme and limiters for λ_i need to be specified in the `fvSchemes` in the same manner as the other transported quantities.

Variable	Description
E_0	Detonation energy [Pa]
lambdaPow	Exponent used for blending [ ] (default is 1)

4.6.1.1 No Activation (none)

The `None` activation model assumes that all cells are activated when the simulation begins. This is the equivalent to an infinitely fast detonation velocity.

4.6.1.2 Linear Activation (linear)

The most simple activation models uses a list of detonation points and a constant detonation velocity to determine which cells have been activated. This gives

$$\lambda_{i,j} = \begin{cases} 1 & |\mathbf{x} - \mathbf{x}_{\mathrm{det,i,j}}| < \mathrm{v}_{\mathrm{det,i}} t \\ 0 & \text{else} \end{cases} \tag{71}$$

where $\mathbf{x}$ is the cell center, $\mathbf{x}_{\mathrm{det,i,j}}$ is the $\mathrm{j^{th}}$ detonation point, $\mathrm{v}_{\mathrm{det,i}}$ is the detonation velocity, and t is the time. In order to account for multi-point detonation the actual λ is defined as

$$\lambda_i = \max(\lambda_{i,j}) \tag{72}$$

Variable	Description
points	List of activation points
vDet	Speed of propagation within the material [m/s]

4.6.1.3 Pressure based activation (pressureBased)

A more advanced activation model uses the local cell pressure, determined by the equation of state, and follows the evolution equation

$$\frac{\mathrm{d}\lambda}{\mathrm{dt}} = \mathrm{I}\left(\frac{\rho_i}{\rho_{0,i}} - 1 - a\right)^x (1-\lambda)^b + \mathrm{G}_1(1-\lambda)^c\lambda^d p^y + \mathrm{G}_2(1-\lambda)^e\lambda^f p^z \tag{73}$$

The first term is responsible for the initial detonation based on the compression of the material. The following two terms account for the subsequent reaction based on the local pressure. All terms can be turned "on" or "off" by specifying the minimum and maximum values of lambda at which the stages occur.

Variable	Description
I	Ignition rate [s^{-1}]
a	Compression ratio offset [ ]
b	Ignition $(1-\lambda)$ exponent [ ]
x	Compression ratio exponent [ ]
maxLambdaI	Maximum value of λ that the ignition term is used [ ]
G_1	First activation rate [$\mathrm{Pa}^{-y}\mathrm{s}^{-1}$]
c	First $(1-\lambda)$ exponent [ ]
d	First λ exponent [ ]
y	First pressure exponent [ ]
minLambda1	Minimum value of λ that the first step is used [ ]
maxLambda1	Maximum value of λ that the first step is used [ ]
G_2	Second activation rate [$\mathrm{Pa}^{-z}\mathrm{s}^{-1}$]
e	Second $(1-\lambda)$ exponent [ ]
f	Second λ exponent [ ]
z	Second pressure exponent [ ]
minLambda2	Minimum value of λ that the second step is used [ ]
maxLambda2	Maximum value of λ that the second step is used [ ]

4.6.1.4 Arrhenius rate activation (ArrheniusRate)

A more physical representation of the activation rate uses the local temperature, defined by the internal energy and the specific heat of the mixture. The evolution equation for the reaction progress variable is

$$\frac{\mathrm{d}\lambda}{\mathrm{dt}} = (1-\lambda)\begin{cases} \mathrm{d}_\mathrm{p}^2\mathrm{A}_\mathrm{low}\mathrm{exp}(\frac{T_\mathrm{a,low}}{T}) & T < T_s \\ \mathrm{A}_\mathrm{high}\mathrm{exp}(\frac{T_\mathrm{a,high}}{T}) & \mathrm{else} \end{cases}. \tag{74}$$

Variable	Description
d_p	Particle diameter [m]
A_{low}	Low activation rate [m^{-2} s^{-1}]
$T_{a,low}$	Low activation temperature [K]
A_{high}	High activation rate [s^{-1}]
$T_{a,high}$	High activation temperature [K]
T_s	Switch temperature [K]

4.6.2 Afterburn Models

When reactions occur over a period of time after the initial charge has been activated, additional energy can be added that results in higher pressure and temperature. This is referred to as *afterburn.* Afterburn is typical in under-oxygenated and/or non-ideal explosives, which may release energy at a later time. The advection scheme and limiters for c_i need to be specified in the `fvSchemes` in the same manner as the other transported quantities.

4.6.2.1 No Afterburn (none [default])

No additional energy is added. If the keyword *afterburnModel* is not found in the phase dictionary, this is the model that is used.

4.6.2.2 Constant Afterburn (constant)

A constant amount of afterburn energy is added where $\dot{Q} = Q_0$.

Variable	Description
Q_0	Afterburn energy [J/kg/s]

4.6.2.3 Linear Afterburn (linear)

Afterburn energy is added linearly from a start time to an end time where

$$\dot{Q} = \begin{cases} \frac{Q_0}{\Delta t} & t > t_{start} \& t < t_{end} \\ 0 & \text{else} \end{cases}. \tag{75}$$

Variable	Description
Q_0	Afterburn energy [J/kg]
t_{start}	Time energy begins being added [s]
t_{end}	Time energy stops being added [s]

4.6.2.4 Miller Extension Afterburn Model (Miller)

The model of Miller [1995/ed] uses an evolution equation to determine the fraction of the total amount of afterburn energy added to the system due to unburnt material. c_i is the reaction progress variable that has a value between 0 and 1, with an evolution equation defined as

$$\frac{\partial \rho c_i}{\partial t} = \rho a (1 - c_i)^m p^n, \tag{76}$$

and the total afterburn energy is found using

$$\dot{Q} = \frac{\mathrm{d}c_i}{\mathrm{d}t} Q_0. \tag{77}$$

Variable	Description
a	Model constant [Pa^{-n}]
m	Reaction progress variable exponent [ ]
n	Pressure exponent [ ]
Q_0	Afterburn energy [J/kg]
p_{min}	Minimum pressure that afterburn occurs [Pa]

4.7 Two/multiphase models

To simulate a two-phase or multiphase system, the keyword *phases* must be defined in the `phaseProperties` dictionary and defines the list of phases used in a simulation. The selection of a two-phase or multiphase system is determined by the number of defined phases. The mixture pressure is defined by

$$p = \frac{\sum_i \alpha_i \xi_i p_i}{\sum_i \alpha_i \xi_i}, \tag{78}$$

where

$$\xi_i(\rho_i) = \frac{1}{\Gamma_i - 1}, \tag{79}$$

and p_i is the pressure from a single equation of state [Zheng et al., 2011]. The speed of sound within a given phase is given by

$$c_i = \sqrt{\frac{\sum_i y_i \xi_i c_i^2}{\sum_i \xi_i}} \tag{80}$$

with

$$y_i = \frac{\alpha_i \rho_i}{\rho}. \tag{81}$$

4.8 Initialization

The initialization of fields can be handled in several ways, but ρ_i, p, and T must be given as initial conditions. Some of these fields may be overwritten, but the boundary conditions are used to construct the conserved quantities. For equations of state in the Mie-Grüneisen form, by default the pressure and density are used to initialize the internal energy. Temperature based equations of state initialize the internal energy using the density and temperature. If the *calculateDensity* keyword is defined, and set to *yes* in the given equation of state dictionary the density is calculated from the pressure and temperature prior to the initialization of the internal energy. Each of the initialization methods use a Newton-Raphson root finding method to find the necessary quantities. If the internal energy field is provided in the initial conditions folder, the pressure and temperature fields are calculated directly from the density and internal energy.

4.9 Example

An example of a single phase phaseProperties can be seen below

```
mixture
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
    }
    transport
    {
        mu      0.0;
        Pr      1.0;
    }
    thermodynamic
    {
        Cv      718;
        Hf      0.0;
    }
    idealGas
    {
        gamma   1.4;
    }
}
```

For a two or multiphase simulation, the dictionary is defined as

```
phases (air water);
air
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
    }
    transport
    {
        mu      0.0;
        Pr      1.0;
    }
    thermodynamic
    {
        Cv      718;
        Hf      0.0;
    }
    equationOfState
    {
        gamma   1.4;
    }
}
```

```
water
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      hConst;
        equationOfState stiffenedGas;
    }
    specie
    {
        molWeight   18.0;
    }
    transport
    {
        mu      0.0;
        Pr      1.0;
    }
    thermodynamic
    {
        Cp      4186;
        Hf      0.0;
    }
    equationOfState
    {
        gamma   4.4;
        a       7e8;
    }
}
```

5 Flux Evaluation

blastFoam relies on explicit solutions for the evolution of conservative variables. This has several benefits in terms of order of accuracy and computational cost. First, higher-order, explicit Runge-Kutta time integration schemes can be used which allow for fully third order solutions to be obtained. Secondly, higher order interpolation schemes, such as cubic, are marginally more computationally expensive than the standard linear interpolation. This is very different than the higher-order divergence, gradient, and Laplacian schemes which are used in implicit solution methods and are significantly more expensive than their linear counterparts. This allows for use of higher order schemes without the downside of significant additional computational cost.

All of the methods currently available to calculate the conservative, hyperbolic fluxes rely on the owner (own) and neighbour (nei) interpolated values on a face (also called left and right states). These interpolated values are found using a combination of a base interpolation scheme (linear or cubic) and flux limiters. OpenFOAM currently has a wide selection of available limiters such as upwind, Minmod, vanLeer, SuperBee, etc. for scalars, and upwind, MinmodV, vanLeerV, SuperBeeV, etc. for vectors; all are run-time selectable.
An example of the `fvSchemes` dictionary is shown below

```
fluxScheme HLLC;

ddtSchemes
{
    default Euler;
    timeIntegrator RK2SSP;
}

gradSchemes
{
    default cellMDLimited leastSquares 1.0;
}

divSchemes
{
    default none;
}

laplacianSchemes
{
    default Gauss linear corrected;
}

interpolationSchemes
{
    default cubic; //Base interpolation scheme

    // Limiters
    reconstruct(alpha)  vanLeer; // Volume fraction
    reconstruct(rho)    vanLeer; // Density
    reconstruct(p)      vanLeer; // Pressure
    reconstruct(U)      vanLeerV; // Velocity
    reconstruct(e)      vanLeer; // Internal energy
    reconstruct(c)      vanLeer; // Speed of sound
}
```

5.1 Riemann Solvers

The currently available Riemann solvers are described below. For all schemes, $\mathbf{n}$ denotes the surface normal vector, and $u = \mathbf{u} \cdot \mathbf{n}$.

5.1.1 HLL

The HLL scheme is based on Toro et al. [1994] in which the contact wave is neglected. The fluxes take the form:

$$\mathbf{F}^{HLL} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \frac{S_{nei}\mathbf{F}_{own} - S_{own}\mathbf{F}_{nei} + S_{own}S_{nei}(\mathbf{U}_{nei} - \mathbf{U}_{own})}{S_{nei} - S_{own}} & \text{if } S_{own} \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \tag{82}$$

The owner and neighbor states to approximate the wave propagation speeds are defined as

$$S_{own} = \min(u_{own} - c_{own}, \tilde{u} - \tilde{c}), \tag{83}$$

$$S_{nei} = \max(u_{nei} + c_{nei}, \tilde{u} + \tilde{c}), \tag{84}$$

where u_K is the normal flux of the respective state, c_K is the speed of sound, and $\tilde{u}$ and $\tilde{c}$ are the Roe averages velocities and speed of sounds, respectively.

5.1.2 HLLC

The approximate HLLC Riemann was developed by Toro et al. [1994], and improves upon the HLL method by providing an estimation of the contact wave between the owner and neighbour waves. This means that the state between the owner and neighbour waves now has two states rather than one. The following fluxes are thus used, using the fact that both pressure and velocity are constant across the contact wave,

$$\mathbf{F}^{HLLC} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \mathbf{F}_{*own} & \text{if } S_{own} \leq 0 \leq S_* \\ \mathbf{F}_{*nei} & \text{if } S_* \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \tag{85}$$

The contact wave speed is given by

$$S_* = \frac{\rho_{own} u_{own}(S_{own} - U_{own}) + \rho_{nei} u_{nei}(S_{nei} - U_{nei}) + p_{nei} - p_{own}}{\rho_{own}(S_{own} - u_{own}) - \rho_{nei}(S_{nei} - u_{nei})} \tag{86}$$

and the pressure in the

$$p_{*K} = \rho_K (S_K - U_K)(S_* - u_K) + p_K \tag{87}$$

The final fluxes are determined by

$$\mathbf{F}^{HLLC}_{*K} = \frac{S_*(S_K \mathbf{U}_K - \mathbf{F}_K) + S_K p_{*K} \mathbf{D}_*}{S_K - S_*} \tag{88}$$

$$\mathbf{D}_* = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{n} \\ S_* \end{pmatrix}. \tag{89}$$

5.1.3 HLLC-P (HLLCP)

Following the work of Xie et al. [2019], a low-speed correction of the standard HLLC flux scheme has been added. Currently it is only available for single phase simulations.

$$\mathbf{F}^{HLLC-P} = \mathbf{F}^{HLLC} + \mathbf{\Phi}^{P}, \tag{90}$$

$$\mathbf{\Phi}_p = (f-1)\frac{S_{own}S_{nei}}{S_{nei}-S_{own}}\frac{1}{1+|\tilde{M}|}\frac{\Delta p}{\tilde{c}^2}\begin{pmatrix} 1 \\ \tilde{\mathbf{u}} \\ \frac{1}{2}\tilde{u}^2 \end{pmatrix}. \tag{91}$$

$$f = \min\left(\frac{p_{own}}{p_{nei}}, \frac{p_{nei}}{p_{own}}\right)^3, \tag{92}$$

where the minimum of all cell faces is used for each cell.

$$p_{**} = \theta p_* + (1-\theta)\frac{p_{own}+p_{nei}}{2} \tag{93}$$

$$\theta = \min(M_*, 1) \tag{94}$$

$$p_{***} = f p_{**} + (1-f)p_*, \tag{95}$$

where p_{***} replaces p_{*K} in Eq. (88)

5.1.4 AUSM+

The AUSM+ scheme was developed by Luo et al. [2004] and constructs the fluxes based on the mass flux across the face.

$$\mathbf{F}^{AUSM+} = 0.5\left[M_* c_*(\mathbf{U}_{own}+\mathbf{U}_{nei})\right] - 0.5\left[|M_*| c_*(\mathbf{U}_{own}+\mathbf{U}_{nei})\right] + \mathbf{F}_p, \tag{96}$$

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ p_*\mathbf{n} \\ p_* \end{pmatrix}. \tag{97}$$

The star state variables are written as

$$c_* = 0.5(c_{own} + c_{nei}), \tag{98}$$

$$M_* = \mathcal{M}^+_{4,own} + \mathcal{M}^-_{4,nei}, \tag{99}$$

and

$$P_* = \mathcal{P}^+_{5,own} p_{own} + \mathcal{P}^-_{5,nei} p_{nei}, \tag{100}$$

where

$$\mathcal{M}^\pm_1(M) = 0.5(M \pm |M|), \tag{101}$$

$$\mathcal{M}^\pm_2(M) = \begin{cases} \mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \pm 0.25(M \pm 1)^2 & \text{else} \end{cases}, \tag{102}$$

$$\mathcal{M}^\pm_4(M) = \begin{cases} \mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \mathcal{M}^\pm_2(M)\left[1 \mp 16\beta\mathcal{M}^\mp_2(M)\right] & \text{else} \end{cases}, \tag{103}$$

$$\mathcal{P}^\pm_5(M) = \begin{cases} \frac{1}{M}\mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \pm\mathcal{M}^\pm_2(M)\left[(2 \mp M) - 16\alpha M\mathcal{M}^\mp_2(M)\right] & \text{else} \end{cases}, \tag{104}$$

with $\alpha = 3/16$ and $\beta = 1/8$.

5.1.5 Tadmor/Kurganov

The Tadmor/Kurganov fluxes were developed by Kurganov and Tadmor [2000]. The *rhoCentalFoam* solver [OpenCFD Ltd., 2018b, Greenshields et al., 2010] utilizes this scheme, which has been ported to the current framework. Two run-time selectable options (e.g. `Kurganov` or `Tadmor`) are available; both follow the same general procedure with a slight difference in the calculation of coefficients.

The Kurganov scheme defines

$$a^+ = \max(\max(u_{own} + c_{own}, u_{nei} + c_{nei}), 0), \tag{105}$$

$$a^- = \min(\min(u_{own} - c_{own}, u_{nei} - c_{nei}), 0), \tag{106}$$

$$a_{own} = \frac{a^+}{a^+ + a^-}, \tag{107}$$

$$a_{nei} = \frac{a^-}{a^+ + a^-}, \tag{108}$$

and

$$a = \frac{a^- a^+}{a^+ + a^-}. \tag{109}$$

The Tadmor scheme defines $a_{own} = a_{nei} = 0.5$ and $a = max(|a^-|, |a^+|)$.
The volumetric fluxes are written

$$\phi_{own} = u_{own} a_{own} - a, \tag{110}$$

and

$$\phi_{nei} = u_{nei} a_{nei} + a. \tag{111}$$

The resulting fluxes are

$$\mathbf{F}^{KT} = (\phi_{own}\mathbf{U}_{own} + \phi_{nei}\mathbf{U}_{nei}) + \mathbf{F}_p, \tag{112}$$

with

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ (a_{own}p_{own} + a_{nei}p_{nei})\mathbf{n} \\ a(p_{own} - p_{nei}) \end{pmatrix}. \tag{113}$$

5.2 Riemann advection scheme

In order to allow for consistency between the conserved quantities (mass, momentum, and energy), and additional transport quantities such as turbulence fields or species mass fractions, an additional advection scheme using the Riemann fluxes has been added. This makes use of the quantities calculated in transport of the conserved variable which then allows for the use of higher order interpolation schemes to be used for additional scalar transport. This is a major advantage over traditional methods which are generally limited to upwinding for additional transport due to presence of strong gradients.

```
divSchemes
{
    default                          none;
    div(alphaRhoPhi.tnt,lambda.tnt)  Riemann;
    div(alphaRhoPhi.tnt,c.tnt)       Riemann;
}

interpolationSchemes
{
    default cubic;
    ...

    reconstruct(lambda.tnt)  vanLeer;
    reconstruct(c.tnt)       vanLeer;
}
```

In the interpolation schemes sub-dictionary, the limiters should be specified for each field using the Riemann fluxes

NOTE: One limitation of using the Riemann fluxes is that, because an explicit advection term for the transported quantities is used, when very stiff implicit terms are present (such as production term in turbulence models), the advection term can become negligible leading to no advection occurring. For this reason, implicit divergence schemes are still recommended for the advection of turbulence quantities and other equations with large implicit source terms.

5.3 Time integration

Higher order time integration has been added allowing for fully third order accurate solutions to the evolution equations. This is a major benefit in comparison to standard OpenFOAM time integration which is at most second order due to the limitation on the implicit time evolution. This functionality applies to both conservative quantities such as mass, momentum, and energy, as well as the activation and afterburn models. The transport of turbulent quantities are still limited by the standard time integration schemes in OpenFOAM. Below are the currently available time integration schemes and the mathematical steps associated with them. The superscript n denotes the state at the beginning of the current time step, (i) denotes the i-th step, and $n+1$ denotes final state. The time integration scheme is specified in the `fvSchemes` dictionary using

```
ddtSchemes
{
    default Euler;
    timeIntegrator RK2SSP;
}
```

5.3.1 Euler

Standard first order time integration

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{114}$$

5.3.2 RK2

Standard second-order Runge-Kutta method (mid point):

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{115}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{116}$$

5.3.3 RK2-SSP (RK2SSP)

Second-order, strong-stability-preserving Runge-Kutta method [Spiteri and Ruuth, 2002]:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{117}$$

$$\mathbf{U}^{n+1} = \frac{1}{2}\left(\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right). \tag{118}$$

5.3.4 RK3-SSP (RK3SSP)

Third-order, strong-stability-preserving Runge-Kutta method [Spiteri and Ruuth, 2002]:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{119}$$

$$\mathbf{U}^{(2)} = \frac{1}{4}\left(3\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{120}$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\left(3\mathbf{U}^n + 2\mathbf{U}^{(2)} - 2\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{121}$$

5.3.5 RK4

Standard fourth-order Runge-Kutta method:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{122}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{123}$$

$$\mathbf{U}^{(3)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{124}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \frac{1}{6}\Delta t \boldsymbol{\nabla} \cdot \left(\mathbf{F}^n + 2\mathbf{F}^{(1)} + 2\mathbf{F}^{(2)} + \mathbf{F}^{(3)}\right) \tag{125}$$

5.3.6 RK4-SSP (RK4SSP)

Fourth-order, strong-stability-preserving Runge-Kutta method [Spiteri and Ruuth, 2002]:

$$\mathbf{U}^{(i)} = \sum_{k=0}^{i-1} \left(a_{ik}\mathbf{U}^{(k)} + \Delta t \beta_{ik} \boldsymbol{\nabla} \cdot \mathbf{F}^{(k)}\right) \tag{126}$$

Table 1: a_{ik}

0	1			
1	$\frac{649}{1600}$	$\frac{951}{1600}$		
2	$\frac{53989}{2500000}$	$\frac{4806213}{20000000}$	$\frac{23619}{32000}$	
3	$\frac{1}{5}$	$\frac{6127}{30000}$	$\frac{7873}{30000}$	$\frac{1}{3}$
	0	1	2	3

Table 2: β_{ik}

0	1			
1	$\frac{-10890423}{25193600}$	$\frac{5000}{7873}$		
2	$\frac{-102261}{5000000}$	$\frac{-5121}{20000}$	$\frac{7873}{10000}$	
3	$\frac{1}{10}$	$\frac{1}{6}$	0	$\frac{1}{6}$
	0	1	2	3

6 AdaptiveFvMesh

`adaptiveFvMesh`, a modified version of `dynamicRefineFvMesh` which uses octree refinement has been added. The modified version allows for both 2D and 3D adaptive refinement. The 3D refinement uses the standard OpenFOAM implementation, and the authors of the 2D refinement method have been included in `hexRef2D.H`. The selection of 2D or 3D refinement is done using the number of solution directions and the user is not required to specify the number of refinement dimensions.

Because the detonation of explosives may not occur when the simulation starts, the option to begin unrefinement at a designated time (*beginUnrefine*) has been added. This allows the mesh to remain refined in specific regions even though the error is small initially.

NOTE: Standard 2D meshes have no limitations, but when a wedge geometry is used, the cells along the axis currently cannot be refined.

NOTE: Whilst AMR may be used with meshes created using `snappyHexMesh`, regions can only be cut at level 0. This means that surface and region refinement cannot be used. If refinement is used with `snappyHexMesh` the simulation will crash when the mesh is refined in these areas.

6.1 Error estimators

In order to determine the cells that can be refined or unrefined, error estimators have been added. Generally, this is achieved by looking for a derivative or difference of a specified fields across all the faces a cell and it neighbours. The maximum value of all face errors of a given cell is taken as the cell center-based value, and is used to define the cells that should be refined.

Entry	Description
errorEstimator	Method used to calculate the error
beginUnrefine	Time that unrefinement can begin (0 by default)
refineInterval	Number of timesteps between refinement/unrefinement
field	Field used to deterermine refinement/unrefinement (error if an errorEstimator is used)
lowerRefinementLevel	Error that mesh is refined at
upperRefinement	Maximum error that mesh is refined (should be large if an error estimator is used)
unrefineLevel	Error that mesh is unrefined
nBufferLayers	Number of cells outside the refinement area that are refined
maxRefinement	Maximum number of refinement levels
maxCells	Maximum number of cells before refinement stops
correctFluxes	Not used with adaptiveFvMesh
dumpLevel	Is the refinement level written

6.1.1 delta

A basic method to detect sharp changes across faces uses a scaled difference approach, where the error is defined by

$$e_\delta = \frac{|x_{own} - x_{nei}|}{\min(x_{own}, x_{nei})} \tag{127}$$

where x is a field chosen by the user, and can be any scalar field stored within the simulation.

Entry	Description
field	Field used to compute error

6.1.2 Density gradient (densityGradient)

A modified density gradient was defined by Zheng et al. [2011], and uses an error defined by

$$e_{\nabla\rho} = \max\left(\frac{|(\nabla l_n\rho)_C - (\nabla l_n\rho)_{own}|}{\alpha_f\rho_C/dl + |\nabla l_n\rho)_{own}|}, \frac{|(\nabla l_n\rho)_C - (\nabla l_n\rho)_{nei}|}{\alpha_f\rho_C/dl + |\nabla l_n\rho)_{nei}|}\right)_f, \tag{128}$$

with

$$(\nabla l_n \rho)_C = \frac{\rho_{nei} - \rho_{own}}{|\mathbf{r}_{nei} - \mathbf{r}_{own}|}, \tag{129}$$

$$(\nabla l_n \rho)_{nei,own} = (\nabla \rho)_{nei,own} \frac{\mathbf{r}_{nei} - \mathbf{r}_{own}}{|\mathbf{r}_{nei} - \mathbf{r}_{own}|}, \tag{130}$$

$\mathrm{r}_{own,nei}$ is the surface normal vector, and $dl = |\mathbf{r}_{nei} - \mathbf{r}_{own}$, is the distance between cell centers. This error estimator uses the total density.

6.1.3 Lohner

A final error estimation method based on Löhner [1987] is also available. It uses a scaled Laplacian to ensure an error bounded between 0 and 1 and indicates the smoothness of the solution. Due to the fact that the method was initially developed for a Cartesian grid, some modifications were required. Namely, the face value of the tracked field is interpolated using a cubic interpolation method. This allows non-zero smoothness across the faces while still allowing for the method to be used on a non-Cartesian grid.

$$e = \frac{|x_{own} - 2x_f + x_{nei}|}{|x_{own} - x_f| + |x_{nei} - x_f| + \varepsilon(|x_{own}| + 2|x_f| + |x_{nei}|)}, \tag{131}$$

where ε is a coefficient which is problem dependent, and x is any scalar field saved within the simulation.

Entry	Description
field	Field name used to compute error
ε	Scaling coefficient

6.2 Dynamic Load Balancing (Experimental)

The ability to use dynamic load balancing is available for 3D grids. Dynamic load balancing can greatly reduce the computational time required by balancing the cells equally across all processors. However, there are known limitations: due to the way the OpenFOAM transfers data between processors, the MPI buffer limit can be exceeded for large cases, leading to the simulation crashing. Due to this, load balancing is still considered experimental and should be used with caution.

6.3 Example

An example of the optional `dynamicMeshDict` for use with *adaptiveFvMesh* class

```
// BlastFoam specific AMR mesh type
dynamicFvMesh   adaptiveFvMesh;

// Error is calculated using the density gradient
errorEstimator  densityGradient;

loadBalance // Can only be used with 3-D geometries
{
    // Is the mesh dynamically balanced
    balance yes;

    // How many refinement steps between rebalancing
    balanceInterval 20;

    // Allowable difference between average load
    // and max load
    allowableImbalance 0.2;

    // Decomposition method
    method scotch;
}

// How often to refine
refineInterval  1;

// When to begin unrefinement
beginUnrefine 1e-5;

// Field to be refinement on
// (error field is specific to blastFoam)
field           error;

// Refine field in between lower..upper
```

```
lowerRefineLevel 1e-2;
upperRefineLevel 1e6; // Large number

// If value < unrefineLevel unrefine
unrefineLevel   1e-2;

// Number of cells between level
nBufferLayers   1;

// Refine cells only up to maxRefinement levels
maxRefinement   4;

// Stop refinement if maxCells reached
maxCells        1000000;

// Flux field and corresponding velocity field.
// Fluxes on changed faces get recalculated by
// interpolating the velocity.
// Not currently used since fluxes are calculated
// at each time step
// Default is none
correctFluxes
(
    (phi none)
    (own none)
    (nei none)
    (alphaPhi.copper none)
    (alphaPhi.gas none)
    (rhoPhi none)
    (alphaRhoPhi.copper none)
    (alphaRhoPhi.gas none)
    (rhoEPhi none)
    (rhoUPhi none)
);

// Write the refinement level as a volScalarField
dumpLevel       true;
```

7 Utilities

7.1 *setRefinedFields*

This is a modified version of the *setFields* utility in the standard OpenFOAM that includes the ability to refine the mesh based on the fields that are set. This is extremely beneficial when a small volume needs to be set within a set of very large computational cells. The same functionality presented in `adaptiveFvMesh` is present allowing for 2D and 3D geometries to be set. The utility works by setting the default field values, setting the cell or face sets, then checking the difference in a specified set of fields across all faces. Each region must have a level defined inside where the mesh is refined up to this level. The cells inside the region can also be refined using the *refineInternal* keyword. The default field values for any fields being set must also be defined so that the region is updated at each iteration. The utility will iterate until the mesh is no longer changing, or the maximum number of iterations is reached. A comparison using the standard *setFields* and setRefinedFields can be seen in Fig. 1.

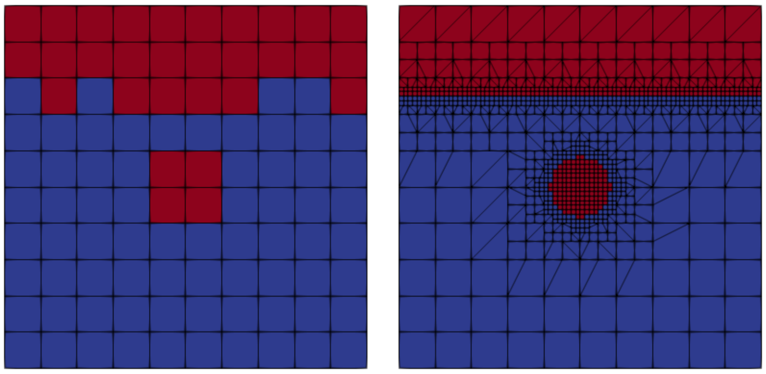

(a) Unrefined initial field (b) Refined initial field

Figure 1: Comparison of initial fields using *setFields* and *setRefinedFields*

In comparison with *setFields*, the following entries are specific to *setRefinedFields*

Entry	Description
fields	Field names that are used to compute error and refine mesh
level	Level of each cell set
refineInternal	Are the cells inside each region refined
nBufferLayer	Number of cells between level boundaries
backup	Dictionary used if a cell set does not contain any cells
maxCells	Maximum number of cells (default is the max interger)
maxIter	Maximum number of iterations (default is 2*max(level)

An example `setFieldsDict` is listed below

```
fields (alpha.air);
nBufferLayers 1;

defaultFieldValues
(
    volVectorFieldValue U ( 0 0 0 )
    volScalarFieldValue p 1.1e5
    volScalarFieldValue alpha.air 0
    volScalarFieldValue rho.air 1
    volScalarFieldValue rho.water 1000
);


regions
(
    cylinderToCell
    {
        p1 (0 0 -1);
        p2 (0 0 1);
        radius 0.1;
        level 3;
        refineInternal yes;
        backup
        {
            p1 (0 0 -1);
            p2 (0 0 1);
            radius 0.2;
        }

        fieldValues
        (
            volScalarFieldValue p 9.12e8
            volScalarFieldValue alpha.air 1
            volScalarFieldValue rho.air 1270
        );
    }

    boxToCell
    {
        boxes ((-0.6 0.3 -1) (0.6 0.6 1));

        fieldValues
        (
            volScalarFieldValue alpha.air 1
        );
    }
);
```

NOTE: Because OpenFOAM does not decompose both the time-based mesh (e.g. *0, 0.001, ...*) and the base mesh (located in *constant*), where both are required to use the mesh modified by *setRefinedFields*, if it is run in parallel, *setRefinedFields* must be run in parallel ***after*** the case has been decomposed.

7.2 *blastToVTK*

In order to save on space, it can often be preferable to write VTK files at timesteps between the specified output times. Due to the way that OpenFOAM writes this data, it can become tedious to view the time series of each data set. In order to make this process simpler, the utility *blastToVTK* will create symbolic links (or create copies) within the VTK directory in the case path so that that the collections of VTK files can be viewed in Paraview in a time series. The real time values are not present due to the fact that they are not written when the VTK files are, instead they are listed using integer indexes. The files will be in the correct order when viewing the collection.

By default only new VTK files will have symbolic links created, however, if the user would like to make hard copies of the files, the ”-hardCopy” option can be used (this may be required on Windows-based platforms). If `blastToVTK` needs to be rerun, the ”-force” option should be used to replace the previous links.

8 Function objects

In addition to the standard OpenFOAM `functionObjects`, several additional blast specific function object have been added. These can be included using the following `controlDict` entry

```
functions
{
    blastFunctionObject
    {
        type    impulse;
        libs    ("libbastFunctionObject.so");

        // writeTime - Writes every output time
        // timeStep - Number of time steps
        // adjustableRunTime - Specified output interval
        writeControl    writeTime;

        // Interval between writes
        writeInterval 1;

        ...
    }
}
```

8.1 Mach number (singlePhaseMachNo)

The *singlePhaseMachNo* replaces the standard machNo available in OpenFOAM by using the speed of sound calculated from the *blastFoam* specific thermodynamic models.

Entry	Description
phaseName	Name of the phase (default is none)

8.2 Speed of sound (speedOfSound)

Similar to the *singlePhaseMachNo*, the *speedOfSound* function object used the calculated speed of sound from the thermodynamic model and writes the field. The field can also be saved within the simulation to allow for sampling with probes or other function objects.

Entry	Description
phaseName	Name of the phase (default is none)
store	Is the field saved in the simulation (default in no)

8.3 dynamicPressure

It can be useful in blast simulations to track the dynamic pressure of a phase defined as

$$p_{dyn} = \frac{1}{2}\rho_i \mathbf{U}|\mathbf{U}|. \tag{132}$$

The function object returns the vector of dynamic pressure, where the magnitude of this output is the standard dynamic pressure. The field can also be saved within the simulation so that additional sampling can be conducted (i.e. probes for time series sampling).

Entry	Description
rhoName	Name of the density field (default is rho)
UName	Name of the velocity field (default is U)
store	Is the field stores (default is no)

8.4 Max field value over time (fieldMax)

The maximum value at every cell in a domain can be output, allowing for quantities such as maximum impulse and peak pressure to be visualized. This can be done for any field currently stored within the simulation. These fields are automatically saved within the calculation.

NOTE: When AMR is used, the unrefined fields use the volume averaged quantities so the maximum values may reduce when the mesh is unrefined. Therefore, the maximum value within a cell may actually be greater than is seen if unrefinement has occurred locally.

Entry	Description
fields	Name of the fields (i.e (rho U p impulse))

8.5 impulse

This function object calculates the impulse in every cell by integrating the overpressure over time using the specified pressure field and reference pressure. The impulse field is automatically saved to allow additional sampling.

Entry	Description
pName	Name of the pressure field (default is p)
pRef	Rreference pressure value [Pa]

8.6 overpressure

This function object calculates and writes the overpressure (e.g. the pressure relative to a given reference pressure, usually ambient) given a pressure field and reference pressure. The field can also be saved to allow for additional sampling.

Entry	Description
pName	Name of the pressure field (default is p)
store	Is the field saved (default is no)

9 Solvers

blastFoam offers three main solvers for simulating detonations.

9.1 blastFoam

The blastFoam solver is the standard solver and has a variety of uses, including single phase, two phase and multiphase flows, where the phases and thermodynamic models are specified in the `phaseProperties`. When a list of phases is provided (and the list has more than one entry), the two phase or multiphase solver is selected. If the key word phases is not specified, then the single phase solver is selected. All number of phases use the *blastFoam* specific equations of states (see Section 4). This solver has been specifically designed for simulating detonating explosive materials and has the option to use any of the models presented.

9.2 blastXiFoam (Experimental)

In addition to the standard blastFoam solver, an additional solver has been added to solve combustion as an extension of the standard OpenFOAM XiFoam solver. The only difference is that the flux schemes presented earlier are used to transport the conserved quantities. This allows for a more accurate description of combusting, highly compressible flows (i.e. deflagration to detonation transition). It is recommended to use the *adibaticFlameT* utility to calculate the model coefficients used in the required *thermophysicalProperties* dictionary.

9.3 blastReactingFoam (Experimental)

blastReactingFoam uses the standard OpenFOAM thermodynamic classes, and can be used to solve multi-specie systems that include reactions. The use of combustion models are also available. The `thermophysicalProperties` dictionary must be included under the constant dictionary along with the relevant dictionaries (`combustionProperties` or `chemistyProperties`) and initial value files (mass fractions). The conserved quantities (mass, momentum, and energy) as well as the mass fractions are all transported using the flux schemes presented herein.

10 Example Cases

Two validation case and one tutorial case are presented to show the general case setup and run procedure. The first case is a single phase, double mach reflection problem. The second is a simple 1-D shock tube consisting of air and water. The final case is a 2-D, three-phase case with two different detonating phases in which the first explosive material is used to detonate the second charge using the pressure based activation model. Only the new or modified files will be shown, but the other required files such as `blockMeshDict` and the initial conditions can be found in the case directories (`validation/blastFoam/doubleMachReflection`, `validation/blastFoam/shockTube_twoFluid`, and `tutorials/blastFoam/twoChargeDetonation`).

10.1 Double mach reflection

The double mach reflection problem is a complex case based on Woodward and Colella [1984] which is meant to demonstrate the capabilities of the single phase solver within *blastFoam*. Coded boundary conditions for velocity, pressure and density are used to correctly describe the shock behavior at the top boundary. The case uses the *singlePhaseCompressibleSystem* along with the RK2-SSP ODE time integrator and HLLC flux scheme.

The challenge in setting up the double mach reflection is due to the fact that the boundary conditions must be set so that the fields on the top boundary follow the correct upstream and downstream shock conditions which are

$$\rho_R = 1.4 \text{ kg/m}^3,\ p_L = 1 \text{ Pa},\ T_R = 1 \text{ [K]},\ \mathbf{U}_R = (0,\ 0,\ 0) \text{ m/s},$$

$$\rho_L = 8\,\text{kg/m}^3\ p_L = 116.5\,\text{Pa},\ T_R = 20.3887\,\text{[K]}\ \mathbf{U}_L = (7.1447,\ -4.125,\ 0)\,\text{m/s}.$$

These conditions change based on the shock location, leading to a modified boundary condition for density, pressure, temperature, and velocity which uses a codedFixedValue, i.e. the initial density field is defined using

```
dimensions      [1 -3 0 0 0 0 0];

internalField   uniform 1.4;

boundaryField
```

```
{
    inlet1
    {
        type                fixedValue;
        value               $internalField;
    }
    inlet2
    {
        type                fixedValue;
        value               $internalField;
    }
    outlet
    {
        type                codedFixedValue;
        value               $internalField;
        name                shockRho;
        code
        #{
            // Shock location (x)
            scalar xS =
                db().time().value()*11.2799
              + 0.74402;
            scalarField x =
                this->patch().Cf().component(0);

            operator==
            (
                pos(x - xS)*1.4
              + neg0(x-xS)*8.0
            );
        #};
    }
    walls
    {
        type                zeroGradient;
    }

    defaultFaces
    {
        type            empty;
    }
}
```

The velocity and temperature fields are initialized using the upstream and downstream values with the same evolution equations. The pressure field used zeroGradient boundary conditions. The internal and boundaries are set using *setFields* with the `setFieldsDict` defined as

```
defaultFieldValues
(
    volVectorFieldValue U (7.1447 -4.125 0)
    volScalarFieldValue T 20.3887
    volScalarFieldValue p 116.5
    volScalarFieldValue rho 8.0
);

regions
(
    rotatedBoxToCell
    {
        origin (0.16667 0 -1);
        i (8.66025 -5.0 0);
        j (5.0 8.66025 0 );
        k (0 0 2);
        fieldValues
        (
            volScalarFieldValue p 1
            volScalarFieldValue T 1
            volVectorFieldValue U (0 0 0)
            volScalarFieldValue rho 1.4
        );
    }
    boxToFace
    {
        box (0.74402 0.999 -1) (5 1.001 1);
        fieldValues
        (
            volScalarFieldValue p 1
            volScalarFieldValue T 1
            volVectorFieldValue U (0 0 0)
            volScalarFieldValue rho 1.4
        );
    }
);
```

The phaseProperties file is

```
mixture
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
    }
    specie
    {
        molWeight   11640.3; // normalizes pressure
    }
    transport
    {
        mu          0;
        Pr          1.0;
    }
    equationOfState
    {
        gamma       1.4;
    }
    thermo
    {
        Cv          1.785714286;
        Hf          0.0;
    }
}
```

The case is then initialized and run using the following commands:

```
blockMesh
setFields
blastFoam
```

The final results can be seen below.

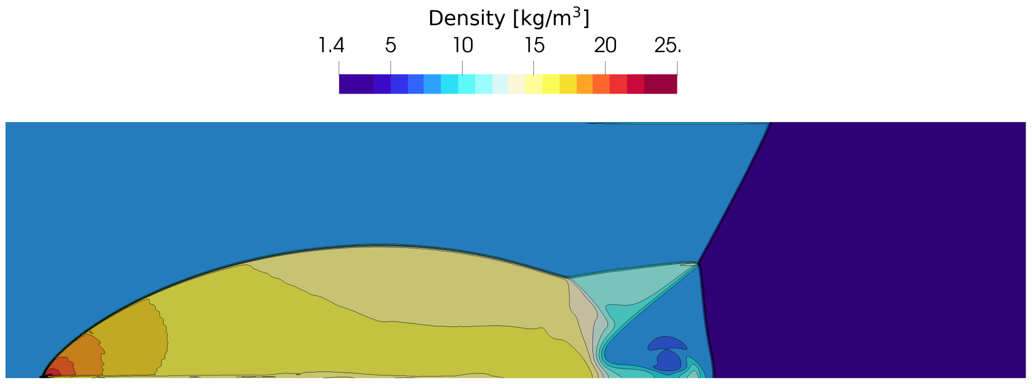


Figure 2: Density contours at t = 200 ms

10.2 Shock Tube - Two Fluid

The presented case is taken from Zheng et al. [2011] and consists of a gas using the van der Waals EOS and water using the Stiffened gas EOS initially separated at the center of the domain. The shock tube is 1 m long, and discretized into 300 computational cells.

The left state is defined as:

$$\rho = 1000 \text{ kg/m}^3 \quad \alpha = 1 \quad p = 1e9 \text{ Pa.}$$

The right state is defined as:

$$\rho = 50 \text{ kg/m}^3 \quad \alpha = 0 \quad p = 1e5 \text{ Pa.}$$

The `phaseProperties` dictionary is:

```
phases  (fluid gas);
```

```
fluid
{
    type          basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState stiffenedGas;
    }
    specie
    {
        molWeight   18.0;
    }
    transport
    {
        mu          0;
        Pr          1.0;
    }
    equationOfState
    {
        gamma       4.4;
        a           6.0e8;
    }
    thermo
    {
        Cv          4186;
        Hf          0.0;
    }

    residualRho 1e-6;
    residualAlpha 1e-10;
}

gas
{
    type          basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState vanderWaals;
    }
    specie
    {
        molWeight   28.97;
    }
    transport
    {
        mu          0;
        Pr          1.0;
    }
    equationOfState
    {
        gamma       1.4;
        a           5.0;
        b           1e-3;
    }
    thermo
    {
        Cv          718;
        Hf          0.0;
    }

    residualRho 1e-6;
    residualAlpha 1e-10;
}
```

and the `fvSchemes` dictionary is:

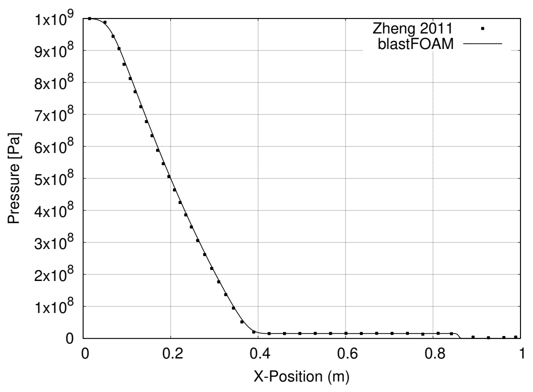
```
fluxScheme      HLLC;

ddtSchemes
{
    default         Euler;
    timeIntegrator  RK2SSP;
}
```

```
gradSchemes
{
    default         leastSquares;
}

divSchemes
{
    default         none;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default             cubic;
    reconstruct(alpha)  vanLeer;
    reconstruct(rho)    vanLeer;
    reconstruct(U)      vanLeerV;
    reconstruct(e)      vanLeer;
    reconstruct(p)      vanLeer;
    reconstruct(c)      vanLeer;
}

snGradSchemes
{
    default         corrected;
}
```

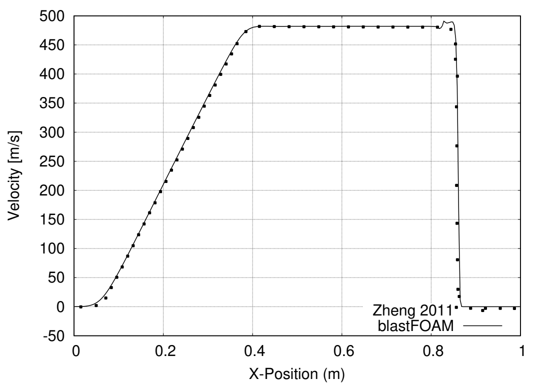
As seen above, the HLLC flux scheme is used with cubic interpolation, vanLeer limiters, and 2nd-order strong-stability-preserving time integration. The following commands were used to initialize and run the case in serial:

```
blockMesh
setFields
blastFoam
```

The results are shown in Fig. 3.



(a) Pressure



(b) X-velocity

Figure 3: Pressure and velocity fields at t = 73 ms

10.3 Two charge detonation

The final case is meant to show the capabilities of *blastFoam* to solve complex problems involving multiple phases and detonation dynamics. Two charges, one composed of RDX and the other of TNT are detonated in air. The RDX has a single detonation point at the center of its circular domain, and the TNT is detonated using the pressure based activation model. Fig. 4 shows the initial charge where the red lines denote the outline of the RDX charge, the red dot the RDX initiation point, and the blue line the outline of the TNT charge.

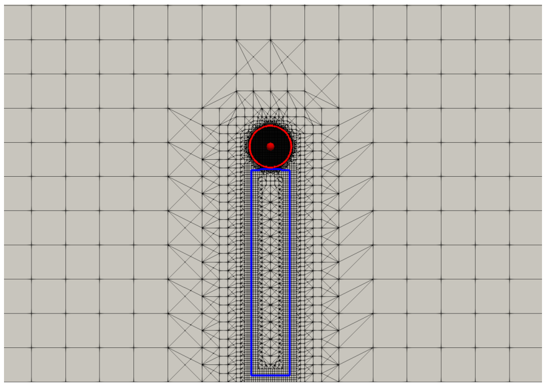


Figure 4: Initial charge configuration

The `phaseProperties` dictionary is:

```
phases (RDX tnt gas);

RDX
{
    type detonating;
    reactants
    {
        thermoType
        {
            transport   const;
            thermo      eConst;
            equationOfState Murnaghan;
        }
        equationOfState
        {
            rho0    1160;
            n       7.4;
            kappa   3.9e11;
            Gamma   0.35;
            pRef    101298;
        }
        specie
        {
```

```
        molWeight 222.12;
    }
    transport
    {
        mu      0;
        Pr      1;
    }
    thermodynamics
    {
        Cv      1130.78;
        Hf      0.0;
    }
}
products
{
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState JWL;
    }
    equationOfState
    {
        A       2.9867e11;
        B       4.11706e9;
        R1      4.95;
        R2      1.15;
        C       7.206147e8;
        omega   0.35;
        rho0    1160;
    }
    specie
    {
        molWeight 55.0;
    }
    transport
    {
        mu      0;
```

```
            Pr      1;
        }
        thermodynamics
        {
            Cv      1000;
            Hf      0.0;
        }
    }

    activationModel none;
    initiation
    {
        E0      4.0e9;
    }

    residualRho     1e-6;
    residualAlpha   1e-10;
}

tnt
{
    type detonating;
    reactants
    {
        thermoType
        {
            transport   const;
            thermo      eConst;
            equationOfState BirchMurnaghan3;
        }
        equationOfState
        {
            rho0    1601; // Reference density [kg/m^3]
            K0      9.6e9; // Bulk modulus [Pa]
            K0Prime 6.6; // dK0/dp []
            Gamma   0.35; // Gruneisen coefficient
            pRef    101298; // Reference pressure [Pa]
        }
        specie
        {
            molWeight 227.13;
        }
        transport
        {
            mu      0;
            Pr      1;
        }
        thermodynamics
        {
            Cv      1095;
            Hf      0.0;
        }
    }
    products
    {
        thermoType
        {
            transport   const;
            thermo      eConst;
            equationOfState JWL;
        }
        equationOfState
        {
            rho0    1601;
            A       371.21e9;
            B       3.23e9;
            R1      4.15;
            R2      0.95;
            omega   0.3;
        }
        specie
        {
            molWeight 55.0;
        }
        transport
        {
            mu      0;
            Pr      1;
        }
        thermodynamics
        {
            Cv      1000;
            Hf      0.0;
        }
    }

    activationModel pressureBased;
    initiation
    {
        E0      7.0e9; // Detonation energy [Pa]

        // Ignition coefficients
        I       0; // Ignition rate [1/s]

        // First stage coefficients
        G1      3.5083e-7; // Activation rate [Pa/s]
        c       1.0; // (1-lambda) exponent
        d       0.0; // lambda exponent
        y       1.3; // pressure exponent
        minLambda1  0.0; // 1st stage start value

        // Second stage coefficients
        G2      0.0; // Activation rate [Pa/s]

        pMin    1.1e5; // Cutoff pressure
    }

    residualRho     1e-6;
    residualAlpha   1e-6;
}

gas
{
    type basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
    }
    equationOfState
    {
        gamma   1.4;
    }
    specie
    {
        molWeight 28.97;
    }
    transport
    {
        mu      0;
        Pr      1;
    }
    thermodynamics
    {
        Cv      718;
        Hf      0;
    }

    residualRho     1e-6;
    residualAlpha   1e-6;
}
```

The HLLC flux scheme is used with cubic interpolation, vanLeer limiters, and 2nd-order strong-stability-preserving time integration. Below is the `fvSchemes` dictionary

```
fluxScheme      HLLC;

ddtSchemes
{
    default         Euler;
    timeIntegrator  RK2SSP;
}

gradSchemes
{
    default cellMDLimited leastSquares 1;
}

divSchemes
{
    default         none;
    div(alphaRhoPhi.tnt,lambda.tnt) Riemann;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default             cubic;
    reconstruct(alpha)  vanLeer;
    reconstruct(rho)    vanLeer;
    reconstruct(U)      vanLeerV;
    reconstruct(e)      vanLeer;
    reconstruct(p)      vanLeer;
    reconstruct(c)      vanLeer;
    reconstruct(lambda.tnt) vanLeer;
}

snGradSchemes
{
    default         corrected;
}
```

Because the circular charge domain is not well represented by the hexahedral mesh generated by *blockMesh*, refinement around this region is used.

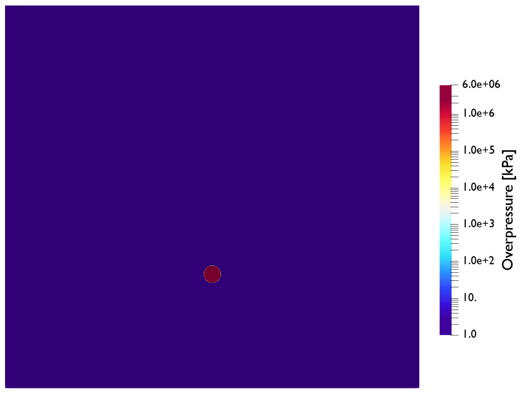
A maximum of four levels of refinement are used, with two cells between each level. This allows for a more accurate initial mass of the charges, and provides better initial resolution of the solution. The *setRefinedFields* utility is used where the `setFieldsDict` is

```
fields (alpha.RDX alpha.tnt);
nBufferLayers 1;



defaultFieldValues
(
    volScalarFieldValue alpha.gas       1
    volScalarFieldValue alpha.RDX       0
    volScalarFieldValue alpha.tnt       0
);

regions
(
    cylinderToCell
    {
        refineInternal yes;
        level 6;

        p1 (0 0.55 -1);
        p2 (0 0.55 1);
        radius 0.05;

        fieldValues
        (
            volScalarFieldValue alpha.RDX   1
            volScalarFieldValue alpha.gas   0
        );
    }
    boxToCell
    {
        level 5;
        refineInternal no;
        box (-0.05 0.01 -1) (0.05 0.5 1);

        fieldValues
        (
            volScalarFieldValue alpha.tnt   1
            volScalarFieldValue alpha.gas   0
        );
    }
 );
```

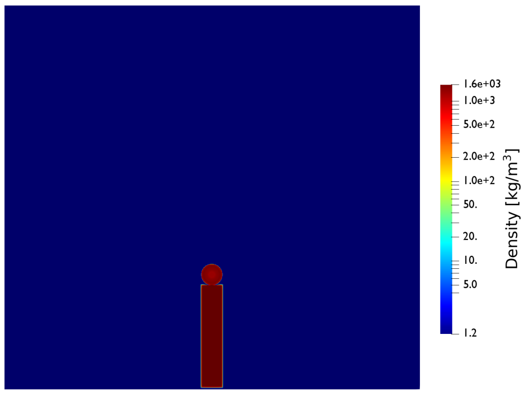
The following commands were used to run the case in parallel:

```
blockMesh
decomposePar
mpirun -np $nProcs setRefineFields -parallel
mpirun -np $nProcs blastFoam -parallel
```

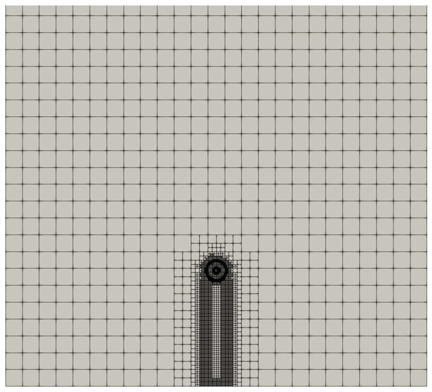
The instantaneous overpressure and density fields can be seen in Fig. 5, Fig. 6, and Fig. 7.



(a) Overpressure

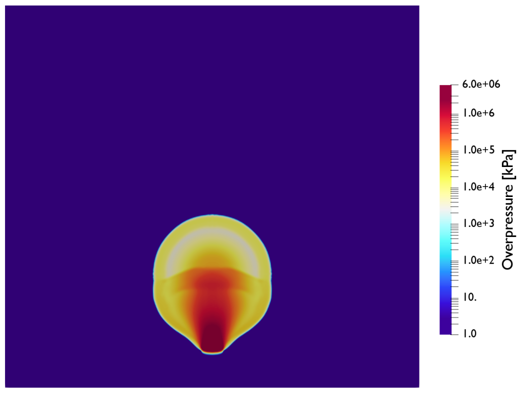


(b) Density

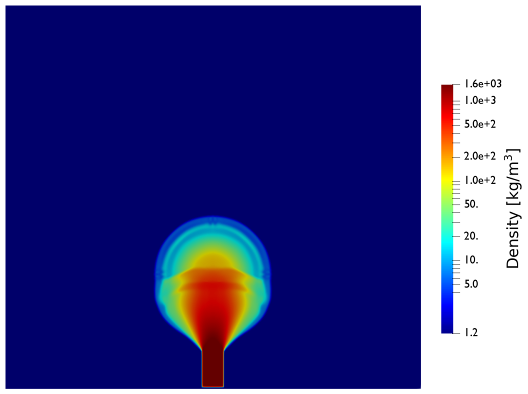


(c) Mesh

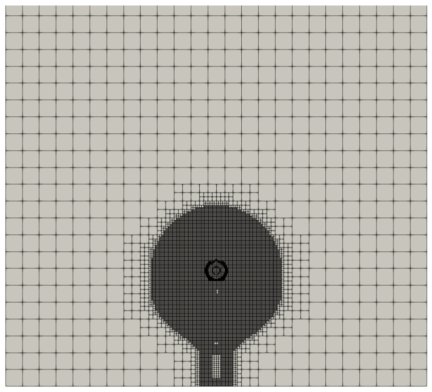
Figure 5: Instantaneous fields at t = 0.005 ms



(a) Overpressure

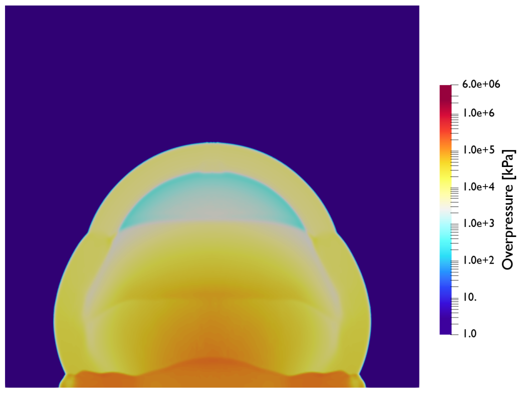


(b) Density

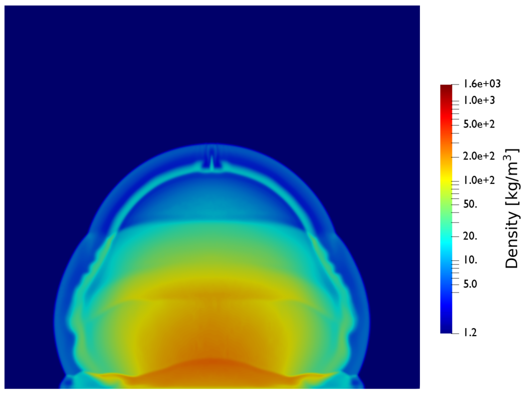


(c) Mesh

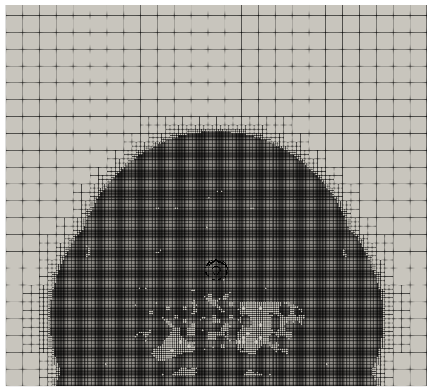
Figure 6: Instantaneous fields at t = 0.05 ms



(a) Overpressure



(b) Density



(c) Mesh

Figure 7: Instantaneous fields at t = 0.15 ms

References

Manson Benedict, George B. Webb, and Louis C. Rubin. An Empirical Equation for Thermodynamic Properties of Light Hydrocarbons and Their Mixtures II. Mixtures of Methane, Ethane, Propane, and n-Butane. *J. Chem. Phys.*, 10(12):747–758, December 1942. ISSN 0021-9606. doi: 10.1063/1.1723658.

L. R. Doan and G. H. Nickel. A subroutine for the equation of state of air. Technical Report RTD (WLR) TN63-2, Air Force Weapons Labratory, 1963.

Christopher J. Greenshields, Henry G. Weller, Luca Gasparini, and Jason M. Reese. Implementation of semi-discrete, non-staggered central schemes in a colocated, polyhedral, finite volume framework, for high-speed viscous flows. *International journal for numerical methods in fluids*, 63(1):1–21, 2010.

Alexander Kurganov and Eitan Tadmor. New high-resolution central schemes for nonlinear conservation laws and convection–diffusion equations. *Journal of Computational Physics*, 160(1):241–282, 2000.

Rainald Löhner. An adaptive finite element scheme for transient problems in CFD. *Computer Methods in Applied Mechanics and Engineering*, 61(3):323–338, April 1987. ISSN 0045-7825. doi: 10.1016/0045-7825(87)90098-3.

Hong Luo, Joseph D Baum, and Rainald Löhner. On the computation of multi-material flows using ALE formulation. *Journal of Computational Physics*, 194(1):304–328, February 2004. ISSN 00219991. doi: 10.1016/j.jcp.2003.09.026.

Morton Lutzky. The flow field behind a spherical detonation in TNT using the landau-stanyukovich equation of state for detonation products. Technical report, NAVAL ORDNANCE LAB WHITE OAK MD, 1964.

Charles Mader. Detonation properties of condensed explosives computed using the becker-kistiakowsky-wilson equation of state. February 1963.

Philip J. Miller. A Reactive Flow Model with Coupled Reaction Kinetics for Detonation and Combustion in Non-Ideal Explosives. *MRS Online*

Proceedings Library Archive, 418, 1995/ed. ISSN 0272-9172, 1946-4274. doi: 10.1557/PROC-418-413.

Charles E. Needham. *Blast Waves*. Shock Wave and High Pressure Phenomena. Springer International Publishing, Cham, 2018. ISBN 978-3-319-65381-5 978-3-319-65382-2. doi: 10.1007/978-3-319-65382-2.

OpenCFD Ltd. *OpenFOAM - The Open Source CFD Toolbox - Programer's Guide*. United Kingdom, second edition, 2018a.

OpenCFD Ltd. *OpenFOAM - The Open Source CFD Toolbox - User's Guide*. United Kingdom, second edition, 2018b.

Keh-Ming Shyue. A Fluid-Mixture Type Algorithm for Compressible Multicomponent Flow with Mie–Grüneisen Equation of State. *Journal of Computational Physics*, 171(2):678–707, August 2001. ISSN 00219991. doi: 10.1006/jcph.2001.6801.

P Clark Souers, Steve Anderson, James Mercer, Estella McGuire, and Peter Vitello. JWL++: A Simple Reactive Flow Code Package for Detonation. page 5, 2000.

Raymond J. Spiteri and Steven J. Ruuth. A New Class of Optimal High-Order Strong-Stability-Preserving Time Discretization Methods. *SIAM Journal on Numerical Analysis*, 40(2):469–491, January 2002. ISSN 0036-1429, 1095-7170. doi: 10.1137/S0036142901389025.

Eleuterio F. Toro, Michael Spruce, and William Speares. Restoration of the contact surface in the HLL-Riemann solver. *Shock waves*, 4(1):25–34, 1994.

Paul Woodward and Phillip Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics*, 54(1):115–173, April 1984. ISSN 0021-9991. doi: 10.1016/0021-9991(84)90142-6.

Wenjia Xie, Ran Zhang, Jianqi Lai, and Hua Li. An accurate and robust HLLC-type Riemann solver for the compressible Euler system at various Mach numbers. *International Journal for Numerical Methods in Fluids*, 89(10):430–463, 2019. ISSN 1097-0363. doi: 10.1002/fld.4704.

H. W. Zheng, C. Shu, Y. T. Chew, and N. Qin. A solution adaptive simulation of compressible multi-fluid flows with general equation of state. *International Journal for Numerical Methods in Fluids*, 67(5):616–637, 2011. ISSN 1097-0363. doi: 10.1002/fld.2380.

Zhaoyang Zheng and Jijun Zhao. Unreacted equation of states of typical energetic materials under static compression: A review. *Chinese Phys. B*, 25(7):076202, July 2016. ISSN 1674-1056. doi: 10.1088/1674-1056/25/7/076202.