Combining the Density-Based Compressible Solver *rhoCentralFoam* with the Reacting Flow Solver *reactingFoam*

Pablo Kandel

Data Analysis and Modeling of Turbulent Flows,

Technische Universität,

Berlin, Germany

January 20, 2025

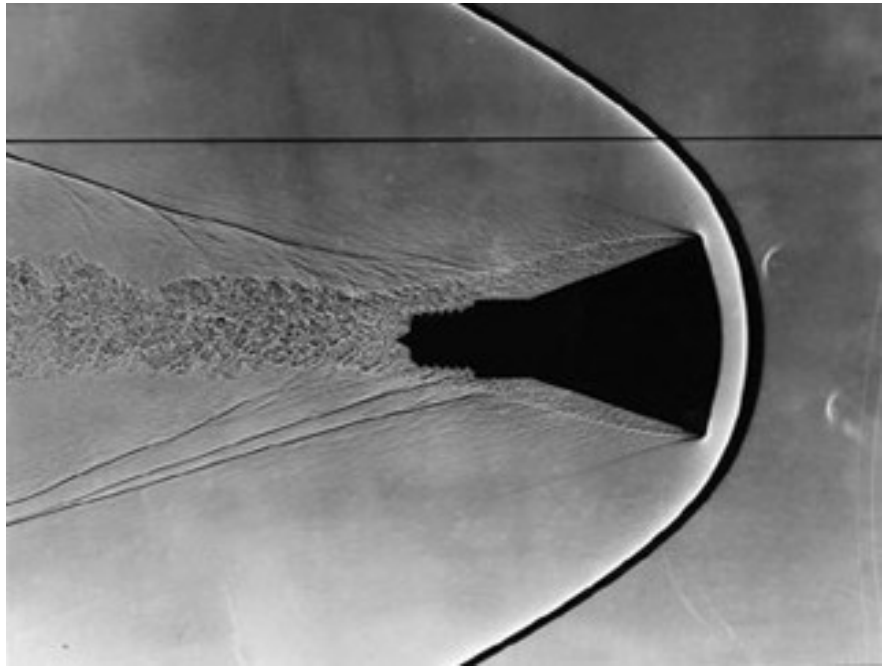
Introduction

- Compressible flow occurs when fluid density changes significantly due to pressure or temperature variations :
  - High-speed aerodynamics.
  - Propulsion systems.
  - Gas dynamics.
  - Combustion.

Mach number : $Ma = \frac{U}{c}$

$Ma > 0.3,$ compressibility effects become significant.

U.S Navy F/A-18 transonic pushing into the sound barrier [1]



NASA Mercury capsule [2]

Aspect	Pressure-Based	Density-Based
Algorithm	SIMPLE, PISO	Godunov methods
Time-Stepping Scheme	Implicit	Explicit
Numerical Diffusion	Introduce numerical diffusion, smearing sharp gradients	Less numerical diffusion, better for sharp gradients like shocks
High-Order Schemes	More complex to implement	Easier to implement in explicit frameworks
Applications	Steady-state problems, transient flows with large time steps	Compressible flows, shocks, and contact discontinuities
Challenges	Numerical diffusion, reduced accuracy for convection dominated flows	Strict time step restrictions, less robust for low-speed or transient flows

- Reacting flow refers to fluid movement that involves chemical reactions, such as combustion, where the transport of species and heat release significantly influence the flow dynamics.

- Role of CFD:
  - Predict performance and optimize processes in industries like aerospace, automotive, and power generation.
  - Critical for analyzing flame stability, pollutant formation and combustion efficiency.

- The most widely used reacting flow solver in OpenFOAM is *reactingFoam*:
  - Employs a pressure-based approach (PIMPLE algorithm).
  - No density-based reacting flow solver exists in the official OpenFOAM release.

- Density-based approach is relevant for combustion :
  - Better handling of high-speed flows and large density variations, such as those encountered in supersonic or hypersonic combustion.
  - Captures acoustic-wave dynamics crucial for capturing interactions between heat release and acoustic phenomena, particularly in combustion systems susceptible to thermoacoustic instabilities.

Theoretical Background

Reacting Flow Governing Equations

- Continuity, momentum, species transport and energy equations :

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0$$

$$\frac{\partial \left(\rho u_i\right)}{\partial t} + \frac{\partial \left(\rho u_i u_j\right)}{\partial x_j} = -\frac{\partial p}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_j}$$

$$\frac{\partial \left(\rho Y_k\right)}{\partial t} + \frac{\partial \left(\rho u_i Y_k\right)}{\partial x_i} = -\frac{\partial J_{k,i}}{\partial x_i} + \dot{\omega}_k$$

$$\frac{\partial \rho\left(e_s + \frac{1}{2}u_j u_j\right)}{\partial t} + \frac{\partial}{\partial x_i}\left(\rho u_i\left(e_s + \frac{1}{2}u_j u_j\right)\right) = -\frac{\partial \left(u_i p\right)}{\partial x_i} - \frac{\partial q_i}{\partial x_i} + \dot{\omega}_T$$

$$\frac{\partial \rho\left(h_s + \frac{1}{2}u_j u_j\right)}{\partial t} + \frac{\partial}{\partial x_i}\left(\rho u_i\left(h_s + \frac{1}{2}u_j u_j\right)\right) = \frac{\partial p}{\partial t} - \frac{\partial q_i}{\partial x_i} + \dot{\omega}_T$$

$$\rho = \frac{W}{RT} p = \psi p$$

- Species *k* mass fraction :

$$Y_k = \frac{m_k}{m}, \quad \sum_{k=0}^{N} Y_k = 1$$

- Diffusive flux, assuming Fick's law :

$$J_{k,i} = \rho D_k \frac{\partial Y_k}{\partial x_i}$$

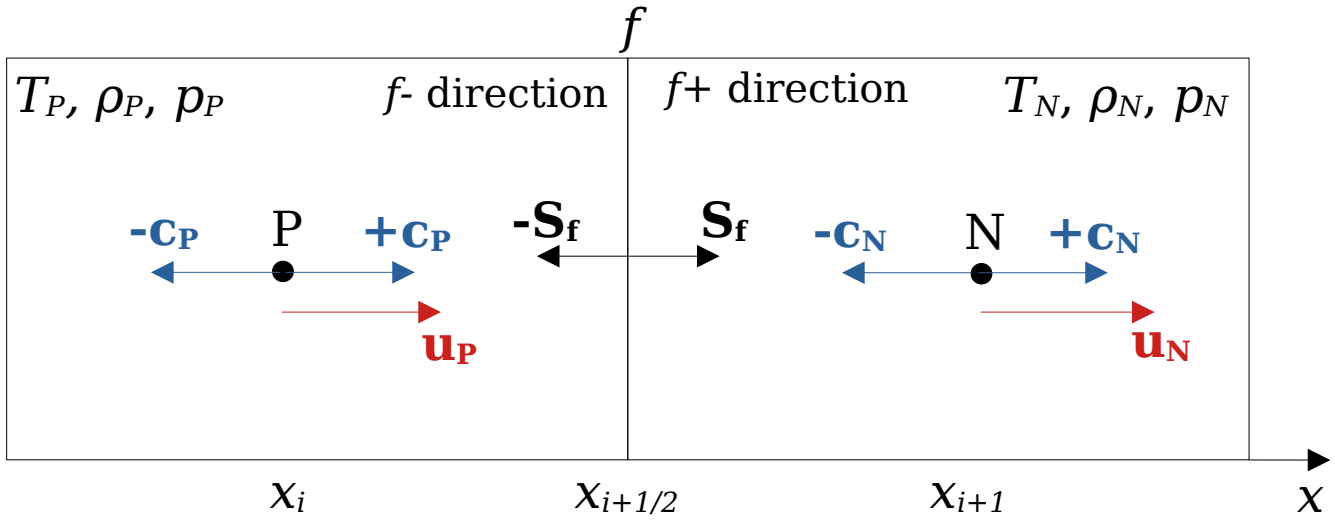
- Unity Lewis and Prandtl numbers :

$$\rho D_k = \mu$$

- Heat flux :

$$q_i = \lambda \frac{\partial T}{\partial x_i}$$

- Central schemes introduced by Kurganov et al. [3], also refferred to as Kurganov-Tadmor's scheme (KT) and Kurganov-Noelle-Petrova scheme (KNP).
- Discretization of the convective fluxes $\nabla \cdot \boldsymbol{u}\Psi$, taking into account the propagation of waves in any direction.



- Integrated over a control volume and linearized :

$$\int_V \nabla \cdot (\boldsymbol{u}\Psi) dV = \int_S \boldsymbol{u}\Psi \cdot d\boldsymbol{S} \approx \sum_f \boldsymbol{S_f} \cdot \boldsymbol{u_f} \Psi_f = \sum_f \phi_f \Psi_f$$

- Interpolation is carried out in both directions :

$$\sum_f \phi_f \Psi_f = \sum_f \alpha \phi_{f+} \Psi_{f+} + (1-\alpha)\phi_{f-} \Psi_{f-} + \omega_f (\Psi_{f-} - \Psi_{f+})$$

- The weighting factor α :

$$\alpha = \begin{cases} \frac{1}{2}, & KT \\ \frac{a_{f+}}{a_{f+} - a_{f-}}, & KNP \end{cases}$$

- Volumetric fuxes relative to the speed of propagation :

$$a_{f+} = max(\phi_{f+} + c_{f+}|S_f|, \phi_{f-} + c_{f-}|S_f|, 0)$$

$$a_{f-} = min(\phi_{f+} - c_{f+}|S_f|, \phi_{f-} - c_{f-}|S_f|, 0)$$

- Diffusive volumetric flux :

$$\omega_f = \begin{cases} \alpha max(a_{f+}, a_{f-}), & KT \\ \alpha(1-\alpha)(a_{f+} + a_{f-}), & KNP \end{cases}$$

reactingFoam

- Construct the thermophysical model *thermo* based on the *constant/thermophysicalProperties* file :

```
Info<< "Reading thermophysical properties\n" << endl;
autoPtr<psiReactionThermo>
pThermo(psiReactionThermo::New(mesh));
psiReactionThermo& thermo = pThermo();
thermo.validate(args.executable(), "h", "e");

basicSpecieMixture& composition = thermo.composition();
PtrList<volScalarField>& Y = composition.Y();
```

reactingFoam/createFields.H

```
const volScalarField& psi = thermo.psi();
const volScalarField& T = thermo.T();
const label inertIndex(composition.species().find(inertSpecie));
```

reactingFoam/createFieldRefs.H

```
thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleEnthalpy;
    equationOfState perfectGas;
    specie          specie;
}

inertSpecie             N2;
chemistryReader         foamChemistryReader;
foamChemistryFile       "<constant>/reactions";
foamChemistryThermoFile "<constant>/thermo.compressibleGas";
```

constant/thermophysicalProperties

- Different combination of *thermoType* are possible.
- Sensible enthalpy *h* or sensible enargy *e* as energy variable.
- The list of pointers to store the species mass fractions *Y* is initialized, the total number and names of the species will depend on the files *constant/reactions* and *constant/thermo.compressibleGas*.
- The temperature *T*, and compressibility *ψ*, are created as references in the *createFieldRefs.H* file, as well as the *inertIndex* variable read from the *thermophysicalProperties* file.

- Example of *reactions* and *thermo.compressibleGas* files :

```
elements    3(H O Ar);

species     9(H2 O2 H O OH HO2 H2O2 H2O AR);

reactions
{
            un-named-reaction-0
            {
                        type        reversibleArrheniusReaction;
                        reaction    "H2 + O2 = 2OH";
                        A           1.7e+10;
                        beta        0;
                        Ta          24042.47739;
            }
            ...
}
```

constant/reactions

```
O2
{
            specie
            {
                        molWeight 31.9988;
            }
            thermodynamics
            {
                        Tlow        200;
                        Thigh       3500;
                        Tcommon     1000;
                        highCpCoeffs (...);
                        lowCpCoeffs (...);
            }
            transport
            {
                        As          0;
                        Ts          0;
            }
            elements
            {
                        O           2;
            }
            ...
}
```

constant/thermo.compressibleGas

- *reactions* defines the elements and species as well as the chemical reactions involved in the chemistry mechanism.
- *thermo* lists the dfferent thermophysical properties for each species.

```
volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh
    ),
    thermo.rho()
);

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

volScalarField& p = thermo.p();

#include "compressibleCreatePhi.H"
```

reactingFoam/createFields.H

```
Info<< "Creating reaction model\n" << endl;
autoPtr<CombustionModel<psiReactionThermo>> reaction
(
    CombustionModel<psiReactionThermo>::New(thermo, turbulence())
);

multivariateSurfaceInterpolationScheme<scalar>::fieldTable fields;

forAll(Y, i)
{
    fields.add(Y[i]);
}
fields.add(thermo.he());

volScalarField Qdot
(
    IOobject
    (
        "Qdot",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedScalar(dimEnergy/dimVolume/dimTime, Zero)
);
```

reactingFoam/createFields.H

- Creation of density, velocity and pressure fields as well as compressible mass fluxes.
- Pointer *reaction* is initialised and the species mass fractions *Y[i]* and energy variable are added to *fields* for interpolation.
- Finally, the heat release rate field *Qdot* is created.

reactingFoam/reactingFoam.C

```
#include "rhoEqn.H"

while (pimple.loop())
{
    #include "UEqn.H"
    #include "YEqn.H"
    #include "EEqn.H"

    // --- Pressure corrector loop
    while (pimple.correct())
    {
        [...]

        else
        {
            #include "pEqn.H"
        }
    }

    [...]

}

rho = thermo.rho();

[...]
```

reactingFoam/reactingFoam.C, main lines

- The density field is calculated by solving the continuity equation by including the *rhoEqn.H* header file.
- The momentum equation is solved in order to compute the velocity field in the *Ueqn.H* header file.
- The reacting species transport equations are solved in the *YEqn.H* header file.
- The energy equation is solved in the *EEqn.H* header file.
- At this point, we entered the *while (pimple.correct())* loop or the inner corrector loop, where the PISO algorithm is executed. This is the main difference with density-based solvers. The pressure-velocity-density coupling is solved through the SIMPLE and PISO algorithms instead of calculating the pressure from an equation of state.

reactingFoam/YEqn.H

```
reaction->correct();
Qdot = reaction->Qdot();
volScalarField Yt(0.0*Y[0]);

forAll(Y, i)
{
    if (i != inertIndex && composition.active(i))
    {
        volScalarField& Yi = Y[i];

        fvScalarMatrix YiEqn
        (
            fvm::ddt(rho, Yi)
          + mvConvection->fvmDiv(phi, Yi)
          - fvm::laplacian(turbulence->muEff(),
          Yi)
          ==
          reaction->R(Yi)
          + fvOptions(rho, Yi)
        );

        YiEqn.relax();

        fvOptions.constrain(YiEqn);

        YiEqn.solve("Yi");

        fvOptions.correct(Yi);

        Yi.clamp_min(0);
        Yt += Yi;
    }
}

Y[inertIndex] = scalar(1) - Yt;
Y[inertIndex].clamp_min(0);
```

reactingFoam/YEqn.H

- The lines *reaction->correct();* and *Qdot = reaction->Qdot();* compute all species reaction rates, and retrieves the updated heat release rate per unit volume, *Qdot*, from the turbulence-chemistry interaction model.
- One transport equation for each reacting species is solved (*forAll(Y, i)* loop).
- We find all the terms presented in slide 7:
  - Temporal derivative: *fvm::ddt(rho, Yi)*.
  - Advection term: *mvConvection→fvmDiv(phi, Yi)*.
  - Diffusion term: *fvm::laplacian(turbulence->muEff(), Yi)*, the diffusion coefficient is the effective kinematic viscosity *muEff*, which is the sum of the turbulent and laminar kinematic viscosity and depends on the chosen turbulence and thermophysical models.
- The term *reaction->R(Yi)* is the species production rate and corresponds to $\dot{\omega}_k$.
- Negative mass fractions avoided with *Yi.clamp min(0);* and added to *sum Yt += Yi;*.
- Mass fraction of the inert species computed *Y[inertIndex] = scalar(1) - Yt;*.

rhoCentralFoam

- Similarly, a *createFieldRefs.H* and a *createField.H* files are present :

```
volScalarField& p = thermo.p();
const volScalarField& T = thermo.T();
const volScalarField& psi = thermo.psi();

bool inviscid(true);
if (max(thermo.mu().cref().primitiveField()) > 0.0)
{
    inviscid = false;
}
```

rhoCentralFoam/createFieldRefs.H

```
Info<< "Reading thermophysical properties\n" << endl;

autoPtr<psiThermo> pThermo
(
    psiThermo::New(mesh)
);
psiThermo& thermo = pThermo();

volScalarField& e = thermo.he();
```

rhoCentralFoam/createField.H

- A boolean variable, *inviscid*, is created as *true* and then determined based on the viscosity *mu*.
- Like in *reactingFoam*, the *thermo* reference is created from the *constant/thermophysicalProperties* file to access and update the thermophysical properties.
- Also based on compressibility *psi* but the thermodynamic class is *psiThermo* and not *psiReactionThermo*.
- The energy variable here is called *e* instead of *he*, in *rhoCentralFoam* we should only choose the sensible internal energy as energy variable.

- The solver solves the conservative variables or density weighted fields, ρ *(rho)*, $\hat{u} = \rho u$ *(rhoU)*, $\hat{E} = \rho E$ *(rhoE)* :

```
volVectorField rhoU
(
    IOobject
    (
        "rhoU",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    rho*U
);

volScalarField rhoE
(
    IOobject
    (
        "rhoE",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    rho*(e + 0.5*magSqr(U))
);
```

rhoCentralFoam/createFields.H

- Created using the *rho*, *U*, *e* fields and the formula for the total energy $E = e + 0.5\|u^2\|$.

- Creation of fields *pos* and *neg*, as *surfaceScalarField* variables :

```
surfaceScalarField pos
(
    IOobject
    (
        "pos",
        runTime.timeName(),
        mesh
    ),
    mesh,
    dimensionedScalar("pos", dimless, 1.0)
);

surfaceScalarField neg
(
    IOobject
    (
        "neg",
        runTime.timeName(),
        mesh
    ),
    mesh,
    dimensionedScalar("neg", dimless, -1.0)
);
```

rhoCentralFoam/createFields.H

- Used in the solver as part of the central discretization process to take into account the propagation of waves in any direction, previously introduced as $f+$ and $f-$.

- Central flux scheme implementation in *rhoCentralFoam.C* :

```
surfaceScalarField rho_pos(interpolate(rho, pos));
surfaceScalarField rho_neg(interpolate(rho, neg));

surfaceVectorField rhoU_pos(interpolate(rhoU, pos, U.name()));
surfaceVectorField rhoU_neg(interpolate(rhoU, neg, U.name()));

volScalarField rPsi("rPsi", 1.0/psi);
surfaceScalarField rPsi_pos(interpolate(rPsi, pos, T.name()));
surfaceScalarField rPsi_neg(interpolate(rPsi, neg, T.name()));

surfaceScalarField e_pos(interpolate(e, pos, T.name()));
surfaceScalarField e_neg(interpolate(e, neg, T.name()));

surfaceVectorField U_pos("U_pos", rhoU_pos/rho_pos);
surfaceVectorField U_neg("U_neg", rhoU_neg/rho_neg);

surfaceScalarField p_pos("p_pos", rho_pos*rPsi_pos);
surfaceScalarField p_neg("p_neg", rho_neg*rPsi_neg);

surfaceScalarField phiv_pos("phiv_pos", U_pos & mesh.Sf());
// Note: extracted out the orientation so becomes unoriented
phiv_pos.setOriented(false);
surfaceScalarField phiv_neg("phiv_neg", U_neg & mesh.Sf());
phiv_neg.setOriented(false);
```

rhoCentralFoam/rhoCentralFoam.C

- At the beginning of the time loop, the code interpolates primitive variables $\Psi_{f\pm}$ (e.g., density, velocity, and energy) from cell centers to face centers in the positive (*pos*) and negative (*neg*) directions and the outward and inward fluxes at the face $\varphi_{f\pm}$.
- The interpolation is performed using the *interpolate* function delcared in the *rhoCentralFoam/directionInterpolate.H* header file.
- The interpolation is performed using the discretization scheme specified in the *fvSchemes* dictionary, for example :

```
interpolationSchemes
{
    default         linear;

    reconstruct(rho) vanLeer;
    reconstruct(U)  vanLeerV;
    reconstruct(T)  vanLeer;
}
```

fvSchemes

- Central flux scheme implementation in *rhoCentralFoam.C* :

```
volScalarField c("c", sqrt(thermo.Cp()/thermo.Cv()*rPsi));
surfaceScalarField cSf_pos
(
    "cSf_pos",
    interpolate(c, pos, T.name())*mesh.magSf()
);

surfaceScalarField cSf_neg
(
    "cSf_neg",
    interpolate(c, neg, T.name())*mesh.magSf()
);

surfaceScalarField ap
(
    "ap",
    max(max(phiv_pos + cSf_pos, phiv_neg + cSf_neg),
v_zero)
);

surfaceScalarField am
(
    "am",
    min(min(phiv_pos - cSf_pos, phiv_neg - cSf_neg), v_zero)
);
```

rhoCentralFoam/rhoCentralFoam.C

- The local speed of sound is first calculated with :

$$c = \sqrt{\frac{\gamma}{\psi}}$$

- Then interpolated in the positive and negative direction and multiplied by the surface area of the cell face to obtain the associated volumetric fluxes *cSf_pos* and *cSf_neg*.
- *ap* and *am* correspond to a_{f+} and a_{f-} defined previously as :

$$a_{f+} = max\left(\phi_{f+} + c_{f+}|S_f|,\ \phi_{f-} + c_{f-}|S_f|,\ 0\right)$$

$$a_{f-} = min\left(\phi_{f+} - c_{f+}|S_f|,\ \phi_{f-} - c_{f-}|S_f|,\ 0\right)$$

- Central flux scheme implementation in *rhoCentralFoam.C* :

```
surfaceScalarField a_pos("a_pos", ap/(ap - am));

surfaceScalarField amaxSf("amaxSf", max(mag(am), mag(ap)));

surfaceScalarField aSf("aSf", am*a_pos);

if (fluxScheme == "Tadmor")
{
    aSf = -0.5*amaxSf;
    a_pos = 0.5;
}

surfaceScalarField a_neg("a_neg", 1.0 - a_pos);

phiv_pos *= a_pos;
phiv_neg *= a_neg;

surfaceScalarField aphiv_pos("aphiv_pos", phiv_pos - aSf);
surfaceScalarField aphiv_neg("aphiv_neg", phiv_neg + aSf);

// Reuse amaxSf for the maximum positive and negative fluxes
// estimated by the central scheme
amaxSf = max(mag(aphiv_pos), mag(aphiv_neg));

#include "centralCourantNo.H"
```

$$\alpha = \begin{cases} \frac{1}{2}, & KT \\ \frac{a_{f+}}{a_{f+} - a_{f-}}, & KNP \end{cases}$$

rhoCentralFoam/rhoCentralFoam.C

- The weigting factors *a_pos* and *a_neg* for α and $1-\alpha$ are calculated.
- The diffusive volumetric flux *aSf* for ω_f is then computed.
- They are updated according to the chosen scheme (Kurganov for KNP, and Tadmor for KT).
- Finally *aphiv_pos* and *aphiv_neg* are calculated as:

$$\sum_f \phi_f \Psi_f = \sum_f \alpha \phi_{f+} \Psi_{f+} + (1-\alpha)\phi_{f-}\Psi_{f-} + \omega_f(\Psi_{f-} - \Psi_{f+})$$

$$\sum_f \phi_f \Psi_f = \sum_f (\alpha \phi_{f+} - \omega_f)\Psi_{f+} + ((1-\alpha)\phi_{f-} + \omega_f)\Psi_{f-}$$

- We can now compute the different convective terms of the governing equations $\nabla\cdot\boldsymbol{u}\Psi$, for example $\nabla\cdot(\boldsymbol{u}(\rho\boldsymbol{u}))$:

```
surfaceVectorField phiU(aphiv_pos*rhoU_pos + aphiv_neg*rhoU_neg)
```

- Solve the governing equations :

```
// --- Solve density
solve(fvm::ddt(rho) + fvc::div(phi));

// --- Solve momentum
solve(fvm::ddt(rhoU) + fvc::div(phiUp));

U.ref() =
    rhoU()
   /rho();
U.correctBoundaryConditions();
rhoU.boundaryFieldRef() ==
rho.boundaryField()*U.boundaryField();

if (!inviscid)
{
    solve
    (
        fvm::ddt(rho, U) - fvc::ddt(rho, U)
      - fvm::laplacian(muEff, U)
      - fvc::div(tauMC)
    );
    rhoU = rho*U;
}
```

rhoCentralFoam/rhoCentralFoam.C

- The continuity equation is solved explicitly to update the density.
- Momentum equation to compute the velocity field is solved in two steps :
  - The inviscid equation for *rhoU* is first solved explicitely without viscous contribution.
  - After updating the velocity field *U*, the diffusiveterms are applied as implicit corrections to the inviscid equation if the viscosity is non-zero.
- A similar procedure is applied for the energy equation.
- Finally the pressure is computed from the ideal gas law :

$$\rho = \frac{W}{RT} p = \psi p$$

```
p.ref() =
    rho()
   /psi();
p.correctBoundaryConditions();
rho.boundaryFieldRef() ==
psi.boundaryField()*p.boundaryField();
```

rhoCentralFoam/rhoCentralFoam.C

Implementation of *reactingRhoCentralFoam*

- *thermo* Object and species mass fraction:

```
Info<< "Reading thermophysical properties\n" << endl;
autoPtr<psiReactionThermo> pThermo(psiReactionThermo::New(mesh));
psiReactionThermo& thermo = pThermo();
volScalarField& e = thermo.he();

basicSpecieMixture& composition = thermo.composition();
PtrList<volScalarField>& Y = composition.Y();
PtrList<volScalarField> rhoY(Y.size());

const word inertSpecie(thermo.get<word>("inertSpecie"));
if (!composition.species().found(inertSpecie))
{
    FatalIOErrorIn(args.executable().c_str(), thermo)
        << "Inert specie " << inertSpecie << " not found in available species "
        << composition.species() << exit(FatalIOError);
}
```

reactingRhoCentralFoam/createFields.H

- First modification, we change the thermophysical model from *psiThermo* to *psiReactionThermo* to take into account chemical reactions and the species mass fractions fields.
- Identical as *reactingFoam*, with the exception of the new line *PtrList<volScalarField> rhoY(Y.size());* which declares the list of pointers to the density weighted species mass fractions

$$\hat{Y}_k = \rho Y_k$$

needed for the implementation of central convective fluxes in the species mass fractions equations.

- *thermo* Object and species mass fraction:

```
multivariateSurfaceInterpolationScheme<scalar>::fieldTable fields;

forAll(Y, i)
{
    volScalarField& Yi = Y[i];
    fields.add(Yi);

    const word Yiname = Yi.name();
    rhoY.set
    (
        i,
        new volScalarField
        (
            IOobject
            (
                "rho"+Yiname,
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            rho*Yi
        )
    );
}
fields.add(thermo.he());
```

reactingRhoCentralFoam/createFields.H

- The block *rhoY.set(i, new volScalarField(...))* creates a new field for the density-weighted species mass fraction *rhoY[i]* and assigns it to the *rhoY* field list.
- The value of *rhoY[i]* is computed as the product of the density field *rho* and the corresponding species mass fraction field *Yi*.
- Similarly as in *reactingFoam*, the variables *Yi* and *he* are added to the *fields* table for interpolation.
- The *IOobject::NO READ* and *IOobject::NO WRITE* flags ensure that the field is not read from or written to disk automatically.

reactingRhoCentralFoam/rhoYEqn.H

- Species transport equation:

```
// --- Solve species
#include "rhoYEqn.H"
```

```
forAll(Y, i)
{
    if (i != inertIndex && composition.active(i))
    {
        volScalarField& rhoYi = rhoY[i];
        volScalarField& Yi = Y[i];

        surfaceScalarField rhoYi_pos(interpolate(rhoYi, pos, "Yi"));
        surfaceScalarField rhoYi_neg(interpolate(rhoYi, neg, "Yi"));

        surfaceScalarField phiYi(aphiv_pos*rhoYi_pos + aphiv_neg*rhoYi_neg);

        // --- Solve Yi
        solve(fvm::ddt(rhoYi) + fvc::div(phiYi));
        rhoYi.clamp_min(0);
        Yi.ref() = rhoYi()/rho();
        Yi.correctBoundaryConditions();
        rhoYi.boundaryFieldRef() == rho.boundaryField()*Yi.boundaryField();

        ...
    }
}
```

reactingRhoCentralFoam/rhoYEqn.H, part 1

- We create a new header file *rhoYEqn.H*, that we include after solving the momemtum equation.
- Modified version of the *YEqn.H* of *reactingFoam* to include the central scheme discretization of the advective fluxes $\nabla\cdot(\boldsymbol{u}(\rho Y_k))$ using *aphiv_pos* and *aphiv_neg*.
- Inviscid transport equation is solved without the chemical source term. The species mass fraction *Yi* is updated and its boundary conditions corrected.

reactingRhoCentralFoam/rhoYEqn.H

- Species transport equation:

```
// --- Solve species
#include "rhoYEqn.H"
```

```
forAll(Y, i)
{
    if (i != inertIndex && composition.active(i))
    {
        ...

        if (!inviscid)
        {
            fvScalarMatrix YiEqn
            (
                fvm::ddt(rho, Yi) - fvc::ddt(rho, Yi)
              - fvm::laplacian(muEff, Yi)
                ==
                reaction->R(Yi)
            );
            YiEqn.solve("Yi");
        }

        ....

    }
}
```

reactingRhoCentralFoam/rhoYEqn.H, part 2

- We create a new header file *rhoYEqn.H*, that we include after solving the momemtum equation.
- Modified version of the YEqn.H of *reactingFoam* to include the central scheme discretization of the advective fluxes $\nabla\cdot(\boldsymbol{u}(\rho Y_k))$ using *aphiv_pos* and *aphiv_neg*.
- Inviscid transport equation is solved without the chemical source term. The species mass fraction *Yi* is updated and its boundary conditions corrected.
- The diffusive term is applied as implicit correction to the inviscid equation if the viscosity is non-zero.
- Like in reactingFoam, we add the species production rate *reaction->R(Yi)* to the transport equation in both inviscid and viscous formulations.

reactingRhoCentralFoam/rhoYEqn.H

- Species transport equation:

```
// --- Solve species
#include "rhoYEqn.H"
```

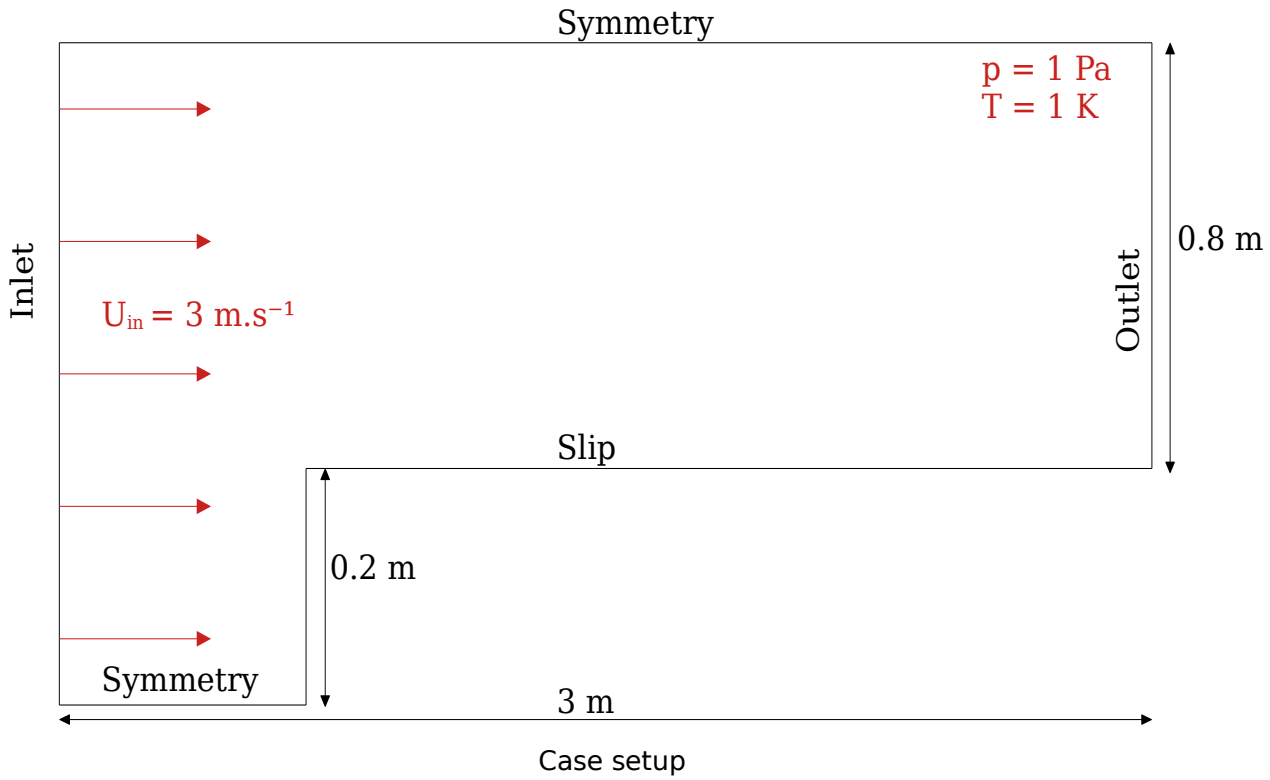
```
forAll(Y, i)
{
    if (i != inertIndex && composition.active(i))
    {
        ...

        else
        {
            fvScalarMatrix YiEqn
            (
                fvm::ddt(rho, Yi) - fvc::ddt(rho, Yi)
                ==
                reaction->R(Yi)
            );
            YiEqn.solve("Yi");
        }

        Yi.clamp_min(0);
        rhoYi = rho*Yi;
        Yt += Yi;
    }
}
```

reactingRhoCentralFoam/rhoYEqn.H, part 3

- We create a new header file *rhoYEqn.H*, that we include after solving the momemtum equation.
- Modified version of the YEqn.H of *reactingFoam* to include the central scheme discretization of the advective fluxes $\nabla\cdot(\boldsymbol{u}(\rho Y_k))$ using *aphiv_pos* and *aphiv_neg*.
- Inviscid transport equation is solved without the chemical source term. The species mass fraction *Yi* is updated and its boundary conditions corrected.
- The diffusive term is applied as implicit correction to the inviscid equation if the viscosity is non-zero.
- Like in reactingFoam, we add the species production rate *reaction->R(Yi)* to the transport equation in both inviscid and viscous formulations.
- The rest of the code is similar to *reactingFoam*, with the mass fractions minimum limited to zero and the inert species solved as *Y[inertIndex] = scalar(1) – Yt;*.

Test Cases and Results

- Non-reacting case, adapted from *$FOAM_TUTORIALS/compressible/rhoCentralFoam/forwardStep* :



- Uniform *Mach 3* flow in a wind tunnel containing a forward-facing step introduced as a test for numerical schemes.
- The properties are set such that this is an inviscid gas for which the speed of sound is *1 m/s* at a temperature of *1 K*.
- Structured uniform mesh with the cell length equal to *1.25 cm*, and we ran the simulation at a CFL number of 0.2 for a duration of *4 s*.
- This is a first check of the implementation without reacting species and check if it gives the same results as *rhoCentralFoam*.

- Non-reacting case, adapted from *$FOAM_TUTORIALS/compressible/rhoCentralFoam/forwardStep* :

```
thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       const;
    thermo          eConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

inertSpecie       N2;

chemistryReader     foamChemistryReader;

foamChemistryFile   "<constant>/reactions";

foamChemistryThermoFile "<constant>/thermo.compressibleGas";
```

thermophysicalProperties

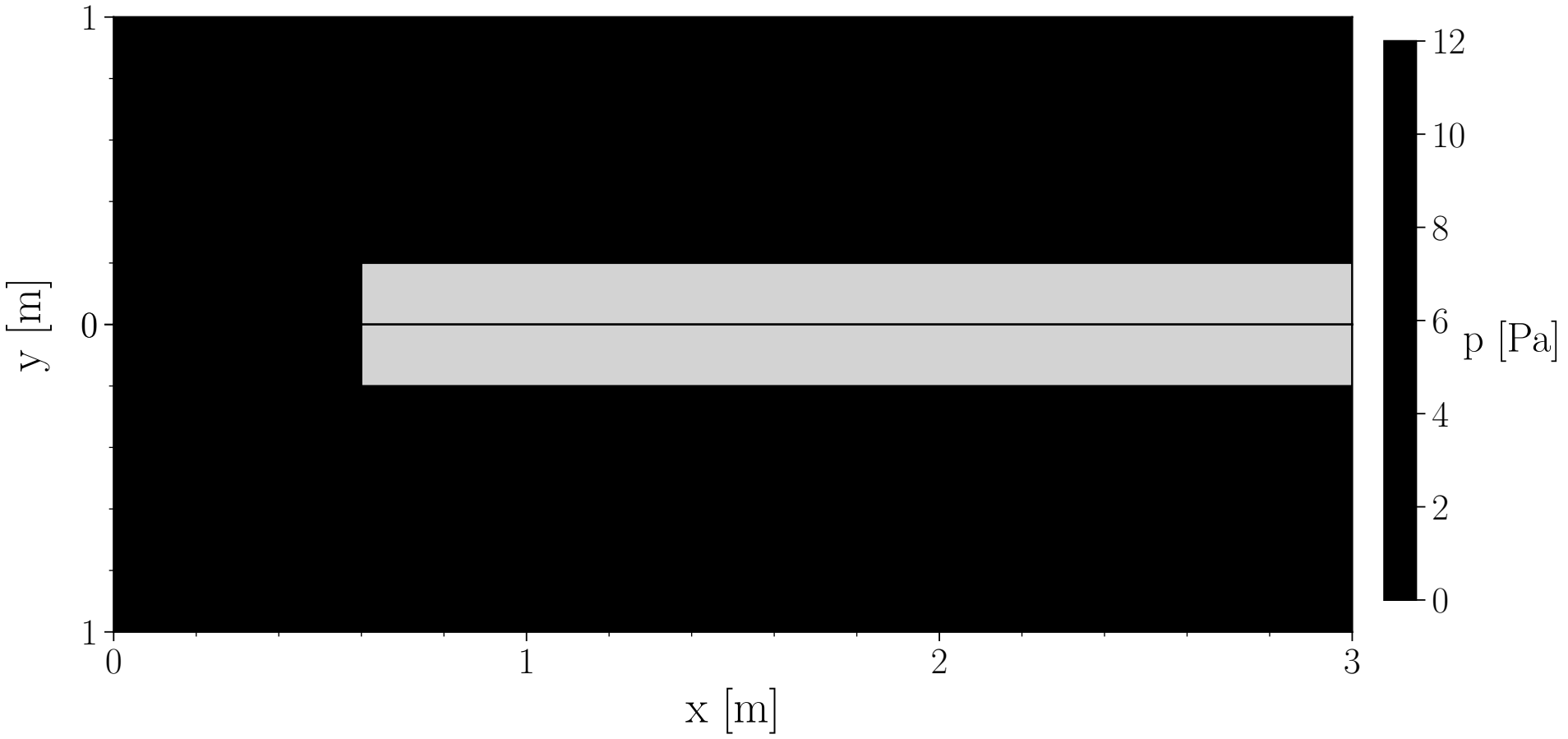
```
N2
{
    specie
    {
        molWeight       11640.3;
    }
    elements
    {
        N       2;
    }
    thermodynamics
    {
        Cv              1.7857;
        Hf              0;
    }
    transport
    {
        mu              0;
        Pr              1;
    }
}
```

thermo.compressibleGas

$$c = \sqrt{\frac{\gamma}{\psi}}, \quad \psi = \frac{1}{RT}$$

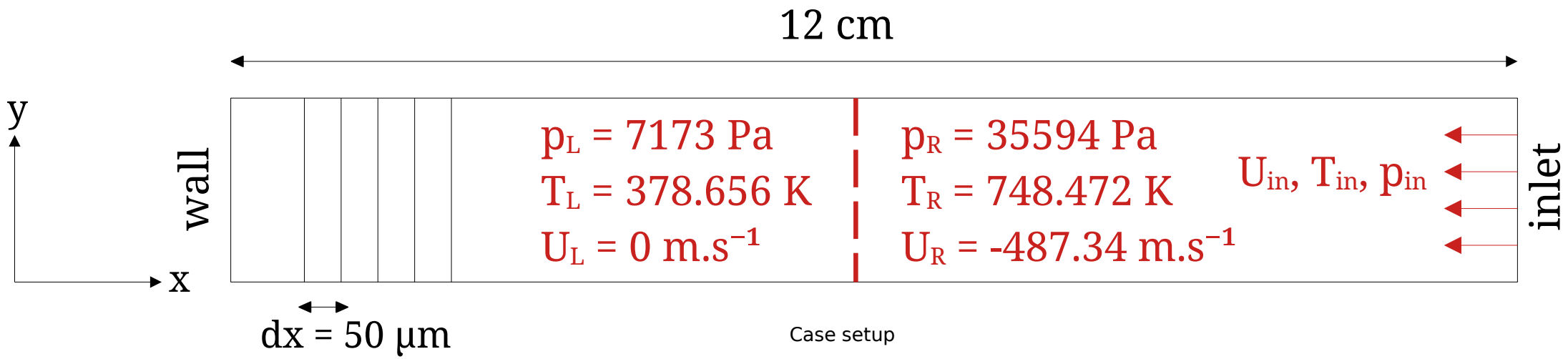
$$\gamma = \frac{C_p}{C_v}, \quad C_p = C_v + \frac{R}{W} \Rightarrow c = 1\,m/s$$

- Non-reacting case, adapted from *$FOAM_TUTORIALS/compressible/rhoCentralFoam/forwardStep* :



forwardStep case results comparison between *rhoCentralFoam* (Top) and mirrored *reactingRhoCentralFoam* (Bottom).

- Reacting case, One-Dimensional Reacting Shock Tube :



Case setup

- We validate the implementation of the transport equation of the reacting species.
- It involves an inviscid reactive mixture in a closed tube, where a shock reflects off a solid boundary, triggering a reaction wave that grows and merges with the shock structure.
- The mixture consists of a 2:1:7 molar ratio of H2 : O2 : Ar.
- The domain is discretized with 2400 uniform grid points and the convective CFL number is set to 0.1.

One-Dimensional Reacting Shock Tube

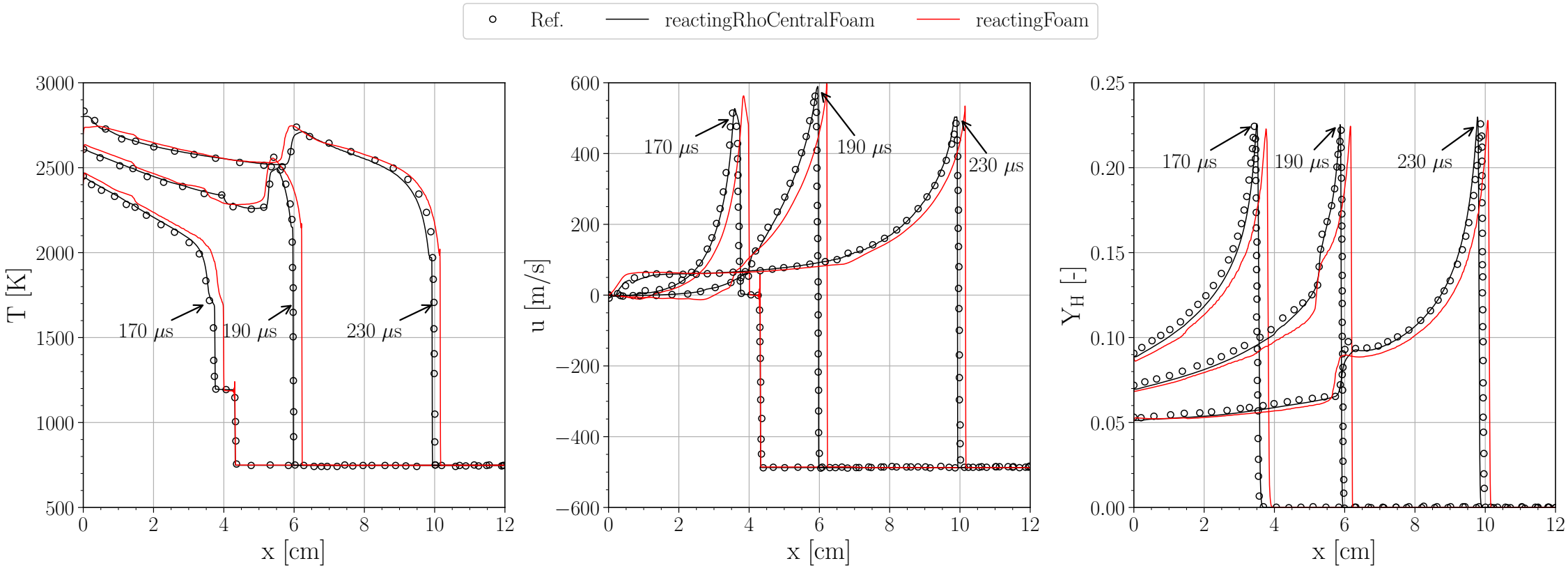
- Reacting case, One-Dimensional Reacting Shock Tube :

```
thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleInternalEnergy;
    equationOfState perfectGas;
    specie          specie;
}

inertSpecie     AR;

chemistryReader     foamChemistryReader;

foamChemistryFile   "<constant>/reactions";

foamChemistryThermoFile "<constant>/thermo";
```

thermophysicalProperties

- Unlike the previous non-reacting case, this simulation involves chemical reactions of combustion, requiring a proper chemistry mechanism for hydrogen combustion.
- We obtained the CHEMKIN files from Li et al. [3] and converted them using the *chemkinToFoam* utility to make them readable for the OpenFOAM-native chemistry reader.
- Those files are created as *reactions* and *thermo* in the *constant* folder.

- Reacting case, One-Dimensional Reacting Shock Tube :



- *reactingRhoCentralFoam* demonstrates superior agreement with the reference solution across all three fields.
- *reactingFoam* schemes introduce higher numerical diffusion, which smears gradients and limits its ability to accurately resolve shocks.
- More outer and inner loops improved accuracy but increased computational time.

Conclusion and Future Work

Conclusion and Future Work

- Developed *reactingRhoCentralFoam*, an extension of the density-based solver *rhoCentralFoam* to include reacting flow capabilities.
- Validated the solver against well-established test cases, including both non-reacting and reacting flows, demonstrating excellent agreement with reference solutions.
- Outperformed *reactingFoam* in the supersonic reacting case by accurately capturing sharp gradients and shocks.
- Provides a robust tool for simulating high-speed compressible flows with chemical reactions.
- Future Work:
  - Optimize the solver for low-speed flows.
  - Explicit nature facilitates the implementation of advanced spatial and temporal schemes.
  - Integrate more complex transport and turbulence-chemistry interaction models for more precise results.

Thank You!

References

References

References

- [1]: Nemiroff, R., & Bonnell, J. (1995). Astronomy picture of the day. Exploration of the Universe Division (EUD), Goddard Space Flight Center (GSFC), NASA.

- [2]: A model of a Mercury capsule captured by high-speed cameras, showing the forward pressure shockwave and the wake. (Image is taken from NASA's web site: http://www.nasa.gov.)

- A. Kurganov, S. Noelle, and G. Petrova, “Semidiscrete central-upwind schemes for hyperbolic conservation laws and hamilton–jacobi equations,” SIAM Journal on Scientific Computing, vol. 23, no. 3, pp. 707–740, 2001.

- [4]:J. Li, Z. Zhao, A. Kazakov, and F. L. Dryer, “An updated comprehensive kinetic model of hydrogen combustion,” International journal of chemical kinetics, vol. 36, no. 10, pp. 566–575, 2004.