Technische Universität Berlin
Fachbgebiet Verbrennungskinetik
Institut für Strömungsmechanik und Technische Akustik

1D ZND detonation model

MATLAB® script suite to calculate kinetic properties and approximate cell sizes

Documentation v1.02

Niclas Hanraths

1 Introduction

The characteristic cell size is an important property to indicate the detonation sensitivity and stability of a given mixture at specific initial conditions. However, due to the highly multidimensional nature of real detonations, correctly determining cell sizes requires either experimental measurements or very complex CFD calculations.

An often made simplification for Chapman-Jouguet detonations is the one-dimensional ZND model which, after iteratively solving the equations for detonation velocity and CJ state properties, can provide a characteristic induction zone length based on the chemical kinetics of the reaction. Subsequently, this induction zone length—in combination with other chemical sensitivity factors—can be used as an input for semi-empirical correlations to estimate the detonation cell size.

The MATLAB® function scripts described within this documentation aim to provide a consistent way of calculating the characteristic induction zone length according to given initial conditions. Applicable formulas, assumptions and algorithms are taken from various sources and combined into modular functions, each of them further described during section 2. The results can then be used to approximate the characteristic cell size using the semi-empirical correlations suggested by NG ET AL. [6], also explained in more detail in subsection 2.6.

2 Functions and subfunctions

All functions and subroutines necessary to approximate the detonation cell size of a certain combustable mixture are explained in detail in the next subsections. It must be stressed that almost all require the open source chemical kinetics software suite *Cantera* to calculate mixture properties and equilibrium states. They were developed and tested using the MATLAB® module of *Cantera 2.1.2* in combination with *Python 3.4.2* and *Numpy 1.9.1*.

An overview of the structure of the entire calculation process is given in Figure 1. Input and output variables for individual functions are also depicted, symbolized by blue and

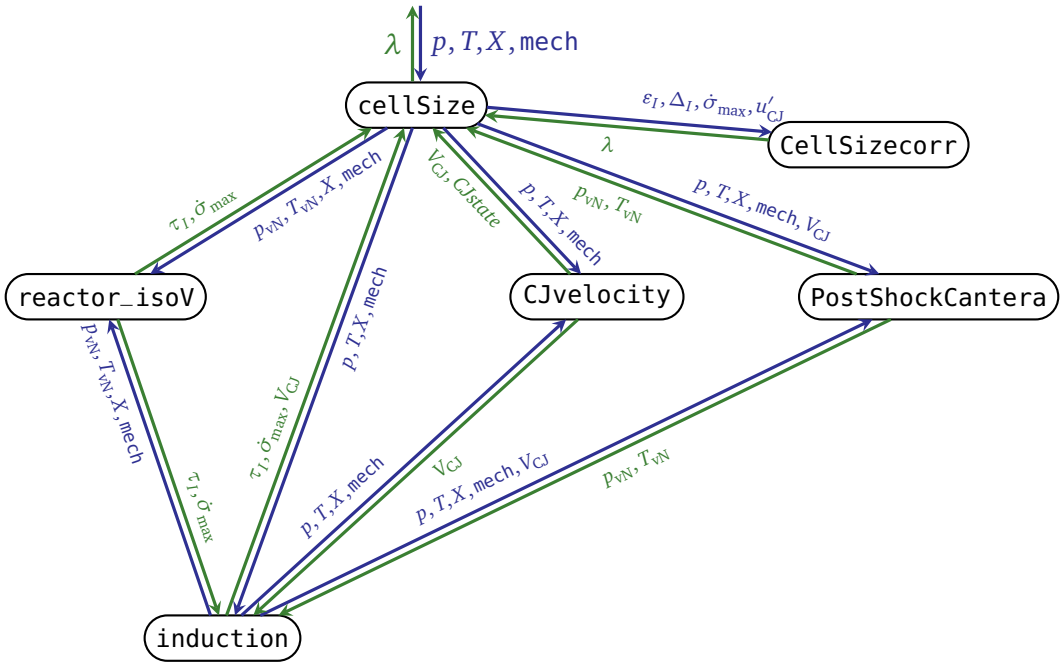

Figure 1: Scheme of function calls, input and output variables for detonation cell size calculation

green arrows, respectively.

Because of the modular approach to the challenge of computing various interdepending parameters, each function and subfunction solves a distinct purpose and can be used as a stand-alone calculation tool. For this reason, a major function like `cellSize` calls another major function, `induction`, to produce the induction time, but is also required to call various subfunctions of `induction` to obtain intermediate results which are not part of its final output.

2.1 `cellSize`

To approximate the characteristic cell size, the `cellSize` function acts as the main function which takes the initial state of the mixture and the chosen kinetic mechanism `mech` as input parameters.

```
function [ lambda ] = cellSize( P, T, X, mech )
```

The initial state must consist of the static pressure p in Pa, the temperature T in K and X, which is either a column vector of mole fractions or a string listing all components according to *Cantera* standards. This function calls all other functions and subfunctions to gather the required quantities needed by the approximation model that yields as a final result the approximated detonation length λ in m.

The declaration is followed by:

```
try
    gas = importPhase([mech '.xml']);
catch
    try
        gas = importPhase([mech '.cti']);
    catch
        error('Selected mechanism not found!');
    end
end
```

This code sections can be found in all functions that need to invoke a *Cantera* phase for chemical calculations. Variable `mech` is used to search for a corresponding mechanism in *Cantera's* database. *Cantera* converts its more verbose .cti files to .xml files via *Python* before using them, therefore already converted mechanism files are preferred. For this reason, `mech` may only consist of the name of the mechanism. For example, `mech = 'gri30'` is allowed, while `mech = 'gri30.xml'` would be searching for `'gri30.xml.xml'` and result in an error.

Even though in principle all *Cantera*-compatible kinetic mechanisms can be used, the latest recommendation for hydrogen-fueled detonations would be the mechanism developed by BURKE ET AL. [1], for its explicit consideration of high-pressure combustion.

The following code section calls the functions which calculate detonation properties of the given mixture:

```
%% Get required values from induction time
     calculation
[ tau, sigma_max, shockSpeed ] = induction( P, T,
      X, mech );
[P_vN, T_vN] = PostShockCantera(P, T, X, mech,
     shockSpeed);
```

The `induction` function receives the exact same input values as `cellSize`. It starts the function chain to obtain the characteristic induction time τ_I and is further described in subsection 2.2. The next subfunction to be called by `cellSize` is `PostShockCantera`, fully explained in subsection 2.4. It solves the Hugoniot- and Rayleigh-equations* to obtain the von-Neumann conditions that are present directly after the shock front.

The next step is the calculation of the non-dimensional global activation energy ε_I, which is another stability parameter required by the approximation model for the characteristic cell size. Its use has been suggested by NG ET AL. [5, 6]. To estimate its value, the method introduced by SCHULTZ AND SHEPHERD [7, p. 81] is applied:

```
%% Calculate non dimensional global activation
     energy epsilon [Schultz, Shepherd, 2000]
shockSpeed_plus  = 1.01 * shockSpeed;
shockSpeed_minus = 0.99 * shockSpeed;
set(gas,'T',T,'P',P,'X',X);
[P_vN_plus , T_vN_plus ] = PostShockCantera(P, T,
      X, mech, shockSpeed_plus);
set(gas,'T',T,'P',P,'X',X);
[P_vN_minus, T_vN_minus] = PostShockCantera(P, T,
      X, mech, shockSpeed_minus);

tau_plus  = reactor_isoV(P_vN_plus,  T_vN_plus,
     X, mech);
tau_minus = reactor_isoV(P_vN_minus, T_vN_minus,
     X, mech);

epsilon = T_vN^-1 * log(tau_plus/tau_minus) / (
     T_vN_plus^-1 - T_vN_minus^-1);
```

The principle is to assume a global Arrhenius law in which the induction time depends on a global activation energy for the combustion reaction and the von-Neumann temperature of the mixture after the compression shock wave. In combination with the shock jump conditions and the assumption of a strong shock wave, inverting this relation leads to the bottom equation of the code section

$$\varepsilon_\mathrm{I} = \frac{E_\mathrm{I}}{RT_\mathrm{vN}} = \frac{1}{T_\mathrm{vN}} \left(\frac{\ln \tau_\mathrm{I}^+ - \ln \tau_\mathrm{I}^-}{\frac{1}{T_\mathrm{vN}^+} - \frac{1}{T_\mathrm{vN}^-}} \right),$$

where the superscripts + and − mark those induction time lengths and von-Neumann temperatures which result from calculations with detonation velocities altered by ±1%. These values are therefore computed in advance by use of the already mentioned `PostShockCantera` subfunction, as well as the `reactor_isoV` subfunction. The latter can be found in subsection 2.5 and solves

*see [4]

the equations of state in an ideal, zero-dimensional and isochoric reactor.

The next section transforms the induction time into the required induction length:

```
%% Calculate induction length
set(gas,'T',T,'P',P,'X',X);
rho0 = density(gas);
set(gas,'P',P_vN, 'T', T_vN, 'X', X);
rho1 = density(gas);
u_vN = shockSpeed * rho0 / rho1;
delta = u_vN * tau;
```

For this, the induction time is multiplied with the von-Neumann particle velocity in a shock attached reference frame. This velocity is also obtained by a shock jump condition, which relates the change in velocity across the shock wave to the change in density.

The last required input parameter for the model by NG ET AL. [6] is the Chapman-Jouguet particle velocity in a shock-attached frame, u'_{CJ}.

```
%% Calculate CJ-plane particle velocity
[ ~, CJstate ]  = CJvelocity(P,T,X,mech);
u_CJ = CJstate.uCJ;
```

It can be extracted from the CJ-state information that is part of the output of the `CJvelocity` subfunction, looked into in subsection 2.3.

Finally, the the approximation polynom itself is called via the `cellSizeCorr` subfunction (see subsection 2.6) and yields the desired characteristic length:

```
%% Approximate cell size through polynom [Dick,
     Ju, Lee, 2007]
lambda = cellSizeCorr(epsilon, delta, sigma_max,
     u_CJ);
```

2.2 induction

The `induction` function serves as a standalone calculation tool to obtain the characteristic induction time τ_I of a Chapman-Jouguet detonation. Its input requirements are the same as for the `cellSize` function.

```
function [ tau, sigma_max, shockSpeed ] =
     induction( P, T, X, mech )
```

The first code block computes and displays the detonation velocity V_{CJ} via `CJvelocity`:

```
%% get shockSpeed via CJ calculation
set(gas,'T',T,'P',P,'X',X);
X = moleFractions(gas);
shockSpeed = CJvelocity(P,T,X,mech);
disp(['Shock speed is ' num2str(shockSpeed) ' m/s
     ']);
```

Following this, like in `cellSize`, the von-Neumann state is provided by `PostShockCantera`:

```
%% get von Neumann state parameters
set(gas,'T',T,'P',P,'X',X);
[P_vN, T_vN] = PostShockCantera(P, T, X, mech,
     shockSpeed);
```

The last step is the use of the ideal isochoric reactor model in `reactor_isoV`, taking the von-Neumann state as input parameters:

```
%% solve ode for reaction
[tau , sigma_max] = reactor_isoV(P_vN, T_vN, X,
     mech);
```

In this case, not only τ_I is calculated, but also the maximum heat release rate $\dot{\sigma}_{max}$. The latter has has a dimension of $1/s$, so its inverse value is used in the model of NG ET AL. [6] to express the characteristic time scale of the heat release, i.e. the reaction time, which is another required stability parameter.

2.3 CJvelocity

The `CJvelocity` function iteratively solves the one-dimensional continuity, momentum and energy equations for a CJ detonation at given initial pressure and temperature. Output parameters are the CJ velocity and the `CJstate` structure consisting of the values of pressure, temperature, mole fractions and

particle velocity at the CJ plane of the detonation (which, per definition, is located where the local Mach number is $Ma = 1$).

```
function [ V_CJ, CJstate ]  = CJvelocity(P1,T1,X1
    ,mech)
```

The following code section is excerpted almost unaltered (apart from certain syntax difference between Fortran95 and MATLAB® code) from the NASA *Chemical Equilibrium with Applications* toolbox (*CEA*, [2]), which is freely available. Its mathematical foundations, which can be inspected in the *CEA* manual, are based on the solution of the CJ detonation state relations by a Newton-Raphson iteration, as proposed by ZELEZNIK [9].

Chemical equilibrium is calculated for assumed values of p_{CJ} and T_{CJ}. Results are then used to solve the momentum and energy relations across a CJ detonation, using the residuals as an adjustment parameter and convergence criteria for the original estimates.

The first challenge is to find suitable values for an initial guess of p_{CJ} and T_{CJ}. This is done in two steps:

```
set(gas,'P',P1,'T',T1,'X',X1);

M1 = meanMolecularWeight(gas);
h1 = enthalpy_mass(gas);

P = 15 * P1;

% Acquire initial guess for T2 by finding flame
    temperature according to
% chapter 8.3 of CEA documentation
h_init = h1 + .75*gasconstant*T1/M1 * P/P1;
setState_HP(gas, [h_init P]);
equilibrate(gas,'HP');
T_init = temperature(gas);
T = T_init;
```

While p_{CJ} is approximated as $\approx 15\,p_0$, the first guess for T_{CJ} is calculated by finding the adiabatic flame temperature for a fixed equilibrium enthalpy. This assumption is based on the relatively small range of possible solutions for the pressure ratio in a detonation as is argued by ZELEZNIK [9].

Additional refinement of the initial conditions is undertaken in the following loop:

```
% Further improving initial guesses for T and P
    by recursion formulas
for i=1:3

    set(gas, 'T',T, 'P',P, 'X',X1);
    equilibrate(gas,'TP');
    M2 = meanMolecularWeight(gas);
    alpha = T1/T * M2/M1;
    [gamma_s , cp_eq] = partDeriv( T, P, X1, gas
        );
    P = P1 * (1+gamma_s)/(2*gamma_s*alpha) * ...
        (1+(1-(4*gamma_s*alpha)/(1+gamma_s)^2)
            ^0.5);
    r = alpha * P/P1;
    T = T1 * T_init/T1 - 0.75 * gasconstant/(M1*
        cp_eq) * 15 ...
        + (gasconstant*gamma_s)/(2*M1*cp_eq) * (r
            ^2-1)/r * P/P1;
end
```

Here, the initial values are further improved using a method of successive substitutions*. These recursion formulas are used three times, as in the original *CEA* tool. Over the course of this, the `partDeriv` subfunction, which is included at the end of `CJvelocity`, is used for the first time to calculate partial derivatives necessary to obtain the correct values for the equilibrium isentropic exponent γ_s and equilibrium isobaric heat capacity $c_{p,\mathrm{eq.}}$. The function is further discussed at the end of this subsection.

The next section of code holds the main iteration loop:

```
% Main iteration loop
I=0;
while 1
    I = I+1;
    set(gas, 'T',T, 'P',P, 'X',X1);
    equilibrate(gas,'TP');
    M2 = meanMolecularWeight(gas);
    [ gamma_s , cp_eq , Dlvtp, Dlvpt ] =
        partDeriv( T, P, X1,gas );
    alpha = T1/T * M2/M1;
```

*see [9]

```
    r = alpha * P/P1;
    a11 = P1/P + gamma_s*r*Dlvpt;
    a12 = gamma_s*r*Dlvtp;
    a21 = 0.5*gamma_s*(r^2 - 1 - Dlvpt*(1+r^2)) +
          Dlvtp - 1;
    a22 = -0.5*gamma_s*Dlvtp*(r^2+1) - M2*cp_eq/
         gasconstant;
    b1 = P1/P - 1 + gamma_s*(r-1);
    b2 = M2*(enthalpy_mass(gas)-h1)/(gasconstant*
         T) - 0.5*gamma_s*(r^2-1);
    d = a11*a22 - a12*a21;
    x1 = (a22*b1-a12*b2)/d;
    x2 = (a11*b2-a21*b1)/d;

    delta = 1;
    temp = max([abs(x1) abs(x2)]);
    if temp > 0.4054652
        delta = 0.4054652/temp;
    end
    P = P*exp(x1*delta);
    T = T*exp(x2*delta);
    soundspeed_equil = sqrt(gasconstant*gamma_s*T
         /M2);
    V_CJ = r*soundspeed_equil;

    % Convergence test
    if I > 8
        display('Could not converge in 8
             iterations')
        break
    elseif I <= 8 && temp < 0.5e-4
        break
    end
end
CJstate = struct('P',P,'T',T,'mole_fractions',
     moleFractions(gas), ...
    'uCJ', soundspeed_equil);
```

As before, a gas state is introduced and brought to equilibrium according to the current values of p_{CJ} and T_{CJ}. Mean molar mass and partial derivatives of this gas state can be calculated using the aforementioned `partDeriv` function. Results are then used in the iteration equations*. Residuals `x1` and `x2`, corresponding to p_{CJ} and T_{CJ}, respectively, are taken into account for the correction of the initial guess. Since the equations are solved for $\ln\left(\frac{p_{\mathrm{CJ}}}{p_0}\right)$ and $\ln\left(\frac{T_{\mathrm{CJ}}}{T_0}\right)$, these corrections must be done via an exponential function. The `delta` parameter that is additionally applied is supposed to suppress divergent behavior of the solution.

If the solution does not converge within 8 iterations, the iteration loop is aborted. Otherwise, V_{CJ} is calculated from u'_{CJ} through the mass equation and other CJ properties are saved in `CJstate`.

The last part of `CJvelocity` is the already mentioned `partDeriv` subfunction. Its output variables are the equilibrium isentropic exponent γ_s, the equilibrium isobaric heat ratio $c_{p,\mathrm{eq.}}$ and the partial derivatives $\left(\frac{\partial \ln v}{\partial \ln p}\right)_T$ and $\left(\frac{\partial \ln v}{\partial \ln T}\right)_p$.

```
function [ gamma_s , cp_eq, Dlvtp, Dlvpt ] =
     partDeriv( T, P, X, gas )
deltaT = T*1e-6;
deltaP = P*1e-8;

v_P = zeros(1,2);
v_T = zeros(1,2);
h = zeros(1,2);

for i = 1:2

    sign = [-1 1];

    set(gas,'P',P+sign(i)*deltaP,'T',T,'X',X);
    equilibrate(gas,'TP');
    v_P(i) = density(gas)^-1;

    set(gas,'P',P,'T',T+sign(i)*deltaT,'X',X);
    equilibrate(gas,'TP');
    v_T(i) = density(gas)^-1;
    h(i) = enthalpy_mass(gas);
end

Dlvtp = log(v_T(2)/v_T(1))/log((T+deltaT)/(T-
     deltaT));
Dlvpt = log(v_P(2)/v_P(1))/log((P+deltaP)/(P-
     deltaP));
cp_eq = (h(2)-h(1))/(2*deltaT);

set(gas,'T',T, 'P',P, 'X',X);
equilibrate(gas,'TP');

gamma_s = -(Dlvpt + ...
    gasconstant * Dlvtp^2 / (cp_eq *
         meanMolecularWeight(gas)))^-1;
```

This part is not included in the original *CEA* code since it uses its own chemical equilibrium solver, and the *Cantera* software used

*Which make use of several abbreviations, see again the *CEA* manual or [9] for a detailed explanations

in this work does not provide an interface to calculate partial derivatives on its own.

Another problem arises from the fact that *Cantera* only offers *frozen* (i.e., fixed mixture composition) values for thermodynamic properties like c_p, resulting from the weighted sum of the properties of each component of a mixture. But according to WOOD AND SALSBURG [8], equilibrium values for speed of sound and for other thermodynamic properties should be used.

The equilibrium specific heat ratio $c_{p,\text{eq.}}$ is therefore approximated as

$$c_{p,\text{eq.}} = \frac{h_{\text{eq.}}(T_0 + \Delta T) - h_{\text{eq.}}(T_0 - \Delta T)}{2\Delta T},$$

using a central difference scheme with the equilibrium enthalpy and a step width of $\Delta T = 1 \cdot 10^{-6}$ K.

The isentropic exponent γ_s is another important parameter. Coming from the definition of the speed of sound as it is made by ZELEZNIK [9], its most general definiton is

$$\gamma_s = \left(\frac{\partial \ln p}{\partial \ln \rho}\right)_s.$$

GORDON AND MCBRIDE [2] use the Bridgman tables to rearrange this to achieve the expression

$$\gamma_s = -\frac{c_{p,\text{eq.}}}{c_{v,\text{eq.}}} \left(\frac{\partial \ln v}{\partial \ln p}\right)_T^{-1}.$$

From the ideal gas law, $\left(\frac{\partial \ln v}{\partial \ln p}\right)_T$ is equal to $-\left(\frac{\partial \ln M_{\text{mean}}}{\partial \ln p}\right)_T - 1$. As can be seen easily, if the last partial derivative is zero, we will obtain the more familiar equation $\gamma_s = \frac{c_p}{c_v}$. However, for a fast reacting mixture in general and a highly pressure dependent detonation combustion especially, the mean molar mass of the components *will* change with respect to pressure perturbations, and it is therefore necessary to use the more general expression above*.

For that reason, the partial derivatives $\left(\frac{\partial \ln v}{\partial \ln p}\right)_T$ and $\left(\frac{\partial \ln v}{\partial \ln T}\right)_p$ are approximated using a central difference scheme for a step width of $\Delta p = 1 \cdot 10^{-8}$ Pa and $\Delta T = 1 \cdot 10^{-6}$ K respectively.†

2.4 PostShockCantera

The purpose of the `PostShockCantera` function is to calculate the post-shock (von-Neumann) state of p and T from an already solved CJ relation.

```
function [ P_vN, T_vN ] = PostShockCantera(P_0,
    T_0, X, mech, shockSpeed)
```

It makes use of the internal MATLAB® function `fzero` to solve a functional composed of the Hugoniot- and Rayleigh-equation for the density ρ.

```
set(gas,'P',P_0,'T',T_0,'X',X);
RHO_0 = density(gas);
U_0 = intEnergy_mass(gas);

RHO_init = 15; % choose wisely, see above!
RHO = fzero(@(RHO) hugoniotEnergyFunctional(gas,
    RHO_0, P_0, U_0, shockSpeed, RHO), RHO_init,
     optimset('Display', 'off', 'TolFun', eps, '
    TolX', eps));

rayleighPressure = P_0 - (shockSpeed*RHO_0)^2 *
    (1/RHO - 1/RHO_0);
energy = (rayleighPressure + P_0) * (1/RHO_0 - 1/
    RHO) / 2 + U_0;

setState_UV(gas,[energy 1/RHO]);
P_vN = pressure(gas);
T_vN = temperature(gas);
```

*The impact of this measure is generally small, but can be important under certain circumstances (especially for low pressures), as is shown in the original article by ZELEZNIK [9, Table 7]

†The second one can be used to replace c_v, after some more rearrangements according to the Bridgman tables, see [2]

The so obtained ρ is taken to solve the Rayleigh equations for pressure and energy, which then can be used by *Cantera* to calculate the corresponding von-Neumann state.

The last code section consists of a function for the aforementioned functional. It calculates the intersection of the Rayleigh- and Hugoniot-equations by iteration.

```
function [ err ] = hugoniotEnergyFunctional(gas,
    RHO_0, P_0, U_0, shockSpeed, RHO)
       % RHS for the Post-Shock state
            calculation
       %
       % Tries to find the density for which the
            Rayleigh line and the Hugoniot
       % intersect in the P/V diagram.
            Calculates the pressure for both
            functions
       % and returns their difference.

       % Calculate pressure using the Rayleigh
            line
       rayleighPressure = P_0 - (shockSpeed*
            RHO_0)^2 * (1/RHO - 1/RHO_0);

       % Calculate energy using the Hugoniot
       energy = (rayleighPressure + P_0) * (1/
            RHO_0 - 1/RHO) / 2 + U_0;

       % Determine the pressure according to
            this energy
       setState_UV(gas,[energy 1/RHO]);
   P_test = pressure(gas);
```

An initial guess of ρ is made, and the difference of the resulting values for p by each corresponding equation is taken as the error to be minimized.

Care must be taken when choosing the initial guess for ρ. Experience has shown that the minimization of the functional can result in convergence for values of ρ which are very close or equal to the initial value of ρ_0. Since we expect a drastic increase in density for the post-shock state in most cases, these incorrect results can usually be identified easily. A solution is to further increase either the value for the initial guess of ρ (which, for this reason, is already 15 by default) or the penalty for convergence values close to the initial value, as described in the comment at the end of the function. However, in some cases it might be also necessary to reduce the default value. This is mostly the case for very weak shocks, resulting for example from very rich or diluted mixtures.

2.5 reactor_isoV

The function `reactor_isoV` solves the ordinary differential equations for the chemical kinetics (`mech`) of a specific mixture (X) in a specific initial state (p, T), assuming a non-dimensional constant volume combustion inside the reaction zone. Output values are the induction time τ_I, i.e., time that passes until autoignition of the mixture, and the maximum heat release rate $\dot{\sigma}_{max}$, which is also called the maximum thermicity.

```
function [ tau, sigma_max ] = reactor_isoV( P, T,
      X, mech )
```

According to LEE [4, p. 195], the error resulting from the simplified constant-volume model can be neglected for an assumed simple linear dependence between the ZND detonation zone length Δ_I and the detonation cell size λ.

```
set(gas,'P',P,'T',T,'X',X);

% initial values
Y0=massFractions(gas);
T0=temperature(gas);
rho=density(gas);
sigma0 = 0;

% ode solver options
options=odeset('RelTol',1.e-6,'AbsTol',1e-12);

% set initial and final time
t0=0; tf=1e-5; % you may need to increase tf
```

The first lines only derive initial values for the temperature, thermicity and the mass frac-

tion of each species from the submitted initial state of the mixture. After this, options for the later used MATLAB® function `ode15s` regarding tolerance values for convergence are set up. The time scale in which the model is run must be adjusted to the needs of the specific mixture. This mostly depends on the fuel that is used. A run time which is too short might push the point of ignition out of scope, while an unnecessary long run time wastes processing time.

The actual solving of the ODE-system is accomplished in the next lines:

```
%% solve ode system
[t,z]=ode15s(@isoVsigma_ode, [t0 tf], [T0; Y0;
     sigma0], options, gas, rho);


sigma_max = max(gradient(z(:,end),t));
```

The `ode15s` function solves the ordinary differential equations for an isochoric reactor, which are are defined at the end of the file in a separate function named `isoVsigma_ode`. The results are two arrays, containing the time steps and the values of the solved variables (i.e., T, X and $\dot{\sigma}$). $\dot{\sigma}_{\max}$ is simply the maximum of all values of $\dot{\sigma}^*$.

The determination of τ_I is done according to the following definition:

```
%% determine ignition delay from maximum
     temperature gradient
[~,imaxgradT]=max(gradient(z(:,1),t)); % max grad
     T
tau = t(imaxgradT);
disp(['ignition delay: ' num2str(tau*1e6) '
     microseconds'])
```

This means that the point of ignition is defined as the moment of steepest ascent in temperature. Other definitions are possible and also common; adjustments can easily be made.

The following function `isoVsigma_ode` consists of the right hand sides of the ordinary differential equations for a constant-volume combustion reactor.

```
function dzdt = isoVsigma_ode(t,z,g,rho)
% RHS for solving constant volume reactor
T=z(1); % temperature
y=z(2:end-1); % mass fractions

set(g,'T',T,'Density',rho,'Y',y); %update gas

Ms  = molecularWeights(g);
dy = ydot(g); % = netProdRates(g).*Ms/rho; change
      in mass fraction
ui  = (enthalpies_RT(g) - 1) *gasconstant * T ./
     Ms; % specific internal energies
cv  = cv_mass(g); % specific heat

dT  = - sum(ui .* dy)/cv; % change in temperature

dsigma = sum ((meanMolecularWeight(g)./ Ms -
     enthalpies_RT(g) * gasconstant /  cp_mole(g)
     ) .* dy);


dzdt=[dT;dy;dsigma]; % return
```

The change of species concentration is calculated by solving the reaction kinetics through *Cantera*. Temperature changes can be directly related to the increase or decrease of the specific internal energy. The definition of thermicity for an ideal gas

$$\dot{\sigma} = \sum_{i=1}^{N_s} \left(\frac{M_\mathrm{mean}}{M_i} - \frac{h_i}{c_p T} \right) \frac{\mathrm{d}y_i}{\mathrm{d}t}$$

is derived by KAO AND SHEPHERD [3].

2.6 `cellSizeCorr`

This function only includes the suggested polynomial factors to relate Δ_I to λ by the model of NG ET AL. [6].

```
function [ lambda ] = cellSizeCorr( epsilon,
     delta, sigma, ucj )
```

*The solver for for the ordinary differential equation per default integrates the solutions over time. Therefore, to obtain the actual thermicity, which is a time rate of dimension s^{-1}, the gradient of the solution has to be taken.

```
A0 = 30.465860763763;
a1 = 89.55438805808153;
a2 = -130.792822369483;
a3 = 42.02450507117405;
b1 = -0.02929128383850;
b2 = 1.026325073064710e-5;
b3 = -1.031921244571857e-9;

zeta = epsilon .* delta .* sigma ./ ucj;

A = A0 + a3./zeta.^3 + a2./zeta.^2 + a1./zeta +
     b1.*zeta + b2.*zeta.^2 + b3.*zeta.^3;
lambda = A .* delta;

end
```

The aforementioned stability parameters ε_I and $\dot{\sigma}_\mathrm{max}$, as well as the particle velocity u'_CJ, are also applied.

3 Prospects

Results of the described model have been compared to various available experimental data for detonation cell sizes of H_2-O_2 and H_2-Air mixtures. Examples of these comparisons can be seen in Figure 2 and Figure 3. For the examined ranges of initial parameters, an overall good agreement is observed.

However, experimental data for higher pressures is sparse, while divergence between experimental results for lower pressures is high. An increasing influence of the choice of the applied kinetic mechanism can be recognized for very lean and very rich mixtures.

Although accuracy within a magnitude of $\approx 1\,\mathrm{mm}$ seems to be possible for H_2-O_2 mixtures, H_2-Air mixtures already only seem to provide an accuracy within a magnitude of $\approx 10\,\mathrm{mm}$, which can prove to be too much for stability considerations.

Moreover, the assumption of a constant volume reactor to calculate the induction time and thermicity is a simplification of the actual detonation reactions. Comparisons of the CJ temperatures calculated by the CEA code and by the constant volume reactor show a discrepancy with a magnitude of several 100 K. As has been stated in subsection 2.5, differences in results for λ can be mostly neglected for a simple linear dependence between detonation zone length and cell size. But since the polynomial parameters of the model also depend on ε_I, Δ_I and $\dot{\sigma}_\mathrm{max}$, which themselves depend on induction time and heat release, strictly speaking, this assumption is not valid. Therefore, the use of a genuine ZND reactor model should be considered. An example for such a reactor is provided by KAO AND SHEPHERD [3] as part of the shock-detonation-toolbox.

References

[1] Michael P. Burke, Marcos Chaos, Yiguang Ju, Frederick L. Dryer, and Stephen J. Klippenstein. *Comprehensive H2/O2 Kinetic Model for High-Pressure Combustion*. Department of Mechanical and Aerospace Engineering, July 2011.

[2] Sanford Gordon and Bonnie J. McBride. *Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications*. NASA Glenn Research Center, `http://www.grc.nasa.gov/WWW/CEAWeb/ceaHome.htm`, October 1994. NASA Reference Publictaion 1311.

[3] S. Kao and Joseph E. Shepherd. Numerical Solution Methods for Control Volume Explosions and ZND Detonation Structure. Galcit report fm2006.007, Aeronautics and Mechanical Engineering, California Institute of Technology, September 2008.

[4] John H. S. Lee. *The Detonation Phenomenon*. Cambridge University Press, 2008. ISBN 978-0-511-41392-6.

[5] Hoi Dick Ng, Matei I. Radulescu, Andrew J. Higgins, Nikos Nikiforakis, and John. H. S. Lee. Numerical investigation of the instability for one-dimensional chapman–jouguet detonations with chain-branching kinetics. *Combustion Theory and Modelling*, 9(3):385–401, 2005.

[6] Hoi Dick Ng, Yiguang Ju, and John H. S. Lee. Assessment of detonation hazards in high-pressure hydrogen storage from chemical sensitivity analysis. *International Journal of Hydrogen Energy*, 32(1):93–99, 2007.

[7] Eric Schultz and Joseph E. Shepherd. Validation of detailed reaction mechanisms for detonation simulation.

Explosion Dynamics Laboratory Report FM99-5, Graduate Aeronautical Laboratories, California Institute of Technology,Pasadena, CA 91125, USA, February 2000.

[8] William W. Wood and Zevi W. Salsburg. Analysis of Steady-State Supported One-Dimensional Detonations and Shocks. *Physics of Fluids (1958-1988)*, 3(4):549–566, 1960.

[9] Frank J. Zeleznik. Calculation of detonation properties and effect of independent parameters on gaseous detonations. *ARS Journal*, 32(4):606–615, 1962.

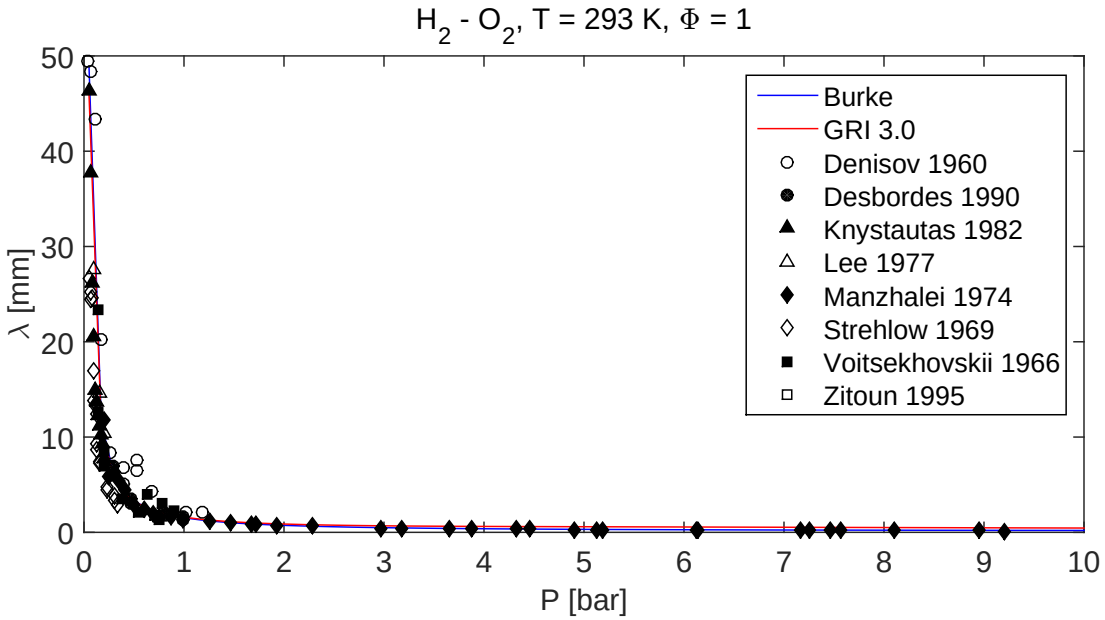

Figure 2: Comparison of numerical and experimental results for H_2-O_2 mixtures

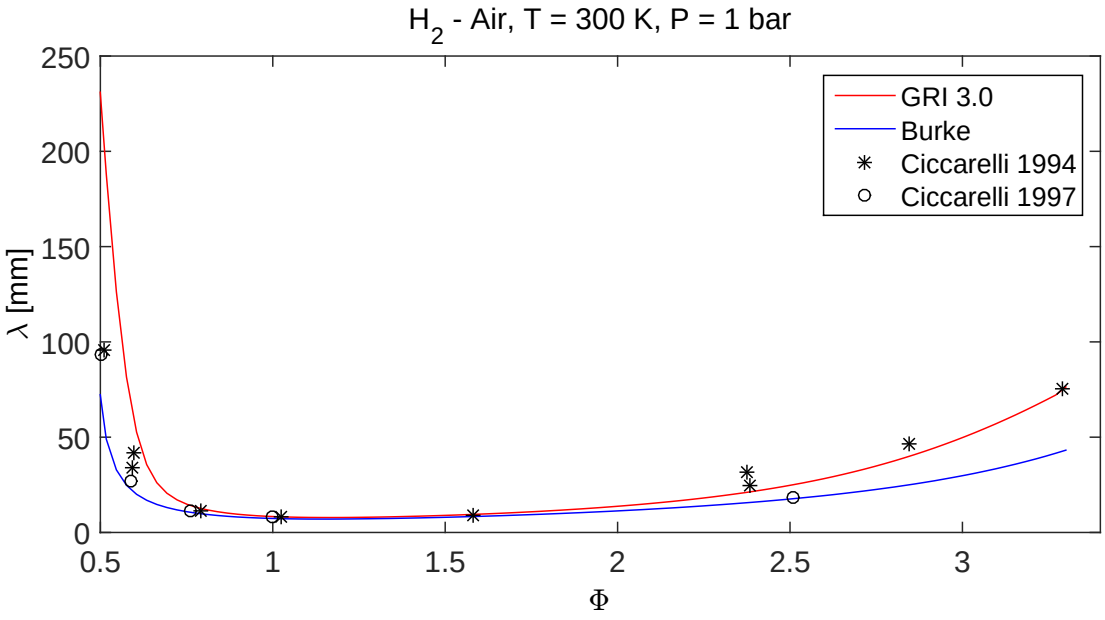

Figure 3: Comparison of numerical and experimental results for H_2-Air mixtures