Chapter 1

Introduction

These notes are a tutorial on the analysis and computation of shock and detonation waves with selected applications to explosion and propulsion. Numerical solution methods are necessary for solving the conservation equations or jump conditions that determine the properties of shock and detonation waves in a multi-component, reacting gas mixture. Only the idealized situations of perfect (constant heat-capacity) gases with fixed chemical energy release can be treated analytically (Appendix B). Although widely used for simple estimates and mathematical analysis, the results of perfect gas models are not suitable for analysis of laboratory experiments and carrying out numerical simulations based on realistic thermochemical properties.

Over the previous 60 years, many numerical solution methods for shock and detonation solutions have been developed and made available as application software. A brief history of some key events is given in Appendix A. Some of these packages are still in use today, however there are issues with using the older software including limited availability due to national security or proprietary concerns and lack of support for legacy software. In response to this situation, we have developed a library of software tools, the **Shock and Detonation Toolbox**, that we are making openly available for academic research. The Toolbox and associated demonstration programs are based on the Cantera software library to evaluate gas thermodynamic and transport properties, chemical reaction rates and carry out chemical equilibrium computations.

In Part I of the report, we describe the algorithms used in the toolbox for the numerical solution of shock and detonation jump conditions in ideal gas mixtures with realistic thermochemical properties. An iterative technique based on a two-variable Newton's method is selected as being the most robust method for both reactive and nonreactive flows. A library of routines is described for Python and Matlab computations of post-shock conditions and Chapman-Jouguet detonation velocity. Notes and demonstration programs are provided to illustrate how to use these routines to solve a range of problems. In addition to numerical methods for realistic thermochemistry, perfect gas analytical solutions are also provided.

In Part II of the report, we describe steady flows and some simple unsteady flows which not in equilibrium or frozen and chemical reaction must be considered. The steady flows treated are the reaction zones behind shock and detonation waves moving at constant speed, the reaction zone along the stagnation streamline in supersonic blunt body flows, flow through a converging-diverging nozzle and quasi-one dimensional flows with friction and heat transfer modeled as wall functions. The unsteady flows modeled include reaction occurring under constant temperature, pressure and volume conditions or with prescribed volume or pressure time dependence.

1.1 Overview and Quickstart

This overview describes situations that are commonly encountered and links to the associated toolbox routines and demonstration programs. For more details on the input and output parameters for these routines, see Chapter 9. For a listing and links to demonstration programs that illustrate various applications of the toolbox, see Chapter 10. In order to use these scripts, the reader must first install the Cantera software and have previously installed Python or Matlab. The toolbox modules, demonstration scripts and instructions for installation are available on the SDToolbox website.

1. Non-reactive shock wave. If the chemical reactions occur sufficiently slowly compared to translational, rotational, and vibrational equilibrium,[1] then a short distance behind a shock wave flow can be considered to be in thermal equilibrium but chemical nonequilibrium. This is often referred to as a "frozen shock" since the chemical composition is considered to be fixed through the shock wave. Computations of post-shock conditions are used as initial conditions for the subsequent reaction zone and are therefore a necessary part of computing shock or detonation structure. Usually, these computations proceed from specified upstream conditions and shock speed; the aim of the computation is to determine the downstream thermodynamic state and fluid velocity. On occasion, we consider the inverse problem of starting from a specified downstream state and computing the upstream state.
   Function PostShock_fr: Demos - Matlab: demo_PSfr.m Python: demo_PSfr.py

2. Reactive shock wave. The region sufficiently far downstream from the shock wave is considered in thermodynamic equilibrium. Thermodynamics can be used to determine the chemical composition, but this is coupled to the conservation equation solutions since the entropy and enthalpy of each species is a function of temperature. As a consequence, the solution of the conservation equations and chemical equilibrium must be self-consistent, requiring an iterative solution for the general case. In the case of endothermic reactions (i.e., dissociation of air behind the bow shock on re-entry vehicle), there are no limits on the specified shock velocity and the computation of the downstream state for specified upstream conditions is straightforward. For exothermic reactions, solutions are possible only for a range of wave speeds separated by a forbidden region. The admissible solutions are detonation (high velocity, i.e., supersonic) and deflagration (low velocity, i.e., subsonic) waves, and there are usually two solutions possible for each case.
   Function PostShock_eq: Demos - Matlab: demo_PSeq.m Python: demo_PSeq.py

3. Chapman-Jouguet (CJ) detonation. This is the limiting case of the minimum wave speed for the supersonic solutions to the jump conditions with exothermic reactions. The Chapman-Jouguet solution is often used to approximate the properties of an ideal steady detonation wave. In particular, detonation waves are often observed to propagate at speeds within 5-10% of their theoretical CJ speeds in experimental situations where the waves are far from failure.
   Function CJSpeed: Demos - Matlab: demo_CJ.m Python: demo_CJ.py

4. Reflected shock wave. When a detonation or shock wave is incident on a hard surface, the flow behind the incident wave is suddenly stopped, creating a reflected shock wave that propagates in the opposite direction of the original wave. If we approximate the reflecting surface as rigid, then we can compute the speed of the reflected shock wave given the incident shock strength. This computation is frequently carried out in connection with estimating structural loads from shock or detonation waves.
   Function reflected_eq and reflected_fr:
   Demos - Matlab: demo_reflected_eq.m and demo_reflected_fr.m
   Python: demo_reflected_eq.py and demo_reflected_fr.py

5. ZND Model Detonation Structure Computation. The idealized reaction zone behind a steady shock or detonation wave is one-dimensional reactive flow. The model equations and properties were first explored by Zel'dovich (1940), von Neumann (1942), and Doering (1943). The solution method used in the toolbox is to convert the differential-algebraic equations representing the conservation of mass, momentum, energy and species evolution to a fully differential system of ODE and integral these with a method suitable for stiff equations.
   Function ZND:
   Demos - Matlab: demo_ZNDCJ.m, demo_ZNDshk.m and demo_ZND_CJ_cell.m
   Python: demo_ZNDCJ.py, demo_ZNDshk.py and demo_ZND_CJ_cell.py

[1]The structure of shock waves with vibration non-equilibrium is discussed at length by Clarke and McChesney (1964) and Vincenti and Kruger (1965)

6. CV Model Explosion Structure Computation. The time-evolution of a mass of fluid reacting at constant volume is frequently used as a surrogate for the reaction process behind incident and reflected shock waves, as well as detonations. The model equations are based on the first law of thermodynamics for an adiabatic, constant-volume system. The ordinary differential equations for energy conservation and species evolution are integrated with a stiff ODE solver.
   Function CV:
   Demos - Matlab: demo_cv.m, demo_cv_comp.m, demo_cvCJ.m, demo_cvshk.m
   Python: demo_cvCJ.py, demo_cvshk.py

Chapter 9

Functions

A summary is provided of the major functions of the toolbox. The basic syntax, input, and output are provided for Matlab and Python. For each function, links are given to both the Matlab and Python implementations. See the website to download and install the complete software package. There are a number of auxiliary files that are required but are not described here.

Core Functions

The core functions for Matlab are in subdirectories in the `SDToolbox` subdirectory of the Matlab toolbox directory, for Python the functions are contained within Python scripts in the `sdtoolbox` subdirectory of the Python `site-packages` directory. Each function contains a header that describes the input and output variables as well as optional parameters.

PostShock CJ speed, frozen and equilibrium state following shock waves

CJSpeed Calculates CJ detonation velocity for a given pressure, temperature, and composition and gas object.

CJSpeed.m

```
FUNCTION SYNTAX:
    If only CJ speed required:
    U_cj = CJspeed(P1,T1,q,mech)
    If full output required:
    [U_cj, curve, goodness, dnew, plot_data] = CJspeed(P1,T1,q,mech)

INPUT:
    P1 = Initial Pressure (Pa)
    T1 = Initial Temperature (K)
    q = string of reactant species mole fractions
    mech = cti file containing mechanism data (i.e. 'gri30.cti')

OUTPUT:
    cj_speed = CJ detonation speed (m/s)
    curve = cfit object of LSQ fit
    goodness = goodness of fit statistics for curve
    dnew = CJ density ratio
    plot_data = structure containing additional parameters to use
```

`CJSpeed` in PostShock.py

```
FUNCTION SYNTAX:
    If only CJ speed required:
    cj_speed = CJspeed(P1,T1,q,mech)
    If full output required:
    [cj_speed,R2,plot_data] = CJspeed(P1,T1,q,mech,fullOutput=True)

INPUT:
    P1 = initial pressure (Pa)
    T1 = initial temperature (K)
    q = reactant species mole fractions in one of Cantera's recognized formats
    mech = cti file containing mechanism data (e.g. 'gri30.cti')

OPTIONAL INPUT:
    fullOutput = set True for R-squared value and pre-formatted plot data
                 (the latter for use with sdtoolbox.utilities.CJspeed_plot)

OUTPUT
    cj_speed = CJ detonation speed (m/s)
    R2 = R-squared value of LSQ curve fit (optional)
    plot_data = tuple (rr,w1,dnew,a,b,c)
                rr = density ratio
                w1 = speed
                dnew = minimum density
                a,b,c = quadratic fit coefficients
```

PostShock_eq Calculates equilibrium post-shock state for a specified shock velocity, pressure, temperature, and composition and gas object.

PostShock_eq.m

```
FUNCTION SYNTAX:
    [gas] = PostShock_eq(U1,P1,T1,q,mech)

INPUT:
    U1 = shock speed (m/s)
    P1 = initial pressure (Pa)
    T1 = initial temperature (K)
    q = reactant species mole fractions in one of Cantera's recognized formats
    mech = cti file containing mechanism data (e.g. 'gri30.cti')

OUTPUT:
    gas = gas object at equilibrium post-shock state
```

PostShock_eq in PostShock.py

```
FUNCTION SYNTAX:
    gas = PostShock_eq(U1,P1,T1,q,mech)

INPUT:
    U1 = shock speed (m/s)
    P1 = initial pressure (Pa)
    T1 = initial temperature (K)
    q = reactant species mole fractions in one of Cantera's recognized formats
    mech = cti file containing mechanism data (e.g. 'gri30.cti')

OUTPUT:
    gas = gas object at equilibrium post-shock state
```

PostShock_fr Calculates frozen post-shock state for a specified shock velocity, pressure, temperature, and composition and gas object.

PostShock_fr.m

```
FUNCTION SYNTAX:
    [gas] = PostShock_fr(U1,P1,T1,q,mech)

INPUT:
    U1 = shock speed (m/s)
    P1 = initial pressure (Pa)
    T1 = initial temperature (K)
    q = reactant species mole fractions in one of Cantera's recognized formats
    mech = cti file containing mechanism data (e.g. 'gri30.cti')

OUTPUT:
    gas = gas object at frozen post-shock state
```

PostShock_fr in PostShock.py

```
FUNCTION SYNTAX:
    [gas] = PostShock_fr(U1,P1,T1,q,mech)

INPUT:
    U1 = shock speed (m/s)
    P1 = initial pressure (Pa)
    T1 = initial temperature (K)
    q = reactant species mole fractions in one of Cantera's recognized formats
    mech = cti file containing mechanism data (e.g. 'gri30.cti')

OUTPUT:
    gas = gas object at frozen post-shock state
```

Reflections Calculated state behind a shock or detonation after reflection from a rigid surface.

reflected_eq Calculates equilibrium post-reflected-shock state.

reflected_eq.m

```
FUNCTION SYNTAX:
    [p3,UR,gas3] = reflected_eq(gas1,gas2,gas3,UI)

INPUT:
    gas1 = gas object at initial state
    gas2 = gas object at post-incident-shock state (already computed)
    gas3 = working gas object
    UI = incident shock speed (m/s)

OUTPUT:
    p3 = post-reflected-shock pressure (Pa)
    UR = reflected shock speed (m/s)
    gas3 = gas object at equilibrium post-reflected-shock state
```

reflected_eq in reflections.py

```
FUNCTION SYNTAX:
    [p3,UR,gas3] = reflected_eq(gas1,gas2,gas3,UI)

INPUT:
    gas1 = gas object at initial state
    gas2 = gas object at post-incident-shock state (already computed)
    gas3 = working gas object
    UI = incident shock speed (m/s)

OUTPUT:
    p3 = post-reflected-shock pressure (Pa)
    UR = reflected shock speed (m/s)
    gas3 = gas object at equilibrium post-reflected-shock state
```

reflected_fr Calculates frozen post-reflected-shock state.

reflected_fr.m

```
FUNCTION SYNTAX:
    [p3,UR,gas3] = reflected_fr(gas1,gas2,gas3,UI)

INPUT:
    gas1 = gas object at initial state
    gas2 = gas object at post-incident-shock state (already computed)
    gas3 = working gas object
    UI = incident shock speed (m/s)

OUTPUT:
    p3 = post-reflected-shock pressure (Pa)
    UR = reflected shock speed (m/s)
    gas3 = gas object at frozen post-reflected-shock state
```

`reflected_fr` in reflections.py

```
FUNCTION SYNTAX:
    [p3,UR,gas3] = reflected_fr(gas1,gas2,gas3,UI)

INPUT:
    gas1 = gas object at initial state
    gas2 = gas object at post-incident-shock state (already computed)
    gas3 = working gas object
    UI = incident shock speed (m/s)

OUTPUT:
    p3 = post-reflected-shock pressure (Pa)
    UR = reflected shock speed (m/s)
    gas3 = gas object at frozen post-reflected-shock state
```

ZND Model Detonation Structure Computation[1]

zndsolve.m

```
FUNCTION SYNTAX:
[output] = zndsolve(gas,gas1,U1)
```

[1]For a detailed exposition of the constant volume (CV) explosion, stagnation reaction zone and one-dimensional detonation structure (ZND) models see the companion document (Browne et al., 2005) from the Shock and Detonation Toolbox website.

```
INPUT:
gas = Cantera gas object - postshock state
gas1 = Cantera gas object - initial state
U1 = Shock Velocity

OPTIONAL INPUT (name-value pairs):
     t_end = end time for integration, in sec
     rel_tol = relative tolerance
     abs_tol = absolute tolerance
     advanced_output = calculates optional extra parameters
     such as induction lengths

OUTPUT:
    output = a dictionary containing the following results:
        time = time array
        distance = distance array

        T = temperature array
        P = pressure array
        rho = density array
        U = velocity array
        thermicity = thermicity array
        species = species mass fraction array

        M = Mach number array
        af = frozen sound speed array
        g = gamma (cp/cv) array
        wt = mean molecular weight array
        sonic = sonic parameter (c^2-U^2) array

        tfinal = final target integration time
        xfinal = final distance reached

        gas1 = a copy of the input initial state
        U1 = shock velocity

        and, if advanced_output=True:
        ind_time_ZND = time to maximum thermicity gradient
        ind_len_ZND = distance to maximum thermicity gradient
        exo_time_ZND = pulse width (in secs) of thermicity  (using 1/2 max)
        ind_time_ZND = pulse width (in meters) of thermicity (using 1/2 max)
        max_thermicity_width_ZND = according to Ng et al definition
```

CV Model Explosion Computation

cvsolve.m

```
FUNCTION SYNTAX:
    output = cvsolve(gas,varargin)

INPUT:
    gas = working gas object

OPTIONAL INPUT (name-value pairs):
    t_end = end time for integration, in sec. If not included
            as an input, set to 10^-3 by default.%
     rel_tol = relative tolerance
     abs_tol = absolute tolerance

OUTPUT:
    output = a structure containing the following results:
         time = time array
         T = temperature profile array
         P = pressure profile array
         speciesY = species mass fraction array
         speciesX = species mole fraction array

         gas = working gas object

         exo_time = pulse width (in secs) of temperature gradient (using 1/2 max)
         ind_time = time to maximum temperature gradient
         ind_len = distance to maximum temperature gradient
         ind_time_10 = time to 10% of maximum temperature gradient
         ind_time_90 = time to 90% of maximum temperature gradient
```

Stagnation Reaction zone structure computation for blunt body flow using the approximation of linear gradient in mass flux = rho u

stgsolve.m

```
SYNTAX
[output] = stgsolve(gas,gas1,U1,Delta)

INPUT
gas = Cantera gas object - postshock state
gas1 = Cantera gas object - initial state
U1 = Shock Velocity
Delta = shock standoff distance

OPTIONAL INPUT (positional argument):
t_end = end time for integration, in sec. If not included
        as an input, set to 10^-3 by default.

OUTPUT
Structure
  output.time = time array
  output.distance = distance array

  output.T = temperature array
  output.P = pressure array
  output.rho = density array
  output.U = velocity array
  output.thermicity = thermicity array

  output.M = Mach number array
  output.af = frozen sound speed array
  output.g = gamma (cp/cv) array
  output.wt = mean molecular weight array
  output.sonic = sonic parameter (c^2-U^2) array
```

Thermo Computation of sound speed and Grüneisen coefficent.

soundspeed_eq Computes the equilibrium sound speed by using a centered finite difference approximation. Directly evaluating pressure at two density/specific volume states along an isentrope requires use of equilibrate('SV'). However, this may not always converge at high pressure. Instead, a more robust method using equilibrate('TP') is used that employs thermodynamic identities detailed further in Appendix G2 of the report.

soundspeed_eq.m

```
FUNCTION SYNTAX:
    aequil =  soundspeed_eq(gas)

INPUT:
    gas = working gas object (restored to original state at end of function)

OUTPUT:
    aequil = equilibrium sound speed = sqrt({d P/d rho)_s, eq) (m/s)
```

`soundspeed_eq` in thermo.py

```
FUNCTION SYNTAX:
    ae =  soundspeed_eq(gas)

INPUT:
    gas = working gas object (restored to original state at end of function)

OUTPUT:
    ae = equilibrium sound speed = sqrt({d P/d rho)_s, eq) (m/s)
```

soundspeed_fr Computes the frozen sound speed by using a centered finite difference approximation and evaluating frozen composition states on the isentrope passing through the reference (S, V) state supplied by the gas object passed to the function.

soundspeed_fr.m

```
FUNCTION SYNTAX:
    afrozen =  soundspeed_fr(gas)

INPUT:
    gas = working gas object (restored to original state at end of function)

OUTPUT:
    afrozen = frozen sound speed = sqrt({d P/d rho)_{s,x0})
```

`soundspeed_fr` in thermo.py

```
FUNCTION SYNTAX:
    afrz =  soundspeed_fr(gas)

INPUT:
    gas = working gas object (restored to original state at end of function)

OUTPUT:
    afrz = frozen sound speed = sqrt({d P/d rho)_{s,x0})
```

gruneisen_eq Computes the equilibrium Grüneisen coefficient by using a centered finite difference approximation and evaluating equilibrium states on the isentrope passing through the reference

(S, V) state supplied by the gas object passed to the function.

gruneisen_eq.m

```
FUNCTION SYNTAX:
    G_eq =  gruneisen_eq(gas)

INPUT:
    gas = working gas object (restored to original state at end of function)

OUTPUT:
    G_eq = equilibrium Gruneisen coefficient [-de/dp)_{v,eq} =
            -(v/T)dT/dv)_{s,eq} = + (rho/T)(dT/d rho)_{s,eq}]
```

`gruneisen_eq` in thermo.py

```
    FUNCTION SYNTAX:
        G_eq =  gruneisen_eq(gas)

    INPUT:
        gas = working gas object (restored to original state at end of function)

    OUTPUT:
        G_eq = equilibrium Gruneisen coefficient [-de/dp)_{v,eq} =
                -(v/T)dT/dv)_{s,eq} = + (rho/T)(dT/d rho)_{s,eq}]
```

gruneisen_fr Computes the frozen Grüneisen coefficient by using a centered finite difference approximation and evaluating frozen states on the isentrope passing through the reference (S, V) state supplied by the gas object passed to the function.

gruneisen_fr.m

```
FUNCTION SYNTAX:
    G_fr =  gruneisen_fr(gas)

INPUT:
    gas = working gas object (not modified in function)

OUTPUT:
    G_fr = frozen Gruneisen coefficient [-de/dp)_{v,x0} =
            -(v/T)dT/dv)_{s,x0} = + (rho/T)(dT/d rho)_{s,x0}]
```

`gruneisen_fr` in thermo.py

```
     FUNCTION SYNTAX:
        G_fr =  gruneisen_fr(gas)

    INPUT:
        gas = working gas object (not modified in function)

    OUTPUT:
        G_fr = frozen Gruneisen coefficient [-de/dp)_{v,x0} =
                -(v/T)dT/dv)_{s,x0} = + (rho/T)(dT/d rho)_{s,x0}]
```

Internal Functions called as part of iteration process.

shk_calc Calculates frozen post-shock state using Reynolds iterative method (see Section 4.2).
Matlab Function - shk_calc.m
Python Function - shk_calc (in PostShock.py)

```
SYNTAX:
[gas] = shk_calc(U1,gas,gas1,ERRFT,ERRFV)
INPUT:
U1 = shock speed (m/s)
gas = working gas object
gas1 = gas object at initial state
ERRFT,ERRFV = error tolerances for iteration
OUTPUT:
gas = gas object at frozen post-shock state
```

shk_eq_calc Calculates equilibrium post-shock state using Reynolds iterative method (see Section 4.2).
Matlab Function - shk_eq_calc.m
Python Function - shk_calc (in PostShock.py)

```
SYNTAX: [gas] = shk_eq_calc(U1,gas,gas1,ERRFT,ERRFV)
INPUT:
U1 = shock speed (m/s)
gas = working gas object
gas1 = gas object at initial state
ERRFT,ERRFV = error tolerances for iteration
OUTPUT:
gas = gas object at equilibrium post-shock state
```

FHFP
Uses the momentum and energy conservation equations to calculate error in current pressure and the enthalpy guesses (see (4.17) & (4.16)). In this case, state 2 is frozen.
Matlab Function - FHFP.m
Python Function - FHFP (in PostShock.py)

```
SYNTAX:
[FH,FP] = FHFP(U1,gas,gas1)
INPUT:
U1 = shock speed (m/s)
gas = working gas object
gas1 = gas object at initial state
OUTPUT:
FH,FP = error in enthalpy and pressure
```

FHFP_reflected_fr
Uses the momentum and energy conservation equations to calculate error in current pressure and the enthalpy guesses (see (4.17) & (4.16)). In this case, state 3 is frozen.
Matlab Function - FHFP_reflected_fr.m
Python Function - FHFP_reflected_fr (in reflections.py)

```
SYNTAX:
[FH,FP] = FHFP_reflected_fr(u2,gas3,gas2)
INPUT:
u2 = current post-incident-shock lab frame particle speed
gas3 = working gas object
gas2 = gas object at post-incident-shock state (already computed)
OUTPUT:
FH,FP = error in enthalpy and pressure
```

CJ_calc
Calculates the wave speed for the Chapman-Jouguet case using Reynolds' iterative method (see Section 4.2).
Matlab Function - CJ_calc.m
Python Function - CJ_calc (in PostShock.py)

```
SYNTAX:
[gas,w1] = CJ_calc(gas,gas1,ERRFT,ERRFV,x)
INPUT:
gas = working gas object
gas1 = gas object at initial state
ERRFT,ERRFV = error tolerances for iteration
x = density ratio
OUTPUT:
gas = gas object at equilibrium state
w1 = initial velocity to yield prescribed density ratio
```

state

Calculates frozen state given T and ρ.
Matlab Function - state.m
Python Function - state (in Thermo.py)

```
SYNTAX:
[P,H] = state(gas,r1,T1)
INPUT:
gas = working gas object
r1,T1 = desired density and temperature
OUTPUT:
P,H = pressure and enthalpy
```

eq_state

Calculates equilibrium state given T and ρ.
Matlab Function - eq_state.m
Python Function - eq_state (in Thermo.py)

```
SYNTAX:
[P,H] = eq_state(gas,r1,T1)
INPUT:
gas = working gas object
r1,T1 = desired density and temperature
OUTPUT:
P,H = equilibrium pressure and enthalpy at constant temperature and specific volume
```

hug_eq

Algebraic expressions of equilibrium (product) Hugoniot pressure and enthalpy. Passed to root solver 'fsolve'.
Matlab Function - hug_eq.m
Python Function - hug_eq (in PostShock.py)

```
SYNTAX:
[x,fval] = fsolve(@hug_eq,Ta,options,gas,array)
INPUT:
Ta = initial guess for equilibrium Hugoniot temperature (K)
options = options string for fsolve
gas = working gas object
array = array with the following values
   vb = desired equilibrium Hugoniot specific volume (m^3/kg)
   h1 = enthalpy at state 1 (J/kg)
   P1 = pressure at state 1 (Pa)
   v1 = specific volume at state 1 (m^3/kg)
OUTPUT:
x = equilibrium Hugoniot temperature corresponding to vb (K)
fval = value of function at x
```

hug_fr

Algebraic expressions of frozen (reactant) Hugoniot pressure and enthalpy. Passed to root solver 'fsolve'.
Matlab Function - hug_fr.m
Python Function - hug_fr (in Thermo.py)

```
SYNTAX:
[x,fval] = fsolve(@hug_fr,Ta,options,gas,array)
INPUT:
Ta = initial guess for frozen Hugoniot temperature (K)
options = options string for fsolve
gas = working gas object
array = array with the following values
   vb = desired frozen Hugoniot specific volume (m^3/kg)
   h1 = enthalpy at state 1 (J/kg)
   P1 = pressure at state 1 (Pa)
   v1 = specific volume at state 1 (m^3/kg)
OUTPUT:
x = frozen Hugoniot temperature corresponding to vb (K)
fval = value of function at x
```

LSQ_CJspeed

Determines least squares fit of parabolic data.
Matlab Function - N/A
Python Function - LSQ_CJspeed (in Thermo.py)

```
SYNTAX:
[a,b,c,R2,SSE,SST] = LSQ_CJspeed(x,y)
INPUT:
x = independent data points
y = dependent data points
OUTPUT:
a,b,c = coefficients of quadratic function (ax^2 + bx + c = 0)
R2 = R-squared value
SSE = sum of squares due to error
SST = total sum of squares
```

PostReflectedShock_eq

Calculates equilibrium post-reflected-shock state for a specified shock velocity.
Matlab Function - PostReflectedShock_eq.m
Python Function - PostReflectedShock_eq (in reflections.py)

```
\begin{verbatim}
FUNCTION SYNTAX:
    [gas3] = PostReflectedShock_eq(u2,gas2,gas3)

INPUT:
    u2 = current post-incident-shock lab frame particle speed
    gas2 = gas object at post-incident-shock state (already computed)
    gas3 = working gas object

OUTPUT:
    gas3 = gas object at equilibrium post-reflected-shock state
```

PostReflectedShock_fr

Calculates frozen post-reflected-shock state for a specified shock velocity.
Matlab Function - PostReflectedShock_fr.m
Python Function - PostReflectedShock_fr (in reflections.py)

```
SYNTAX:
[gas3] = PostReflectedShock_fr(u2,gas2,gas3)
INPUT:
u2 = current post-incident-shock lab frame particle speed
gas2 = gas object at post-incident-shock state (already computed)
gas3 = working gas object
OUTPUT:
gas3 = gas object at frozen post-reflected-shock state
```

Utilities Plotting and output routines

znd_plot Creates four plots from the solution to a ZND detonation Temperature, pressure, Mach number, and thermicity vs. distance. Optionally, also creates plots of species mass fraction vs. time, for given lists of major or minor species. If major_species= 'All', all species will be plotted together.

znd_fileout Creates 2 formatted text files to store the output of the solution to a ZND detonation. Includes a timestamp of when the file was created, input conditions, and tab-separated columns of output data.

cv_plot Creates two subplots from the solution to a CV explosion: Temperature vs. time, and pressure vs. time. Optionally, also creates plots of species mass fraction vs. time, for given lists of major or minor species. If major_species='All', all species will be plotted together.

CJspeed_plot Creates two plots of the CJspeed fitting routine: both display density ratio vs. speed. The first is very "zoomed in" around the minimum, and shows the quadratic fit plotted through the calculated points. The second shows the same fit on a wider scale, with the minimum and its corresponding speed indicated.

Error Control and Limits Setting iteration error and volume limits

Three parameters control the convergence and bounds on the specific volume for the Newton-Raphson iteration used to solve the jump conditions. These are specified in files located in the SDToolbox directory:

Matlab Function - SDTconfig.m
Python Function - config.py

The default values of these parameters are:

```
ERRFT = 1e-4;
ERRFV = 1e-4;
volumeBoundRatio = 5;
```

The values of the error parameters represent the maximum relative errors allowed for convergence of shock and detonation jump condition computations, see the discussion in Section 4.1. Iteration ceases and the solution is returned when the conditions $\Delta T/T <$ `ERRFT` and $\Delta v/v <$ `ERRFV` are both met.

The value of `volumeBoundRatio` is the lower bound on specific volume ratio v_1/v_2 used as a starting point for the iteration. For shock waves in gases with a high specific heat, higher values of `volumeBoundRatio` may be required in order to get solutions but care must be taken not to select `volumeBoundRatio` larger than the maximum value possible on the Hugoniot. The perfect gas analytical solution for strong shock is a useful estimate if the ratio of specific heats γ is known.

$$\frac{v_1}{v_{2,min}} \geq \frac{\gamma+1}{\gamma-1} \tag{9.1}$$

Chapter 10

Demonstration Programs

A number of demonstration programs are provided with the Shock and Detonation Toolbox. These show how Cantera and the SDT routines can be used to carry out a variety of calculations. The programs are available in the `demos` subdirectories in the Python and Matlab branches of the distribution. The links to the Matlab versions are in given in the following list. Python version of all demonstration programs are also available and have identical names except for the extension `.py` instead of `.m`.

1. demo_CJ.m Computes CJ speed.
2. demo_CJ_and_shock_state.m Computes 2 reflection conditions. 1) equilibrium post-initial-shock state behind a shock traveling at CJ speed (CJ state) followed by equilibrium post-reflected-shock state 2) frozen post-initial-shock state behind a CJ wave followed by frozen post-reflected-shock state
3. demo_CJstate.m Computes CJ speed and CJ state.
4. demo_CJstate_isentrope.m Computes CJ speed, CJ state, isentropic expansion in 1-D Taylor wave, plateau state conditions.
5. demo_cv_comp.m Generates plots and output files for a constant volume explosion simulation where the initial conditions are adiabaically compressed reactants.
6. demo_cvCJ.m Generates plots and output files for a constant volume explosion simulation where the initial conditions are given by the postshock conditions for a CJ speed shock wave.
7. demo_cvshk.m Generates plots and output files for a constant volume explosion simulation where the initial conditions are given by the postshock conditions for shock wave traveling at a user specified speed.
8. demo_detonation_pu.m Computes the Hugoniot and pressure-velocity $(P - U)$ relationship for shock waves centered on the CJ state. Generates an output file.
9. demo_equil.m Computes the equilibrium state at constant (T, P) over a range of temperature for a fixed pressure and plots composition.
10. demo_EquivalenceRatioSeries.m - An example of how to vary the equivalence ratio over a specified range and for each resulting composition, compute constant volume explosion and ZND detonation structure. This example creates a set of plots and an output file.
11. demo_exp_state.m Calculates mixture properties for explosion states (UV,HP, TP).
12. demo_ExplosionSeries.m How to compute basic explosion parameters as a function of concentration of one component for given mixture. Creates plots and output file.
13. demo_g.m Compares methods of computing ratio of specific heats and logarithmic isentrope slope using several approaches and compares the results graphically.

14. demo_GasPropAll.m Mixture thermodynamic and transport properties of gases at fixed pressure as a function of temperature. Edit to choose either frozen or equilibrium composition state. The mechanism file must contain transport parameters for each species and specify the transport model 'Mix'.

15. demo_oblique.m Calculates shock polar using FROZEN post-shock state based the initial gas properties and the shock speed. Plots shock polar using three different sets of coordinates.

16. demo_overdriven.m Computes detonation and reflected shock wave pressure for overdriven waves. Both the post-initial-shock and the post-reflected-shock states are equilibrium states. Creates output file.

17. demo_OverdriveSeries.m This is a demonstration of how to vary the Overdrive (U/U_{CJ})) in a loop for constant volume explosions and ZND detonation simulations.

18. demo_PrandtlMeyer.m Calculates Prandtl-Meyer function and polar. Creates plots of polars.

19. demo_PrandtlMeyer_CJ.m Calculates Prandtl-Meyer function and polar expanded from CJ state. Creates plots of polars and fluid element trajectories.

20. demo_PrandtlMeyerDetn.m Calculates Prandtl-Meyer function and polar originating from CJ state. Calculates oblique shock wave moving into expanded detonation products or a specified bounding atmosphere. Creates a set of plots, evaluates axial flow model for rotating detonation engine.

21. demo_PrandtlMeyerLayer.m Calculates Prandtl-Meyer function and polar originating from lower layer postshock state. Calculates oblique shock wave moving into expanded detonation products or a specified bounding atmosphere. Two-layer version with arbitrary flow in lower layer (1), oblique wave in upper layer (2). Upper and lower layers can have various compositions as set by user.

22. demo_precompression_detonation.m Computes detonation and reflected shock wave pressure for overdriven waves. Varies density of initial state and detonation wave speed. Creates an output file.

23. demo_PressureSeries.m Properties computed as a function of initial pressure for a constant volume explosions and ZND detonation simulations Creates a set of plots and an output file.

24. demo_PSeq.m Calculates the equilibrium post shock state based on the initial gas state and the shock speed.

25. demo_PSfr.m Calculates the frozen postshock state based on the initial gas state and the shock speed.

26. demo_quasi1d_eq.m Computes ideal quasi-one dimensional flow using equilibrium properties to determine exit conditions for expansion to a specified pressure. Carries out computation for a range of helium dilutions.

27. demo_reflected_eq.m Calculates post-relected-shock state for a specified shock speed speed and a specified initial mixture. In this demo, both shocks are reactive, i.e. the computed states behind both the incident and reflected shocks are equilibrium states.

28. demo_reflected_fr.m Calculates post-relected-shock state for a specified shock speed speed and a specified initial mixture. In this demo, both shocks are frozen, i.e. there is no composition change across the incident and reflected shocks.

29. demo_RH.m Creates arrays for Rayleigh Line with specified shock speed, Reactant, and Product Hugoniot Curves for H_2-air mixture. Options to creates output file and plots.

30. demo_RH_air.m Creates arrays for Rayleigh Line with specified shock speed and frozen Hugoniot Curve for a shock wave in air. Options to create output file and plot.

31. demo_RH_air_eq.m Creates arrays for Rayleigh Line with specified shock speed in air, frozen and equilibrium Hugoniot curves. Options to create output file and plot.

32. demo_RH_air_isentropes.m Creates arrays for frozen Hugoniot curve for shock wave in air, Rayleigh Line with specified shock speed, and four representative isentropes. Options to create plot and output file.

33. demo_RH_CJ_isentropes.m Creates plot for equilibrium product Hugoniot curve near CJ point, Shows Rayleigh Line with slope U_{CJ} and four isentropes bracketing CJ point. Creates plot showing Gruneisen coefficient, denominator in Jouguet's rule, isentrope slope.

34. demo_rocket_impulse.m Computes rocket performance using quasi-one dimensional isentropic flow using both frozen and equilibrium properties for a range of helium dilutions in a hydrogen-oxygen mixture. Plots impulse as a function of dilution.

35. demo_RZshock.m Generate plots and output files for a reaction zone behind a shock front traveling at a user specified speed.

36. demo_shock_adiabat.m Generates the points on a frozen shock adiabat and creates an output file.

37. demo_shock_point.m This is a demonstration of how to compute frozen and equilibrium postshock conditions for a single shock Mach number. Computes transport properties and thermodynamic states.

38. demo_shock_state_isentrope.m Computes frozen post-shock state and isentropic expansion for specified shock speed. Create plots and output file.

39. demo_ShockTube.m Calculates the solution to ideal shock tube problem. Three cases possible: conventional nonreactive driver (gas), constant volume combustion driver (uv), CJ detonation (initiate at diaphragm) driver (cj).

40. demo_STGshk.m Generate plots and output files for a steady reaction zone between a shock and a blunt body using the model of linear profile of mass flux ρu on stagnation streamline.

41. demo_STG_RZ.m Compare propagating shock and stagnation point profiles using transformation methodology of Hornung.

42. demo_TP.m Explosion computation simulating constant temperature and pressure reaction. Reguires function `tpsys.m` for ODE solver

43. demo_TransientCompression.m Explosion computation simulating adiabatic compression ignition with control volume approach and effective piston used for compression. Requires `adiasys.m` function for ODE solver.

44. demo_vN_state.m Calculates the frozen shock (vN = von Neumann) state of the gas behind the leading shock wave in a CJ detonation.

45. demo_ZNDCJ.m Solves ODEs for ZND model of detonation structure. Generate plots and output files for a for a shock front traveling at the CJ speed.

46. demo_ZNDshk.m Solves ODEs for ZND model of detonation structure. Generate plots and output files for a for a shock front traveling at a user specified speed U.

47. demo_ZND_CJ_cell.m Computes ZND and CV models of detonation with the shock front traveling at the CJ speed. Evaluates various measures of the reaction zone thickness and exothermic pulse width, effective activation energy and Ng stability parameter. Estimates cell size using three correlation methods: Westbrook; Gavrikov et al; and Ng et al.