

SHIMADA LAB. CODE

MANUAL

山中翔太

2014年1月23日

目次

第 I 部 基本事項	3
1 はじめに	4
2 チュートリアル	5
2.1 化学コードを用いたメタン/空気完全平衡モデル簡略化モデル作成	5
2.2 簡略モデルを用いた作動流体にメタンと空気を用いた衝撃波管問題	12
2.3 可視化	21
第 II 部 各機能の説明	25
3 ライブラリ生成アルゴリズムについて	26
3.1 ディレクトリ構造	26
4 流体コード	27
4.1 自動生成ファイル	27
4.2 store/core	28
4.3 store/architecture	30
4.4 store/dim	32
4.5 store/high_order	32
4.6 store/time_step	32
4.7 store/time_scheme	32
4.8 store/viscosity	34
4.9 store/therm.lib	35
4.10 store/cond	48
5 NASA 熱力学データベース	49

6	0 次元化学コード	50
6.1	store/therm.lib/NASA	51
6.2	store/therm.lib/chemkin	53

第Ⅰ部

基本事項

1 はじめに

本プログラムは「流体コード」「化学モデル」「0次元化学コード」の生成プログラムである。

本章ではまずははじめに、「化学コードを用いたメタン/空気完全平衡モデル簡略化モデル作成」と「簡略モデルを用いた作動流体にメタンと空気を用いた衝撃波管問題」という二つの例題を解きながら、全てのコードに関わる重要事項について

重要

このボックスを用いながら

説明する。

その後、condition.f90 で用いられる”すべり壁”と”粘着壁”について説明する。

最後に、デバッグに使える多少のコマンドについて説明する。

2 チュートリアル

2.1 化学コードを用いたメタン/空気完全平衡モデル簡略化モデル作成

この例題は”元モデルの作成”と”その簡略化”にわけられる。それぞれを説明する。

2.1.1 元モデルの作成

コードの生成は checkout.py ができるが、

重要

checkout.py ができる動作は全て GUI.py を通して行うことができる。

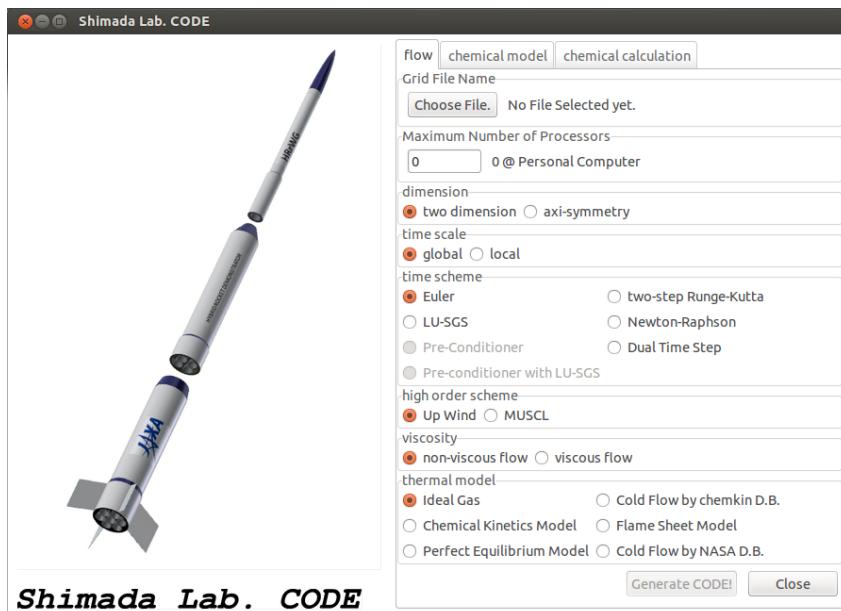
よって、コード生成は GUI.py を通して行うことをお勧めする。

GUI.py は以下のコマンドで起動される。

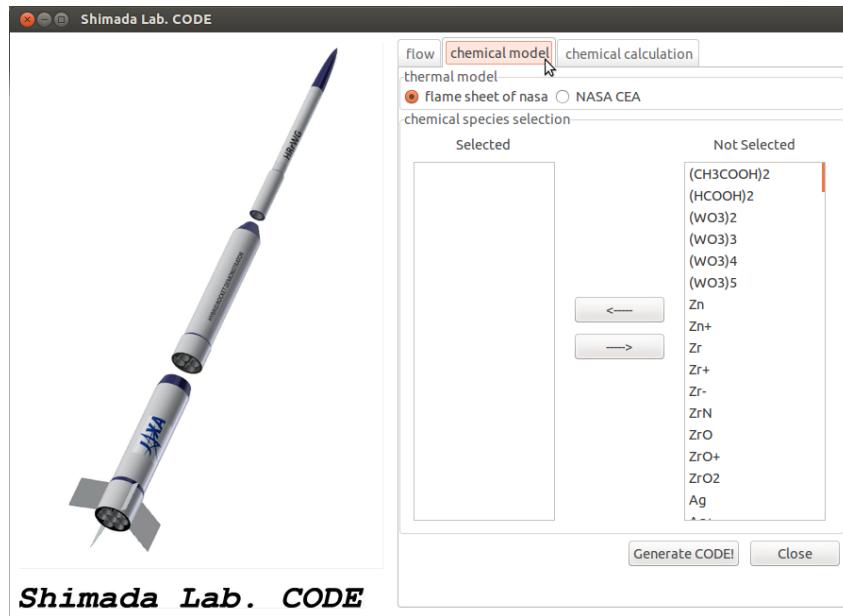
重要

./GUI.py

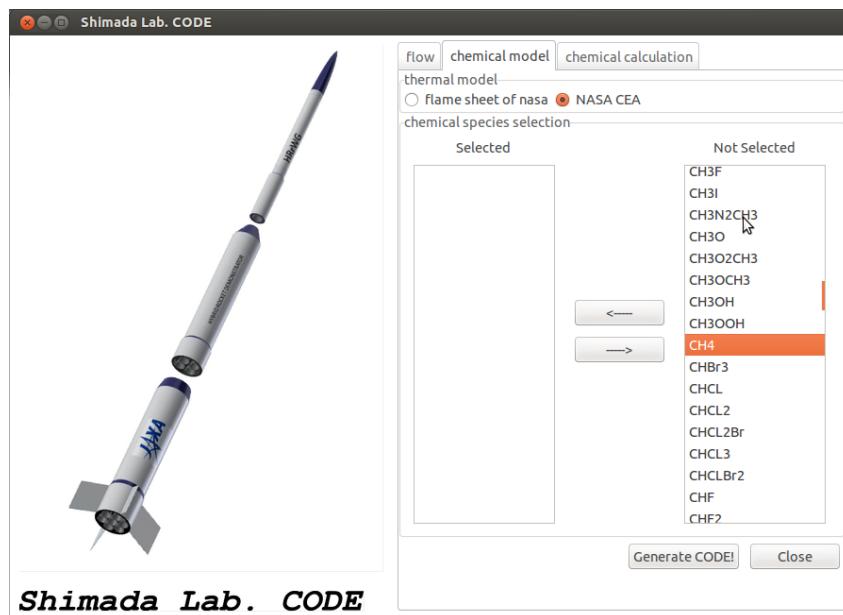
起動されると以下の画面が立ち上がる。

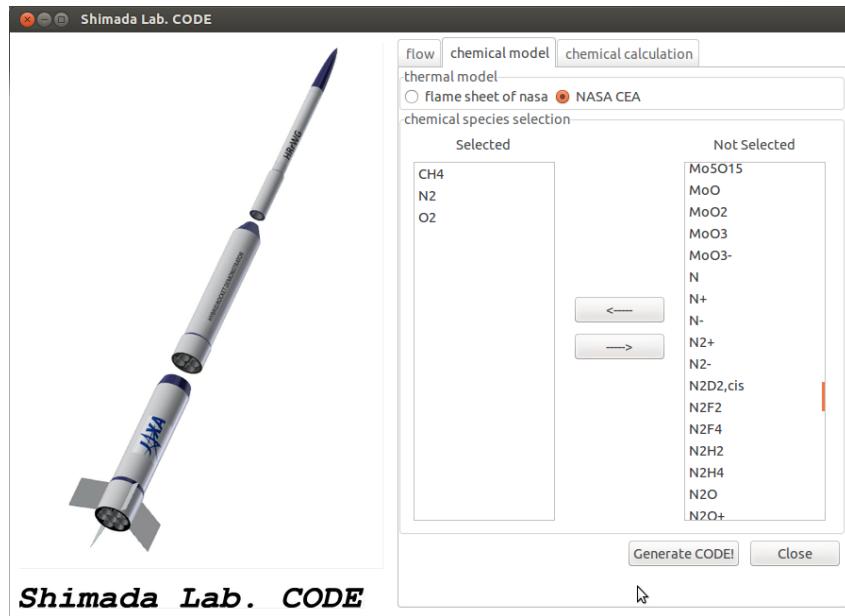


今回はまず元モデルの生成を行う。まずは上のタブで”chemical model”を選択する。

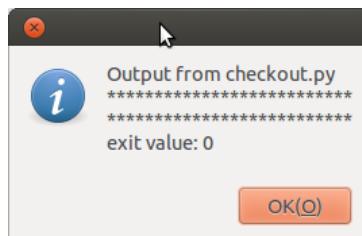


起動されたら、まず thermal model で完全平衡モデル”NASA CEA”を選び、“Not Selected”の中から生成物として使いたい化学種をダブルクリックすることで選択する。今回はメタン空気の燃焼なので、CH₄, N₂, O₂の三つを選べばよい。なお、この際”Not Selected”内で Ctrl+L をうつと検索モードとなり、所望の化学種を素早くみつけることができる。





選び終えたら”Generate CODE!”を押下すると、コードが生成される。



重要

流体コード生成、化学コード生成の場合でも、”exit value:0”が正常終了である。”*****”の行から”*****”の行までがコード生成の際に生成されたメッセージとなるので、正常終了しなかった場合にはこのメッセージを読んで対策を講じること。

上の例の場合はコード生成の際にメッセージは生成していない。

以上の動作を終えるとディレクトリに”chem.inp”と”control_chem.raw.inp”が生成される。

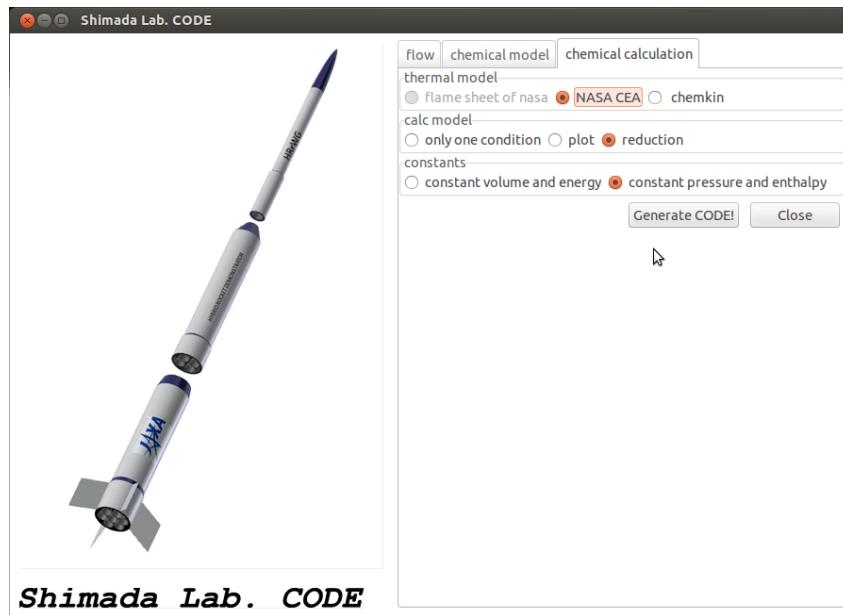
重要

熱力学モデルが理想気体でない場合、化学コードでも流体コードでも、コード生成の際には checkout.py と同じ階層に使用される chem.inp を配置する必要があり、また生成したコードを走らせる際には”control_chem.raw.inp”を編集し”control_chem.inp”として実行ファイルと同じ階層に配置する必要がある。

NASA データベースを用いる際には”control_chem.inp”的形はこの操作でしか生成しないので取扱いに気をつけること。

2.1.2 モデルの簡略化

続けて化学コードの生成を行う。このためには GUI.py の上のタブで”chemical calculation”を選ぶ。



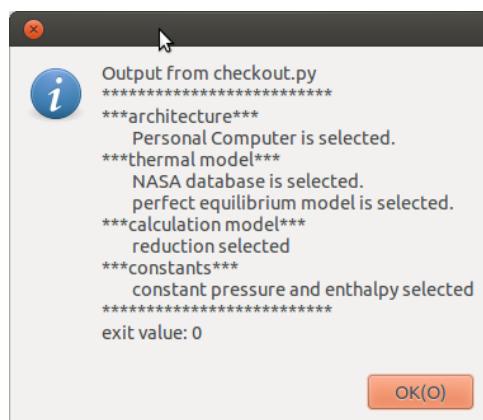
図のように

- thermal model : NASA CEA
- calc model : reduction
- constants : constant pressure and enthalpy

を選んだら”Generate CODE!”を押下する。

重要

この際、”chem.inp”が”checkout.py”と同じ階層にいなければ異常終了するので注意すること。



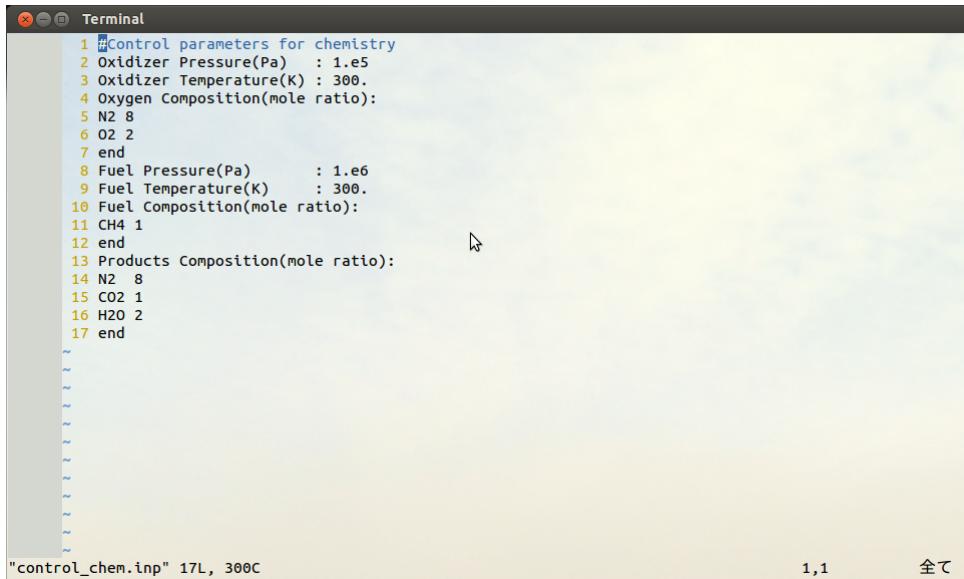
コード生成時の”checkout.py”からのメッセージから適切な設定が生成されたこと、また”exit value:0”からコード生成に成功したことがわかる。ここで OK を押すと、自動的に GUI.py が閉じられる。

以上の動作を行うと、ディレクトリは以下のようになる。



checkout_chem が新たに生成されていることが分かる。

ここで、control_chem.raw.inp を以下のように設定する。



control_chem.inp は燃料と酸化剤それぞれの組成、圧力、初期温度、必要な場合は生成物の組成と時間積分ライブラリに使うパラメータを設定する。今回は完全平衡モデルであるので本来ならば Products の欄はいらないが、流体に用いるために定義しておく。詳しい書式は 4.9.2 に記載している。以上のように設定できたら control_chem.raw.inp を control_chem.inp に名前変更し、checkout_chem にコピーする。すると checkout_chem の中身は以下のようになる。


```
$ ls
LU.f90    chem.inp    control_chem.inp  init.vi      n_grid.f90    trans.inp
MW.inp    conditions.f90  driver.f90    mod_chem.f90  sub_chem.f90
Makefile  control.inp   func_chem.f90  mod_mpi_dummy.f90 thermo.inp
$
```

ここまで終了したら make コマンド

make

をうつ。するとコードがコンパイルされるので

./driver

を実行すると、以下のように出力される。

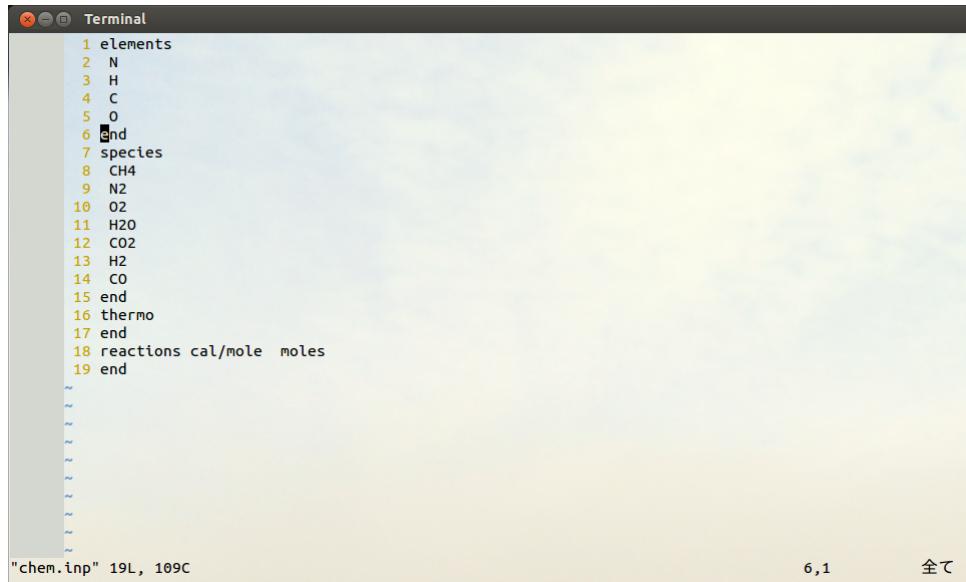
```
$ ./driver
the number of species of chem.inp:      158
the number of species of trans.inp:      24
not converted. at cea_hp
not converted. at cea_hp
not converted. at cea_hp
not converted. at cea_hp
final error(%):  1.70
final ns       :  7
$
```

ここから、元々のモデルでは 158 あった化学種が 7 まで削減され、また最大温度誤差が 1.7% であることがわかる。

重要

なお、実行ファイルは流体コードでは”main”、化学コードでは”driver”である。

ここで生成されたファイルは chem.inp.new というファイルで保存される。chem.inp.new のファイルの中は以下のようにになっている。



```
Terminal
1 elements
2 N
3 H
4 C
5 O
6 end
7 species
8 CH4
9 N2
10 O2
11 H2O
12 CO2
13 H2
14 CO
15 end
16 thermo
17 end
18 reactions cal/mole moles
19 end
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
 
"
"chem.inp" 19L, 109C 6,1 全て
```

ここから、僅か 7 つの化学種のみが用いられていることがわかる。これを chem.inp と名前変更して使用すると、元モデルの代わりに簡略モデルを用いることができる。

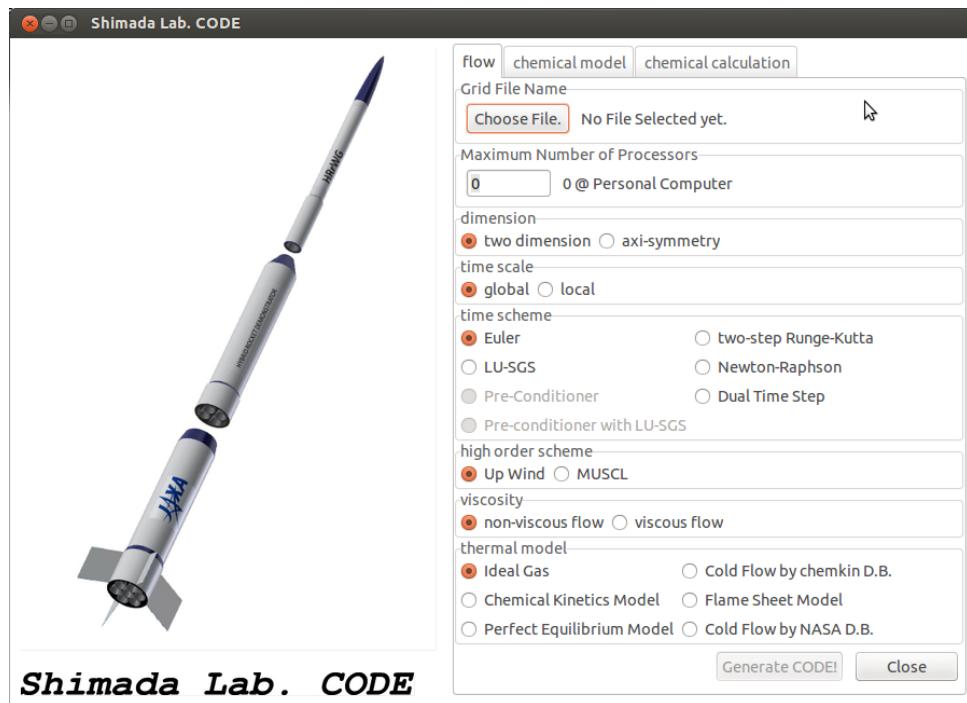
2.2 簡略モデルを用いた作動流体にメタンと空気を用いた衝撃波管問題

前節が終わった時点で checkout.py と同じ階層にある chem.inp には 158 化学種存在する。これを簡略化モデルに置き換えるには checkout.py と同じ階層で

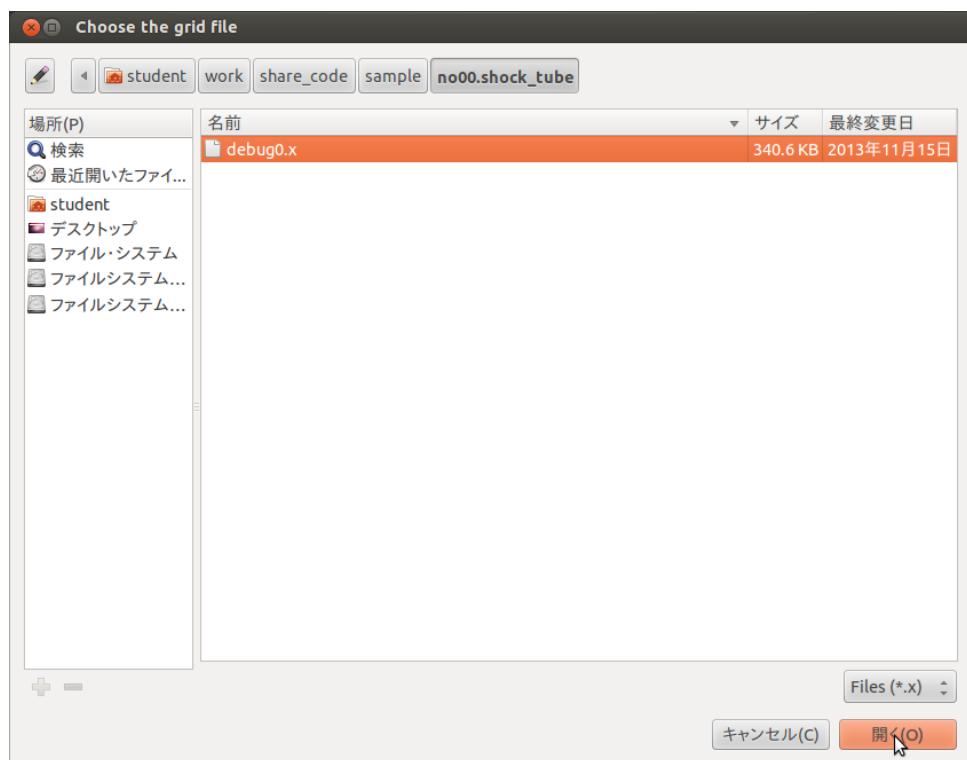
```
cp checkout_chem/chem.inp.new chem.inp
```

などのコマンドをうつなどして、checkout_chem で生成した chem.inp.new で checkout.py と同じ階層にある chem.inp を置き換えればよい。

流体コードを生成するには、また./GUI.py で GUI を起動すればよい。



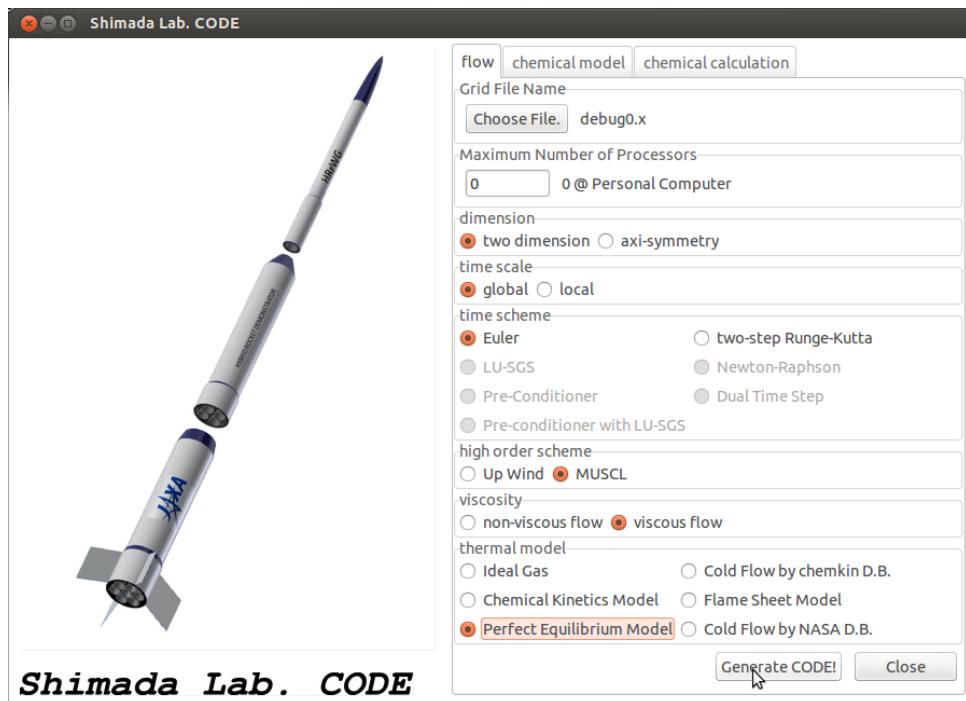
まずは Grid File Name でグリッドを選ぶ。”Choose File”を押下すれば、ファイル選択画面が現れる。本チュートリアルでは sample/no00.shock_tube/debug0.x を選択する。



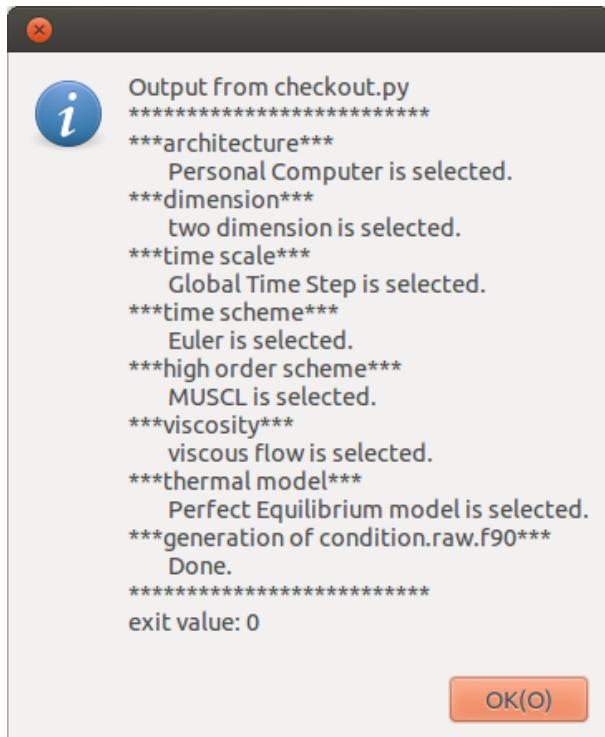
グリッドファイル選択後は

- Maximum Number of Processors : 0
- dimension : two dimension
- time scale : global
- time scheme : Euler
- high order scheme : MUSCL
- viscosity : viscous flow
- thermal model : Perfect Equilibrium Model

を選ぶ。それぞれの詳しい意味は section 4 参照のこと。



この状態で”Generate CODE!”を押下すると以下のようなメッセージがでてくる。このメッセージから、正常終了していること (Exit Value: 0)、所望の機能が生成されていることがわかる。



重要

この際に、checkout ディレクトリ生成以前に元々 checkout ディレクトリが存在する場合、その中に存在する control.inp と condition.f90、grid ディレクトリについてはそれぞれ control.bak.inp、condition.bak.f90、grid_bak ディレクトリとしてバックアップが checkout.py と同じ階層にとられる。

以上の操作を行うと、checkout ディレクトリが生成し、中身が以下のようになる。

```
Terminal
$ ls
LU.f90          func_chem.f90  main.f90        result      time_euler_viscous.f90
MW.inp          gen_cond.py   mod_chem.f90   sch_viscous.f90  trans.inp
Makefile         geometry.f90  mod_mpi_dummy.f90 scheme.f90   variable.f90
check_convergence.f90  grid      muscl.f90     set_dt.f90
chem.inp         init.vi      n_grid.f90    sub_chem.f90
condition.raw.f90  init_sq.f90  prmtr.f90   thermal_model.f90
control.raw.inp  inout.f90    read_control.f90 thermo.inp
```

まず、NASA ライブリを使ってるので、前述のとおり control_chem.inp をコピーしてくる必要がある。よって、簡略化の際用いたファイルをコピーしてくる。

```
$ ls
LU.f90           control_chem.inp  inout.f90      read_control.f90  thermo.inp
MW.inp          func_chem.f90    main.f90      result            time_euler_viscous.f90
Makefile        gen_cond.py     mod_chem.f90  sch_viscous.f90  trans.inp
check_convergence.f90  geometry.f90  mod_mpi_dummy.f90  scheme.f90       variable.f90
chem.inp         grid           muscl.f90    set_dt.f90
condition.raw.f90  init.vi      n_grid.f90  sub_chem.f90
control.raw.inp  init_sq.f90   prmtr.f90   thermal_model.f90
$
```

また、前述のとおり”raw”がついているものを編集しなければならない。今回は control.inp と condition.f90 である。control.inp は以下のように編集する。

```
1 #FOR GENERAL PARAMETERS
2 Max Step Number      : 1.e3
3 convergent RMS       : 1.e-14
4 CFL Number           : 0.3
5 File Output Period   : 100
6
7
8 #FOR MUSCL
9 MUSCL ON(1)/OFF(0)   : 1
10 MUSCL Precision Order : 3
11
12
13 #FOR CFH
14 Temp. Diff Limit(K) : 1.e-3
15 cfh check freq.     : 10
~
```

これも詳しい書式は section 4 参照のこと。

condition.f90 については、左半分を空気、右半分をメタンのそれぞれ静止気体とする。初期圧力は control_chem.inp でそれぞれ 1 気圧、10 気圧としているので、これは衝撃波管問題となる。

condition.f90 は冒頭を以下に記す。

```
%{{{
subroutine set_IC
```

```

use grbl_prmtr
use prmtr
use variable
use mod_mpi
use chem_var
implicit none
integer i,j,plane

do plane=nps,npe
  do i=nxs(plane),50
    do j=nys(plane),nye(plane)
      q(:,     i,j,plane) = qo
      w(:,     i,j,plane) = wo
    end do
  end do
  do i=51,nxe(plane)
    do j=nys(plane),nye(plane)
      q(:,     i,j,plane) = qf
      w(:,     i,j,plane) = wf
    end do
  end do
end do
end subroutine set_IC

subroutine set_BC(step)
use grbl_prmtr
use prmtr
use variable
use mod_mpi
use chem_var
implicit none
integer,intent(in)::step
integer i,j,plane

integer,parameter::DLength=dimw+2*nV !for MPI Communication
integer,parameter::MaxMPIcomm = 0

call section_exchange
call MPI_COMMUNICATIONS_CUT_LINE

```

```

!boundary right and left
do plane=nps,npe
  select case(plane)
    case( 1)
      if(gx(plane) .eq. 1) then
        !i=1/2
        do j=max( 1,nys(plane)),min(nye(plane), 49)
          w(:, 0,j,plane) = w(:, 1,j,plane)
          vhi(:, 0,j,plane) = vhi(:, 1,j,plane)
          Yv(:, 0,j,plane) = Yv(:, 1,j,plane)
          w(:, -1,j,plane) = w(:, 0,j,plane)
          vhi(:, -1,j,plane) = vhi(:, 0,j,plane)
          Yv(:, -1,j,plane) = Yv(:, 0,j,plane)
        end do
      end if

      if(gx(plane) .eq. ngx(plane)) then
        !i=ni+1/2
        do j=max( 1,nys(plane)),min(nye(plane), 49)
          w(:, ni(plane)+1,j,plane) = w(:, ni(plane) ,j,plane)
          vhi(:,ni(plane)+1,j,plane) = vhi(:,ni(plane) ,j,plane)
          Yv(:, ni(plane)+1,j,plane) = Yv(:, ni(plane) ,j,plane)
          w(:, ni(plane)+2,j,plane) = w(:, ni(plane)+1,j,plane)
          vhi(:,ni(plane)+2,j,plane) = vhi(:,ni(plane)+1,j,plane)
          Yv(:, ni(plane)+2,j,plane) = Yv(:, ni(plane)+1,j,plane)
        end do
      end if
    end select
  end do

  call MPI_COMMUNICATIONS_I_DIRECTION

!boundary upper and lower
do plane=nps,npe
  select case(plane)
    case( 1)
      if(gy(plane) .eq. 1) then
        !j=1/2
        do i=max( 1,nxs(plane)),min(nx(e(plane), 99)

```

```

w(:, i, 0,plane) =w(:, i, 1,plane)
vhi(:,i, 0,plane) =vhi(:,i, 1,plane)
Yv(:, i, 0,plane) =Yv(:, i, 1,plane)
w(:, i,-1,plane) =w(:, i, 0,plane)
vhi(:,i,-1,plane) =vhi(:,i, 0,plane)
Yv(:, i,-1,plane) =Yv(:, i, 0,plane)

end do
end if

if(gy(plane) .eq. ngy(plane)) then
!j=nj+1/2
do i=max( 1,nxs(plane)),min(nxe(plane), 99)
w(:, i,nj(plane)+1,plane) = w(:, i,nj(plane) ,plane)
vhi(:,i,nj(plane)+1,plane) = vhi(:,i,nj(plane) ,plane)
Yv(:, i,nj(plane)+1,plane) = Yv(:, i,nj(plane) ,plane)
w(:, i,nj(plane)+2,plane) = w(:, i,nj(plane)+1,plane)
vhi(:,i,nj(plane)+2,plane) = vhi(:,i,nj(plane)+1,plane)
Yv(:, i,nj(plane)+2,plane) = Yv(:, i,nj(plane)+1,plane)
end do
end if
end select
end do

call MPI_COMMUNICATIONS_J_DIRECTION

call set_corners
contains

```

(以下省略。condition.raw.f90 と変更なし)

以上のように、NASA データベースを用いる場合にはモジュール chem_var を用いることで qo,wo,qf,wf の利用が可能になる。このようなデータベースは熱力学モデル毎に決められており、section 4 で詳細の説明がなされている。

以上の変更を行うと、コンパイルが可能になる。

```

make
./main

```

の二つのコマンドをうつと、最終的には以下のように出力される。

```

Terminal
step=    760 fs/all(%)=  90.91 Yf_cfh=  9.0E-08 Yo_cfh=  2.4E-02
step=    770 fs/all(%)=  88.89 Yf_cfh=  1.8E-08 Yo_cfh=  9.0E-07
step=    780 fs/all(%)=  87.88 Yf_cfh=  3.3E-08 Yo_cfh=  7.3E-07
step=    790 fs/all(%)=  88.89 Yf_cfh=  5.5E-08 Yo_cfh=  5.9E-07
step=    800 log(RMS)=  2.1824060E+00 dt_grbl=  2.0457402E-06
step=    800 fs/all(%)=  89.90 Yf_cfh=  8.7E-08 Yo_cfh=  1.4E-02
step=    810 fs/all(%)=  89.90 Yf_cfh=  2.1E-08 Yo_cfh=  3.0E-02
step=    820 fs/all(%)=  89.90 Yf_cfh=  3.5E-08 Yo_cfh=  2.6E-02
step=    830 fs/all(%)=  89.90 Yf_cfh=  5.4E-08 Yo_cfh=  2.3E-02
step=    840 fs/all(%)=  89.90 Yf_cfh=  8.1E-08 Yo_cfh=  2.0E-02
step=    850 fs/all(%)=  88.89 Yf_cfh=  2.1E-08 Yo_cfh=  1.8E-02
step=    860 fs/all(%)=  88.89 Yf_cfh=  3.4E-08 Yo_cfh=  1.5E-02
step=    870 fs/all(%)=  88.89 Yf_cfh=  5.2E-08 Yo_cfh=  9.5E-07
step=    880 fs/all(%)=  89.90 Yf_cfh=  7.6E-08 Yo_cfh=  7.8E-07
step=    890 fs/all(%)=  87.88 Yf_cfh=  2.1E-08 Yo_cfh=  6.3E-07
step=    900 log(RMS)=  2.1535851E+00 dt_grbl=  1.9638576E-06
step=    900 fs/all(%)=  88.89 Yf_cfh=  3.2E-08 Yo_cfh=  2.2E-02
step=    910 fs/all(%)=  88.89 Yf_cfh=  4.7E-08 Yo_cfh=  1.9E-02
step=    920 fs/all(%)=  89.90 Yf_cfh=  6.8E-08 Yo_cfh=  1.6E-02
step=    930 fs/all(%)=  89.90 Yf_cfh=  2.0E-08 Yo_cfh=  3.4E-02
step=    940 fs/all(%)=  89.90 Yf_cfh=  2.9E-08 Yo_cfh=  3.0E-02
step=    950 fs/all(%)=  88.89 Yf_cfh=  4.2E-08 Yo_cfh=  2.6E-02
step=    960 fs/all(%)=  87.88 Yf_cfh=  5.9E-08 Yo_cfh=  9.8E-07
step=    970 fs/all(%)=  87.88 Yf_cfh=  1.8E-08 Yo_cfh=  8.0E-07
step=    980 fs/all(%)=  87.88 Yf_cfh=  2.7E-08 Yo_cfh=  6.5E-07
step=    990 fs/all(%)=  88.89 Yf_cfh=  3.8E-08 Yo_cfh=  1.5E-02
step=   1000 log(RMS)=  2.1316109E+00 dt_grbl=  1.9097683E-06
Error:Exceed Max Step. NOT CONVERTED.
normal end
$
```

重要

ここで”ERROR”とでているが、これは本コードが control.inp で設定した最大ステップ数を越えて收束しなかったため出力されており、非定常現象を解く場合、正常な終了である。

重要

なお、実行ファイルは流体コードでは”main”、化学コードでは”driver”である。

結果は result ディレクトリ内に保存される。

```

Terminal
$ ls result/
result000000000100.bin  result000000000400.bin  result000000000700.bin  result000000001000.bin
result000000000200.bin  result000000000500.bin  result000000000800.bin
result000000000300.bin  result000000000600.bin  result000000000900.bin
$
```

2.3 可視化

前節で出力された結果は fortran unformatted ファイルのため、そのままでは可視化ソフトに読ませることはできない。よって、可視化用の変換を行わなければならない。

変換は checkout.py と同じ階層にある postprocess ディレクトリ内の postprocess.f90 ができる。postprocess ディレクトリ内は以下のようにになっている。



```
Terminal
$ ls
MW.inp      bin2vtk.f90      control.inp      modify.f90      result      visdebug.f90
Makefile    change_num_step.f90  flow_history.f90  postprocess.f90  trunk
$
```

まずはコンパイルを行う。

重要

コンパイルには checkout ディレクトリ内の n_grid.f90 を用いるので、格子を変更した際には make をやり直すこと。

make を実行すると、実行ファイル postprocess が生成される。この入力ファイル control.inp は以下のように設定する。

```

1 #Input File for PostProcessing
2 File Read Freq.          : 100
3 From Where              : -1
4 Database Place          : ../../checkout/
5 OverWrite? Yes(1)/No(0) : 0
6 1st Plane i start index : 0
7 1st Plane i end index : -1
8 1st Plane j start index : 0
9 1st Plane j end index : -1
10
~
```

"control.inp" 10L, 274C 書込み 10,0-1 全て

それぞれの意味は section 4 参照のこと。

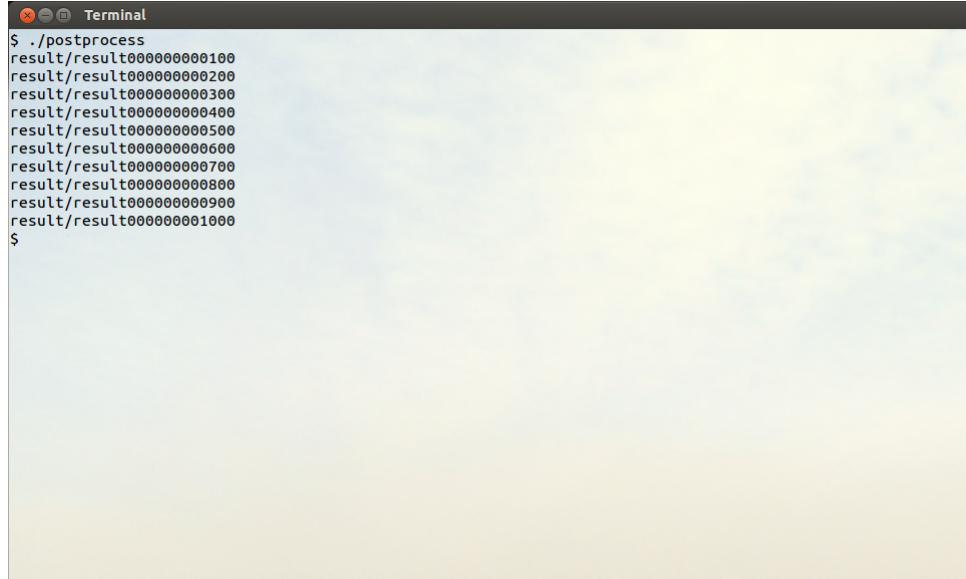
なお、postprocess は必要な情報が得られれば残りの行は無視する。今回の場合 10 行目以降無視されるので、以下のように書いてあっても動作はまったく同じである。

```

1 #Input File for PostProcessing
2 File Read Freq.          : 100
3 From Where              : -1
4 Database Place          : ../../checkout/
5 OverWrite? Yes(1)/No(0) : 0
6 1st Plane i start index : 0
7 1st Plane i end index : -1
8 1st Plane j start index : 0
9 1st Plane j end index : -1
10 1st Plane i start index : -1
11 1st Plane i end index : 210
12 1st Plane j start index : 0
13 1st Plane j end index : 80
14 1st Plane i start index : 0
15 1st Plane i end index : -1
16 1st Plane j start index : 0
17 1st Plane j end index : -1
18 2nd Plane i start index : 0
19 2nd Plane i end index : -1
20 2nd Plane j start index : 0
21 2nd Plane j end index : -1
22 3rd Plane i start index : 0
23 3rd Plane i end index : -1
24 3rd Plane j start index : 0
25 3rd Plane j end index : -1
26
27 #Database Place          : ../../checkout/
~
```

"control.inp" 27L, 776C 書込み 10,1 全て

実行すると、以下のように出力される。

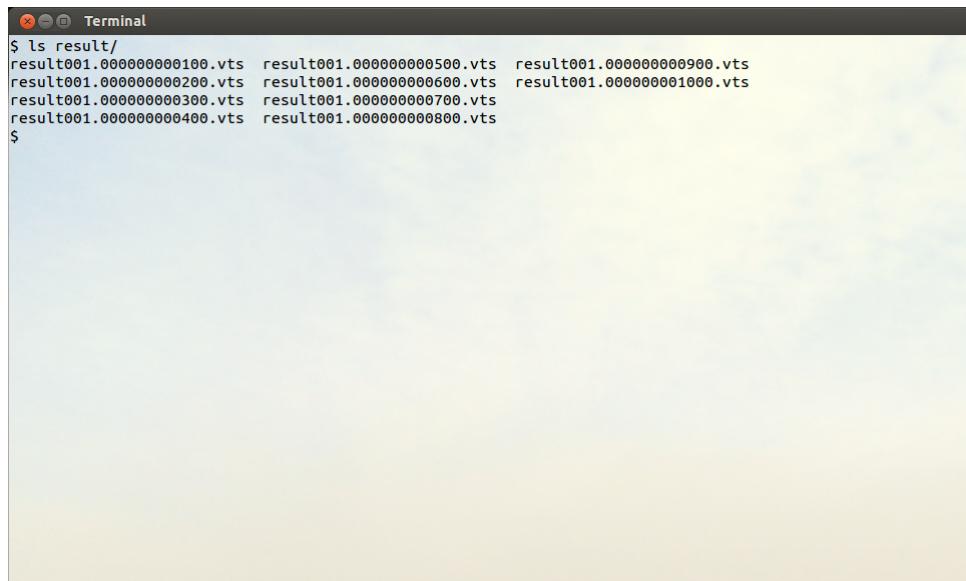


```
Terminal
$ ./postprocess
result/result000000000100
result/result000000000200
result/result000000000300
result/result000000000400
result/result000000000500
result/result000000000600
result/result000000000700
result/result000000000800
result/result000000000900
result/result000000001000
$
```

出力は可視化を試行したファイル名(拡張子は除く)である。出力は vts ファイルとして postprocess/result に保存される。

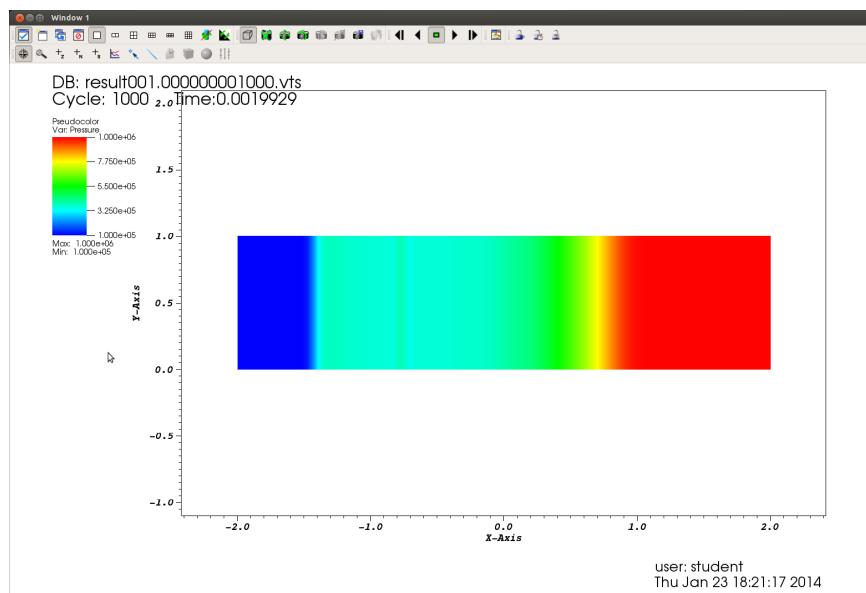
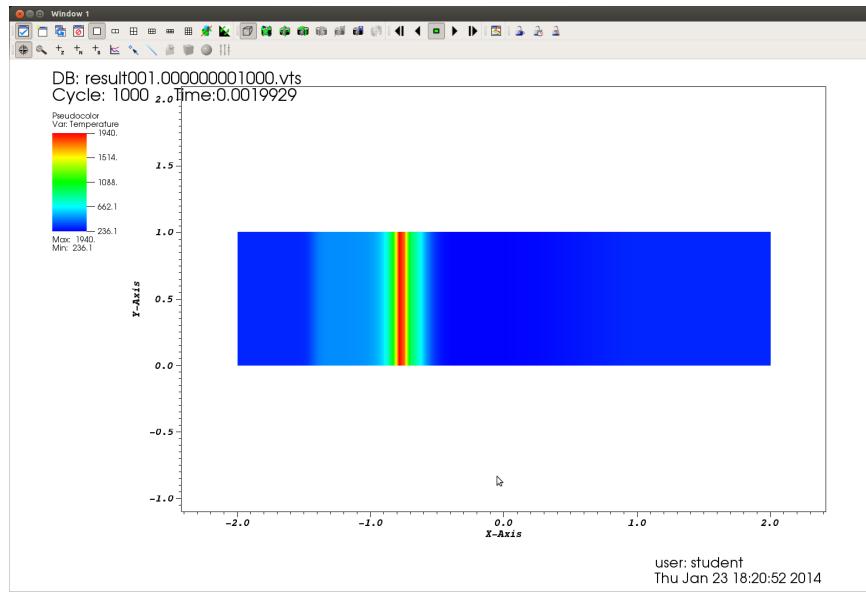
重要 —

なお、すでに同じファイル名で vts ファイルが存在する場合上書きするかどうかを control.inp で設定することができる。上書きしない場合、ファイルを更新しても vts ファイルは更新されないので、その場合は vts ファイルを削除すること。



```
Terminal
$ ls result/
result001.000000000100.vts  result001.000000000500.vts  result001.000000000900.vts
result001.000000000200.vts  result001.000000000600.vts  result001.000000001000.vts
result001.000000000300.vts  result001.000000000700.vts
result001.000000000400.vts  result001.000000000800.vts
$
```

これを可視化ソフトに入れればよい。例えば visit を使うと、以下のように接触面で発熱している衝撃波管問題の結果を見ることができる。



第 II 部

各機能の説明

3 ライブリ生成アルゴリズムについて

本プログラムは基本的に「ライブラリ生成プログラム」となる。よって、各種パラメータ設定ファイルに加えて、流体計算の場合は”condition.f90”というファイルで直接初期条件と境界条件を設定しなければならない。これらのファイルは”raw”というサフィックス付きでディレクトリにコピーされる。

GUI インターフェースである GUI.py は、checkout.py を動かすための input ファイル (checkout.inp, checkout_model.inp, checkout_chem.inp) を生成し、checkout.py を実行するプログラムであり、主動作は GUI.py を用いた場合でも多くを checkout.py が受け持つ。

checkout.py は input ファイルに基づき、各種ライブラリを生成するプログラムである。基本的にライブラリの生成は、該当ファイルをディレクトリからコピーすることによって行い、ある特定のファイルを除いては fortran ソースファイル自体を checkout.py が編集することはない。コピーされるファイルは全て”store”のサブディレクトリ内に入っている。checkout.py またはユーザーによって編集が加えられるファイルには”.raw.”というサフィックスがつけられている。なお、checkout.py 内でそのソースファイルの編集が終了する、つまりそれ以上ユーザーによる編集が不要になったファイルからは自動的に”.raw.”サフィックスが取り除かれ、そのままコンパイルにかけられるようになる。

3.1 ディレクトリ構造

基本的に類似する機能を持つファイルがまとめられている。例えばオイラー陽解法と LU-SGS 法は”time_scheme”ディレクトリ内に”euler”, ”lu-sgs”としてそれぞれ格納されている。また化学計算と流体計算は基本的に同一ライブラリを用いることになっている。以下、全てのディレクトリについて説明を行う。

4 流体コード

checkout.inp を記入し、checkout.py を起動すると生成される。実処理は checkout.py から呼び出された”store/checkout/checkout_flow.py”によって生成される。生成されたファイルは”checkout”ディレクトリ内に生成される。

4.1 自動生成ファイル

流体コードで自動生成されるファイルは、checkout.py で選択されたディレクトリ内にある”部分的なファイル”をつなげることで生成される。自動生成されるファイル及びその”部分的なファイル”は以下のとおりである。

- main.f90... 主プログラムがかけたソースとなる。主ループもここで回される。以下の順にファイルの内容が追記されていく。
 - main.top.f90...”program main”からモジュールのインポートまで。
 - main.head.f90... 共通して利用する変数の定義。
 - main.variable.f90... それぞれのスキームで特有に利用される変数の定義。
 - main.part_init.f90... 変数の初期化。グリッドの読み込みやそのバイナリファイルの生成や幾何ヤコピアンの生成、熱力学ライブラリが必要とされる場合にはその初期化も行われる。
 - main.body.raw.f90... 初期化。restart.bin がある場合はそれを用い、ない場合は初期化サブルーチンを用いて初期化を行う。また主ループを回す変数も初期化される。part_primitive という文字列がファイル内に記されており、後述する文字列に置き換えられる。主ループの開始まで記述してある。
 - main.main.raw.f90, main.main.point_implicit.f90... 主ループ内部を記述しており、時間スキームのディレクトリ内に格納されている。part_point_implicit, part_primitive, part_main という文字列がファイル内に記されており、それぞれ以下の文字列に置き換えられる part_point_implicit に文字列が存在する場合 main.main.point_implicit.f90、存在しない場合 main.main.raw.f90 が使われる。なお、point implicit なスキームを受け付けない時間積分法 (ex.Dual Time Step) には main.main.point_implicit.f90 は存在せず、無理やり使おうとする場合コンパイルエラーが起きる。
 - * part_point_implicit...point implicit に計算される部分がここに入る。現状、化学生産項の時間積分を進めるサブルーチンがここに入りうる。main.part_point_implicit.f90 にかけられた文字列に置き換えられる。
 - * main.part_primitive.f90... 時間刻みの計算やデータアウトプット前にすべき処理、具体的には基本量の計算及び境界条件の設定をするサブルーチンがここに入る。main.part_primitive.f90 にかけられた文字列に置き換えられる。
 - * main.part_main.f90... 高次精度化や対流項及び拡散項の計算、時間積分に必要な場合にはそれらの保存量ヤコピアンが計算される。main.part_main.f90 にかけられた文字列に置き換えられる。
 - main.foot.f90... 主ループの終了及び MPI 関数の終了処理、”end program main”を記述してある。
- Makefile...Makefile. 以下の順にファイルの内容が追記されていく。

- Makefile.head... コンパイラに何を使うのか定義する。アーキテクチャ (PC かスーパーコンピュータ) によって変更される。
- Makefile.var ... 各種変数の定義
- Makefile.main... 各オブジェクトファイル作成ルール
- Makefile.foot... プログラムの最終的なコンパイル部分及び clean の定義
- control.raw.inp...cfl の定義やファイル出力間隔の定義など。使用する場合、MUSCL のパラメータもここで入力される。control.part.inp に記された内容が入力される。

4.2 store/core

どのような場合にも使用されるファイルがまとめられてある。上記自動生成ファイルのための各種ファイルの他、以下のファイルが存在する。

- check_convergence.f90... 収束判定及び途中経過ファイル,restart ファイルの生成を行っている。
- init.vi...vim 用初期化ファイル。これを読み込むと、RUN コマンドで./main を走らせることができる。
- inout.f90...restart ファイル読み書きや途中経過ファイル書き込み。
- n_grid.raw.f90... 重要なパラメータの定義。以下の文字列は checkout.py で置き換えられる。
 - NPLANE...multi-block で block の数。muti-block でない場合 1.
 - NumI... それぞれの面での i 方向セル数
 - NumJ... それぞれの面での j 方向セル数
 - NIMAX...i 方向セル数最大値
 - NJMAX...j 方向セル数最大値
 - NumY...rho の数.ex) 理想気体:1, flame sheet, 完全平衡モデル:2, 素反応モデル:化学種数
 - NV... 拡散で空間微分をとるべき次元数.flame sheet で 3, 完全平衡モデルで化学種数.
 - GridFileName...plot3d で記述されたグリッドファイルの場所.

また以下の文字列は IC や BC の設定に使える。

- dimq ... q(後述) の次元
- dimw ... w(後述) の次元
- idxg ... w に於ける比熱比のインデックス
- idxht ... w に於ける質量あたり全エンタルピー (J/kg) のインデックス
- idxR ... w に於ける質量あたり気体定数 (J/kg/K) のインデックス
- idxMu ... w に於ける粘性係数 (Pa*s) のインデックス
- prmtr.f90...pi やモルあたりの気体定数”R_uni”の定義.
- read_control.f90...control.inp の読み込み. 全計算に共通する部分に限る。
- scheme.f90... 流束項評価サブルーチン。現在は SLAU のみ。
- variable.f90... 一般的に使われる変数がまとめられてある。
 - q ... 保存量.
 - * インデックス 1 から nY : それぞれの気体グループ (理想気体なら全化学種、flame sheet が完全平衡モデルなら燃料と酸化剤、素反応モデルならそれぞれの化学種) の密度 (kg/m³)
 - * インデックス nY+1 : x 方向運動量 (kg*m/s)

- * インデックス nY+2 : y 方向運動量 (kg*m/s)
- * インデックス nY+3 : 全エネルギー (J/m³)
- qp ... 前ステップの q
- qpp ... 全ステップの qp
- w ... 基本量
 - * インデックス 1 : 全化学種の密度の和 (kg/m³).
 - * インデックス 2 : x 方向速度 (m/s).
 - * インデックス 3 : y 方向速度 (m/s).
 - * インデックス 4 : 圧力 (Pa)
 - * インデックス 5 ... nY+4 : それぞれの気体グループの質量分率
 - * インデックス nY+5(=idxg @n_grid.f90) : 比熱比
 - * インデックス nY+6(=idxht@n_grid.f90) : 全エンタルピー (J/kg)
 - * インデックス nY+7(=idxR @n_grid.f90) : 気体定数 (J/kg/K)
 - * インデックス nY+8(=idxMu@n_grid.f90) : 粘性係数 (Pa*s)
- vhi ... それぞれの気体グループの生成熱を含む内部エンタルピー (J/kg)
- wHli, wHri, wHlj, wHrj ... 高次精度化で予測された w(ex. MUSCL). 'l' は'left'、'r' は'right'、'i' は i 方向に +1/2 を足した位置、'j' は j 方向に 1/2 を足した位置。例えば、wHli(:,2,3) は $w_{2+\frac{1}{2},3}$ の左側の値を表す。
- TG_i, TG_j ... それぞれ i 方向 j 方向の流束項 TG. TG の定義については嶋田テキストの Section 9 参照。これらの値も 1/2 だけセル中心からずれた場所の値であり、ずれ方は wH と同じ。
- TG_{vi}, TG_{vj} ... 粘性項。
- Vol ... 軸対称問題ではそれぞれのセルの体積、二次元問題では面積。
- dsi, dsj ... i 方向 j 方向にそれぞれ垂直なセル辺の長さ。軸対称問題ではさらに半径がかけられる。これもセル中心から 1/2 だけずれた場所の値。
- vni, vnj ... それぞれ dsi, dsj で示された辺の直交正規ベクトル (絶対値 1). 向きは i,j 正の向き。これもセル中心から 1/2 だけずれた場所の値。
- xh, rh ... メッシュ格子点の座標。これもセル中心から 1/2 だけずれた場所の値。
- x, r ... セル中心の座標。
- dt_mat ... local time step のそれぞれのセルでの Δt
- dt_grbl ... global time step の Δt . (スペルミスったので'r' です。)

また全ての計算において、以下の値を control.inp で設定する必要がある。

- Max Step Number... 計算を再開してからの最大ステップ数.
- convergent RMS... 収束とするエネルギー残渣.
- CFL Number... CourantFriedrichsLewy 数.
- File Output Period... ファイル出力周期.

4.3 store/architecture

アーキテクチャ依存のサブルーチン。サブディレクトリは PC と SC でそれぞれ PC とスーパーコンピュータに関するファイルがある。

SC には mod_mpi.f90、PC には mod_mpi_dummy.f90 が使われる。スーパーコンピュータの場合は grid_separation.inp からそれぞれのプロセッサが受け持つグリッドの範囲を計算し、変数を初期化する。また、自動生成されたグリッドの分割の状態は、スーパーコンピュータを選んだ際に生成される vis_grid.dat を用い、gnuplot で

```
plot "vis_grid.dat" with lines
```

のコマンドを打つことで確認することができる。

どちらの場合も以下のように受け持つブロックの範囲, i 方向の範囲, j 方向の範囲が定義される。

- ブロックの範囲...nps から npe
- ブロック plane の i 方向の範囲...nxs(plane) から nxe(plane)
- ブロック plane の j 方向の範囲...nys(plane) から nye(plane)

パソコンの場合、nps=1, npe=Nplane である。

格子ブロックで境界になっているが、ブロック間で共有されており、ユーザーが直接入力する必要のない境界がある。これを本プログラムでは cut_copro*.inp, MPIcomm*.inp を用いて記録している。*はプロセッサ番号で、パソコンでは 0 である。cut_copro*.inp は同一コア間でのデータ転送で足る場合、MPIcomm*.inp は異なるコア間でのデータ転送が必要な場合に使われる。よってパソコンの場合 MPIcomm*.inp は出力されない。

4.3.1 PC での MPI ダミー関数の定義について

PC の場合は MPI 関数のダミー関数を入れている。このダミー関数は完全ではない。つまり、例えば MPI_Reduce の定義は

```
subroutine MPI_Reduce(tmpt, tmp, te, MPI_DOUBLE_PRECISION, MPI_SUM, te2, MPI_COMM_WORLD, ierr)
    double precision tmpt, tmp
    integer te1, MPI_DOUBLE_PRECISION, MPI_SUM, te2, MPI_COMM_WORLD, ierr
    tmp=tmpt
end subroutine MPI_Reduce
```

となっており、これでは MPI_INTEGER に対応できない。MPI 関数を使った変更をプログラムに施す際には mod_mpi_dummy.f90 で行われているダミー関数の定義を参照すること。

4.3.2 スーパーコンピュータ用ユーティリティについて

スーパーコンピュータ用にいくつかのユーティリティプログラムが用意されている。

プロセッサへのグリッド割当について

- generate_proc_grid.py... グリッド分割用ライブラリ。基本的には全てのプロセッサが持つグリッドができるだけ均等になるよう分割するが、split_ratio.py が存在するとき、その split_ratio.ratio(plane,cind) を用いて寄せることが可能である。
- trim_balance.py... それぞれのプロセッサの計算負荷を可視化するためのプログラム。

ジョブキューの投入・確認・削除 基本的に large システムで実行すること。

- mpirun.sh...sh mpirun.sh でジョブキュー投入を行える。check_and_throw.pl 及び StepJob.py は内部的に mpirun.sh をよんでいる。
- check.pl...perl check.pl でジョブの動作状況 (QUEUED, RUNNING, EXISTING) と現在出力されている最新のファイル名を一秒おきに出力する。
- check_and_throw.pl...perl check_and_throw.pl で check.pl の動作を行った上、全てのジョブが終了している場合、新たに sh mpirun.sh を実行する。これを用いると、600 秒に限られたデバッグジョブにおいて、ジョブが終わった瞬間に新しいジョブ投げることができますので、半永久的にデバッグジョブを走らせることが可能。よって、使いっぱなしになると確実にスーパーコンピュータ利用禁止となり卒業できなくなるので、程度を考えて利用すること。経験上一時間程度なら怒られない。
- StepJob.py...python StepJob.py で、ステップジョブを投げられるだけ投げる。つまり、実行時にステップジョブが 19 投げられている場合、50 まで投げられるので、31 回 sh mpirun.sh が実行される。これを cron と組み合わせ、

```
0 0,6,12,18 * * * cd /large/y/y535/2dcfd/share_code; python StepJob.py
```

のようにしておけば、一タターミナルを開きスーパーコンピュータにログインすることなしに、自動的に 6 時間おきに最大数までジョブを予約することができます。また、cron の実行結果はメールで送られるので、現在のステップ数を 6 時間おきにメールで知ることが可能になります。

cron を利用する際の注意点として、cron はサーバー毎に設定できるらしく、同じ maja.jss.jaxa.jp でも maja0.jss.jaxa.jp, maja1.jss.jaxa.jp が存在する。よって、crontab を編集する際には、maja0.jss.jaxa.jp など明示的にどのサーバーを用いるのかを示してログインしなければならない。

- qdel.pl...perl qdel.pl で qdel コマンドを予約している全てのジョブに投げる、つまり予約している全てのジョブをキャンセルできる。実行中のジョブも強制終了させる。プログラムが落ちた際や、間違った境界条件を使用していたことに気づいた際に用いる。

データファイルの転送コマンド

- make remove on SC... スーパーコンピュータ上で打つと、標準入力および標準出力、result 内の全ての結果を削除する。restart.bin は残す。
- make tar on SC... スーパーコンピュータ上で打つと、RMS.dat, geometry.bin, result/*.bin を result.tar.gz ファイルに圧縮する。
- make pull on postprocess...postprocess ディレクトリにおいて、Makefile の変数 remote に result.tar.gz が存在するディレクトリを指定すると、このコマンドでその result.tar.gz をダウンロードし解凍する。

4.4 store/dim

軸対称流か二次元流かを選ぶものである。サブディレクトリは 2d と q2d で、それぞれ以下のファイルが含まれている。

- store/dim/2d...geometry.f90 と init_Sq.f90
- store/dim/q2d...geometry.f90 と Sq.f90

Sq は軸対称流のとき式変形の際ソース項に出てくる値を定義しており、二次元流では 0 である。init_Sq.f90 と Sq.f90 はそれぞれ Sq に関する処理を行っている。geometry.f90 には plot3d で記述されたグリッドファイルの読み込み及び可視化用バイナリファイルの生成、幾何ヤコビアンやセルの面積を計算するルーチンがまとめられている。

4.5 store/high_order

高次精度化を行うルーチンである。サブディレクトリは muscl と upwind であり、それぞれ MUSCL スキームと風上差分が入っている。MUSCL を用いる場合、以下の項目を control.inp で設定する必要がある。

- MUSCL ON(1)/OFF(0)...MUSCL の on/off 切り替え。1 で on, 0 で off. その他の値は受け付けない。
- MUSCL Precision Order...MUSCL のオーダー調整。2 と 3 のみ受けつけ、2 の場合は $\kappa = -1$, 3 の場合は $\kappa = \frac{1}{3}$ に MUSCL のパラメータが設定される。

4.6 store/time_step

global time step または local time step が設定される。ファイルは set_dt.f90。またこのどちらかを選ぶことにより、checkout.py の DT_GLOBAL_LOCAL が dt_mat(i,j,plane) または dt_grbl が設定され、時間積分スキームの該当する部分で置き換えられる。

4.7 store/time_scheme

時間スキームの選択。main.main.raw.f90 や main.main.point_implicit.f90, 時間積分サブルーチン (time_...f90) が入っており、主ループ（時間積分ループ）の中身を定義する。

- euler... オイラー陽解法.
- RK2... ルンゲ=クッタ 2 次精度.
- LU-SGS...LU-SGS で実装したオイラー陰解法. 上記の他 sch_lusgs.f90, var_lusgs.f90 で LU-SGS 法を行う。
- NR...Newton-Raphson 法. 同様に LU-SGS 法を使う. 後述するパラメータを用いる. 上記の他 sch_NR.f90, var_NR.f90 で LU-SGS 法を行う。
- precon... 前処理法を用いた Newton-Raphson 法. 同様に LU-SGS 法を使う. 上記の他 sch_precon.f90, var_precon.f90 で LU-SGS 法を行う. 仮時間ステップにも前処理行列をかけている. 後述するパラメータを用いる.

- dual...Dual Time Step. 同様に LU-SGS 法を使う。後述するパラメータを用いる。上記の他 sch_dual.f90, var_dual.f90 で LU-SGS 法を行う。
- preconLU-SGS... 前処理法をオイラー陰解法 (LU-SGS) で実装している。上記の他 sch_precon.f90, var_precon.f90 で LU-SGS 法を行う。

なお、LU-SGS から preconLU-SGS には”2d”と”q2d”というディレクトリがそれぞれあるが、それぞれ二次元流、軸対称流のコードが収められている。

NR,precon,dual で用いられるパラメータについて これらのスキームでは以下のパラメータを control.inp で定義する。ただし、 ω は内部反復の緩和係数である。

- InternalLoop CFL tau... 疑似時間ステップの cfl 数
- InternalLoop Omega Max... 最大緩和係数。以下 ω_{\max} とする。
- InternalLoop Omega Min... 最小緩和係数。これを割り込む ω が設定されると、計算を中止する。
- InternalLoop Dqrate Max... 各気体グループの密度の内部反復あたり変化割合の最大値。以下 Dq_{\max} とする。
- InternalLoop ResRateWarn... 内部反復が収束していないことを標準出力に投げる最小エネルギー残渣比
- InternalLoop ResRateErr... 計算を中止する最小エネルギー残渣比
- InternalLoop OutOmega... ファイル internal_res.dat に緩和係数 ω の履歴を出すか。
- InternalLoop Max Number... 内部ループ数 (固定)

これらのスキームの内部ループで、保存量 q は緩和係数 ω をかけられた Δq により更新される。 ω は以下のように定義される。

$$\omega = \min(\omega_{\max}, \frac{Dq_{\max}\rho_i}{\Delta\rho_i}) \quad (1)$$

ただし $i = 1 \dots nY$ である。これにより、 ρ_i の一内部反復での変化は Dq_{\max} 以下に抑えられる。この定義であると、 ω が小さくなりすぎることがある。それを補足するのが”最小緩和係数”的パラメータである。

内部ループの発散は’InternalLoop Omega Min’, ’InternalLoop ResRateWarn’, ’InternalLoop ResRateErr’ で補足されることとなる。ここでエネルギー残渣比は(最後の内部反復でのエネルギー残渣)/(最初の内部反復でのエネルギー残渣)と定義している。なお、NR, precon, dual の時間スキームでは、基本コンセプトは”内部反復が収束”することに基づいているので、エネルギー残渣比が十分小さくならない場合はコンセプト自体が崩壊している。よって、その場合これらのスキームは使うべきではない。(これを見落としている研究は大変多く、私は悲しい。)

パラメータ調整方法 パラメータ調整には”internal_res.dat”が使える。これは”InternalLoop OutOmega”を”true”にセットすることで得られる。(このファイル出力は時間がかかるので、本計算のときには効率向上のため”false”にすべき。) このファイルには 1 列目に内部反復数、2 列目に Dqrate により計算された $\omega(\omega_{\max}$ で修正する前)、3 列目にエネルギー残渣比が記録されている。以下にこれらの出力を用いてパラメータを調節する手段の一つを記す。

- ’InternalLoop Max Number’ を増やす... 内部反復数が小さいことが原因で、エネルギー残渣比が指数的に減少しているのに’InternalLoop ResRateErr’ に達していないとき。
- ’InternalLoop Max Number’ を減らす...’InternalLoop Max Number’ に比べ遥かに少ない内部

反復数で'InternalLoop ResRateErr' に達しているとき。

- 'InternalLoop Omega Max' を増やす... 下記の目安に比べ内部反復の収束が遙かに遅く、 ω_{\max} により ω が主に決まっているとき。
- 'InternalLoop Omega Max' を減らす... Dqrate により決定された ω ('internal_res.dat' の第二列目に記録された ω) が指数的に増加せず、一回目の内部反復の ω が ω_{\max} により決定されている場合。この状況は $\omega \Delta q$ があまりにも大きすぎて、一回目の内部反復における q の推測値が悪くなりすぎ、そのためその後のステップで正しい値に修正できなくなった場合に起こる。
- 'InternalLoop Dqrate Max' を増やす... 内部反復の収束が遅く、 ω が Dqrate によって主に決定されている場合。
- 'InternalLoop Dqrate Max' を減らす... ω が指数的に増加せず、一回目の内部反復で Dqrate により ω が決定されている場合。この状況も $\omega \Delta q$ があまりにも大きすぎて、一回目の内部反復における q の推測値が悪くなりすぎ、そのためその後のステップで正しい値に修正できなくなった場合に起こる。

これらの手段は理論的なものではなく、私の経験によるものである。よって間違っている可能性もある。他の方法も試してください。

パラメータの目安

- 0.1 @ Omega Max... もしこれで動く場合、0.8 や 0.9 も可能な場合もある。Dual Time Step では 0.01 を使わざるを得なかった場合もある。
- 0.001 @ Omega Min... この値を下回り正しい解を出した計算に出会ったことはない。このパラメータは固定すべきであると思う。
- 0.01 @ Dqrate Max... 現実問題このパラメータは多分子流にのみ使われる。Dual Time Step では 0.001 を使わざるを得なかった場合もある。経験上 0.1 は大きすぎる。
- 1.e-5 @ ResRateWarn, 1.e-3 @ ResRateErr... 経験上、エネルギー残渣比は 3 オーダーは下げなければ正しい解を出さない。よって 1e-3 は固定すべきだと考える。ResRateWarn は警告メッセージを出すだけなので、好みの値を用いてよい。
- 60 @ Max Number... 理想気体では経験上 20 で十分。多分子流を Dual Time Step で解いたとき、100 必要だった場合もあった。

4.8 store/viscosity

non-viscous と viscous というディレクトリがあり、viscous にはその中に with-nV と no-nV、さらにそれに”2d”と”q2d”というディレクトリが収められている。non-viscous が非粘性流、viscous が粘性流である。2d と q2d の違いは time_scheme と同じ。

with-nV と no-nV について、化学モデルには”単純に ρ_i の空間微分でエンタルピー拡散を扱えるもの”とそうでないものがあり、それぞれが with-nV と no-nV に対応している。with-nV の場合、”Yv”という”質量拡散によるエンタルピー拡散を考える際考慮しなければならない気体グループの質量分率”が熱力学ライブラリから生成される。例えば、一段総括反応の場合、化学組成は”反応物としての”燃料と酸化剤の質量のみを追いかければいいが、質量拡散によるエンタルピー拡散を計算する場合”生成物としての”燃料、酸化剤、生成物それぞれの質量分率を計算せねばならず、with-nV が使用される。

ファイルは sch_viscous.f90 である。プラントル数 Pr とシュミット数 Sc はここで名前付き定数として共に 1 に定義してあるので、条件を変えたい場合はここを変更するとよい。

4.9 store/therm.lib

therm.lib 内には必ず thermal_mode.f90 が入っており、set_thermo_prop というサブルーチンにより、前ステップに計算された

圧力、気体定数、保存量

から

温度の推定値、質量あたり内部エネルギー、各気体グループの質量分率

を計算し、そこから現ステップでの

圧力、比熱比、質量あたり全エンタルピー、粘性係数、気体定数、各化学種のエンタルピー vhi 、各化学種の修正生成熱 DHi

を計算する。また、viscosity の節でも述べたとおり、一部熱力学ライブラリではエンタルピー拡散の計算のために Yv も計算する。(圧縮性流体ではエネルギー保存であるためエンタルピーは圧力依存となる。圧力は温度で変わり、温度は熱力学ライブラリで変わるため、全エンタルピーも基本量としている。) これら出力結果は基本量 w の配列に格納される。

化学ライブラリには大きくわけて理想気体,NASA データベースを用いるもの,chemkin ファイルを用いるものにわけられ、それぞれディレクトリ ideal,NASA,chemkin に対応する。NASA データベースを用いる計算は凍結流、一段総括反応、完全平衡モデルがつかえ、chemkin ファイルを用いる計算は凍結流、素反応モデルが使える。

vhi と DHi の定義 それぞれのライブラリ詳細の説明を行う前に、 vhi と DHi の説明をする。

vhi はエンタルピー拡散の計算に使われ、次元は $nV(Yv$ の次元と同じ。 Yv を使わない場合は nY) である。それぞれの気体グループの生成エネルギーを含む単位質量あたりの内部エンタルピーである。拡散の計算に使われるため、 vhi に関しては境界での値が必要となる。よって、適切な境界条件を設定しなければならない。

DHi は修正生成熱であり、陰解法や前処理法に用いる流束の保存量でのヤコビアンを求めるとき出てくる。定義としては以下のとおり。

$$DHi_i = \kappa R_i T - (\kappa - 1)vhi_i \quad (2)$$

ただし、 κ は混合ガス全体の平均比熱比、 R_i はその気体グループの質量あたりの気体定数 ($=(\text{一般気体定数})/(\text{平均分子量})$).

DHi_i は理想気体では 0 であるが、実在気体では有限の値を持つ。また、陰解法を許さない熱力学モデルでは、これは 0 にセットされている。

4.9.1 store/therm.lib/ideal

特有モジュール	nY	nV	with- nV or no- nV
gas	1	1	no- nV

使用可能時間ステップ	global, local
使用可能時間スキーム	euler, RK2, LU-SGS, NR, precon, dual, preconLU-SGS

理想気体。ファイルは thermal_model.f90 のみである。粘性係数は動粘性係数一定の場合にのみ現在対応している。質量あたり気体定数 R_gas、比熱比 kappa_gas、動粘性係数 nu_gas は”module gas”をインポートすれば使用可能であり、checkout.py でそれぞれ 1.4, 287, 1.6e-5 に定義されるが、thermal_model.f90 の該当部分を直接編集すれば任意の値に変更可能である。境界条件設定時にはぜひ”module gas”をインポートして上記の名前付き定数を使っていただきたい。

保存量、基本量や vhi は以下のように表される。ただし、 ρ, u, v, p は密度、x,y 方向速度、圧力である。

```

q(1)=rho
q(2)=rho*u
q(3)=rho*v
q(4)=p/(kappa_gas-1d0)

w(1)=rho
w(2)=u
w(3)=v
w(4)=p
w(5)=1d0
w(indxg )=kappa_gas
w(indxht)=kappa_gas/(kappa_gas-1d0)*p/rho+0.5d0*(u**2+v**2)
w(indxR )=R_gas
w(indxMu)=rho*nu_gas

vhi(1)=kappa_gas/(kappa_gas-1d0)*p/rho
DH(1)=0d0

```

4.9.2 store/therm_lib/NASA

特有モジュール	nY	nV	with-nV or no-nV
const_chem, chem, chem_var	2	モデルによる	with-nV

このモデルを選択する際には chem.inp が存在しなければならない。

NASA-CEA 用に公開されたデータベースを用いた熱力学モデル集。凍結流と一段総括反応 (flame sheet)、完全平衡モデルを用意している。

気体グループは 2 つであり、ライブラリでは一つ目に燃料、二つ目に酸化剤を当てている。

全てのモデルで使用される熱力学モデルは core 内に格納されており、以下のファイルがある。

- LU.f90...LU 分解で線形連立方程式を解くモジュール。完全平衡モデルで利用されている。
- thermo.inp...NASA-CEA 内に格納されている熱力学データ。
- trans.inp...NASA-CEA 内に格納されている熱輸送データ。

- func_chem.f90... 热力学関数多項式の係数を返す関数ライブラリ.
- mod_chem.raw.f90...NASA データベースを用いた計算に必要なモジュール.(上述特有モジュール) 変数後述.
- sub_chem.f90...mod_chem 内の変数初期化や input ファイル読み込み、上述反応モデルの全てのコア部のサブルーチン.

mod_chem.f90 の変数について

const_chem 定数と重要な値 (化学種数) を格納している.

- ne... 原子種数
- max_ns... 最大化学種数
- Ru... 一般気体定数 (prmtr.f90 と同じ値ではあるが、化学ライブラリのみで 0 次元を行うときのために定義している。)
- pst=1d5... 標準状態の圧力
- omega=0.5d0... 一段総括反応モデルで用いられる緩和係数
- eps=1d-9... 収束計算に用いる微小量. 基準値がこれ以下になればよい.
- initial_eps=1d-5... 完全平衡モデルで係数行列の行列式が 0 にならないようにそれぞれの化学種量初期値に加える微小量
- Y_eps=1d-6... 完全平衡モデルで使用。燃料または酸化剤の質量分率がこれ以下の場合、存在しえない原子種に関する拘束条件を計算から排除する.
- TSIZE=1d-11... 完全平衡モデルで使用。E と H を計算する際、物質量がこれ以下の場合その分子がもつエネルギー やエンタルピーは計算しない。
- TTSIZE=1d-14... 完全平衡モデルで使用。各原子の量に関する拘束条件を計算するとき、物質量がこれ以下の分子は計算にいれない。
- ns... 化学種数.
- nt... 粘性係数の計算に使われる化学種数。化学組成の計算をする際に使われる化学種の内、trans.inp に含まれる化学種の数となる。

chem 計算のコアとなるデータ.

- num_sctn... それぞれの化学種の熱力学関数多項式の区間の数. 最大値 6.
- MW... それぞれの化学種の分子量
- Trange... それぞれの区間の始めとおわりの温度. 例えば化学種 i の j 番目の区間は Trange(1,j,i)K から Trange(2,j,i)K まで.
- co... それぞれの区間の熱力学関数の多項式の係数. 例えば化学種 i の j 番目の区間の多項式の係数は co(1:9,j,i).
- SYM_ELM... それぞれの原子種の記号。(HE や H₂O など)
- SYM_SPC... それぞれの化学種の記号。(CO₂ や H₂O など)
- trans... 輸送係数の多項式の係数。熱力学関数の co に対応。
- Trange_trans... 輸送係数の区間の端の値。熱力学関数の Trange に対応。
- species_name_trans... 热力学関数の SYM_SPC に対応

- num_sctn_trans... 热力学関数の num_sctn に対応
- tr2th... 輸送係数のインデックスを熱力学関数のインデックスの変換するもの。例えば H₂O のインデックスが輸送係数では i、熱力学係数が j(このとき species_name_trans(i) と SYM_SPC(j) は共に'H₂O'である。)の場合、tr2th(i)=j である。
- Ac... それぞれの化学種に含まれる原子の数。例えば原子 H,C,O のインデックスがそれぞれ 1,2,3、分子 H₂O のインデックスが 1 の場合、Ac(1,1)=2,Ac(2,1)=0,Ac(3,1)=1 である。

chem_var 境界条件で使うための燃料、酸化剤のそれぞれの各種値、及びそれぞれのセルの前ステップでの化学組成の記録。

- n_save... 完全平衡モデルで使用。各セルの前ステップでの組成。これを初期値に使うことで、反復回数が減る。
- qf,wf... それぞれ燃料の速度が 0 のときの保存量、基本量。
- rhof,pf,Tf,Ef,Hf,MWf,kappaf,muf... それぞれ燃料の密度、圧力、温度、質量あたり内部エネルギー、質量あたり内部エンタルピー、分子量、比熱比、粘性係数
- Yvf,vhif... それぞれ燃料の Yf,vhi
- b0f... 完全平衡モデルで使用。燃料の単位質量あたりそれぞれの原子物質量。
- nf... 完全平衡モデルで使用。燃料単体に完全平衡モデルを用いたときの組成
- nfini... 燃料単体の反応前の組成
- nef... 完全平衡モデルで使用。燃料に含まれる原子種数。
- elistf... 完全平衡モデルで使用。燃料に含まれる原子種のインデックス。
- nelistf... 完全平衡モデルで使用。燃料に含まれない原子種のインデックス。
- maskf... 完全平衡モデルで使用。各化学種で、存在しうるもののが 1、しえないものに 0 が入っている。
- maskbf... 完全平衡モデルで使用。拘束条件で、使用するものに 1、しないものに 0 が入っている。
- np... 一段総括反応モデルで使用。生成物の組成。
- of... 当量での o/f

また、上記には燃料のみ記してあるが、"f"を"o"に変えれば酸化剤のそれを表すことになる。

境界条件での入力について 境界条件での入力について。例えば酸化剤のみのセルで条件を入力する場合、保存量、基本量や vhi は以下のように表される。ただし、u,v は x,y 方向速度である。DH_iについては set_thermo_prop で直接入力されるため、手作業で入力されることはない。

```

q(1)=0d0
q(2)=rhoo
q(3)=rhoo*u
q(4)=rhoo*v
q(5)=rhoo*Eo

```

```

w(1)=rhoo
w(2)=u
w(3)=v

```

```
w(4)=po
w(5)=0d0
w(6)=1d0
w(indxg )=kappa_o
w(indxht)=Ho+0.5d0*(u**2+v**2)
w(indxR )=R_uni/MWo
w(indxMu)=mu_o
```

vhi=vhio

燃料では”o”を”f”に変える他、q(1)=rhof ; q(2)=0d0 ; w(5)=1d0 ; w(6)=0d0 になる。

以下、凍結流と一段総括反応 (flame sheet)、完全平衡モデルについてそれぞれ説明する。

凍結流モデル

使用可能時間ステップ	global, local
使用可能時間スキーム	euler, RK2, LU-SGS, NR, precon, dual, preconLU-SGS

凍結流モデルでは、酸化剤と燃料の混合は扱うが、その反応は考えない。つまり、酸化剤と燃料は混合されたところで反応しない。よって凍結流となる。凍結流モデルには、GUI.py で一段総括反応モデルを選ぶと得られる chem.inp と control_chem.raw.inp を用いる。control_chem.raw.inp は適切な修正をした後に名前を control_chem.inp とし、chem.inp と共に実行ファイルがあるディレクトリ直下に入れなければならない。control_chem.inp の中身は以下のようになっている。

```
#Control parameters for chemistry
Oxidizer Pressure(Pa)      : 1.e5
Oxidizer Temperature(K)    : 300.
Oxygen Composition(mole ratio):
N2 4
O2 1
end
Fuel Pressure(Pa)          : 1.e5
Fuel Temperature(K)        : 300.
Fuel Composition(mole ratio):
H2 1
end
Products Composition(mole ratio):
H2O 2
end
```

”Products Composition”以下の行は一段総括反応のときのみ使用するのでここでは説明しない。凍結流モデルにこのインプットファイルを使った場合、これらは無視される。

酸化剤と燃料はそれぞれの圧力と温度、そのモル比での組成を入力する必要がある。圧力と温度はそれぞれ単位 Pa, K で入力する。このインプットファイルではそれぞれ 1.e5, 300. となっている。

組成は”化学種名 モル分率”の順で一化学種に一行用いて入力し、最後の行には”end”のみの行を入力する。モル分率の和が 1 である必要はない。このインプットファイルでは、酸化剤に N2:O2=4:1 の混合気体(空気)、燃料に H2 純粋气体を用いることとなっている。

ファイルは thermal_model.f90 のみである。メインの計算は sub_chem.f90 内の cold_flow が行うこととなる。また、nV=2 である。

一段総括反応モデル

使用可能時間ステップ	global, local
使用可能時間スキーム	euler, RK2, LU-SGS, NR, precon, dual, preconLU-SGS

store/therm.lib/NASA/flame_sheet 内に格納されており、flow, mono, plot がある。mono と plot は 0 次元計算用のライブラリであり、流体計算には flow のみが使われる。

flow には自動生成ファイル用のファイル以外には thermal_model.f90 のみが存在する。これは実質的には sub_chem.f90 の flame_sheet を呼び出すためだけのルーチンとなり、メインの計算は flame_sheet が行うこととなる。

一段総括反応モデルでは組成は自由エネルギーが最小になるように決まる。一段総括反応モデルには、GUI.py で完全平衡モデルを選ぶと得られる chem.inp と control_chem.raw.inp を用いる。control_chem.raw.inp は適切な修正をした後に名前を control_chem.inp とし、chem.inp と共に実行ファイルがあるディレクトリ直下に入れなければならない。control_chem.inp の中身は例えば以下のようにになっている。

```
#Control parameters for chemistry
Oxidizer Pressure(Pa)      : 1.e5
Oxidizer Temperature(K)    : 300.
Oxygen Composition(mole ratio):
N2 4
O2 1
end
Fuel Pressure(Pa)          : 1.e5
Fuel Temperature(K)        : 300.
Fuel Composition(mole ratio):
H2 2
end
Products Composition(mole ratio):
N2 4
H2O 2
end
```

酸化剤と燃料はそれぞれの圧力と温度、そのモル比での組成を入力する必要がある。圧力と温度はそれぞれ単位 Pa,K で入力する。このインプットファイルではそれぞれ 1.e5, 300. となっている。生成物はそのモル比での組成のみ入力する。

組成は”化学種名 モル分率”の順で一化学種に一行用いて入力し、最後の行には”end”のみの行を入力する。このインプットファイルでは、酸化剤に N2:O2=4:1 の混合気体(空気)、燃料に H2 純粋気体を用いることになっている。モル分率の和が 1 である必要はないが、酸化剤、燃料、生成物の量の分母は一致させなければならない。つまり、このファイルで生成物の組成を”N2 2”, ”H2O 1”としてはならない。このような入力を行うと、反応式の右辺と左辺の原子の数が釣り合わないことを感知し、以下のようなエラーメッセージを出す仕様となっている。

```
ERROR: The amount of element: H
ERROR: does not balance between reactants and products.
ERROR: Please modify 'control_chem.inp'.
```

できるだけ化学反応式のそれぞれの化学種の係数をそのまま入力すること。

また境界条件において、燃料だけ、酸化剤だけではなく、その間、または入力した以外の圧力、温度を持つ組成を欲しい場合がある。その際は、thermal_model.f90 にある YPT2w(Yf,p,T,wt,vhit)(入力=Yf:燃料質量分率,p:圧力,T:温度)(出力=wt:速度 0 での基本量 w,vhit:vhi) を使う。これ以外の境界条件が欲しい場合(ほとんどの研究ではこのような場合に出くわす)は自分で作成すること。

また、nV=3 であり、Yv(1),Yv(2),Yv(3) にはそれぞれ反応後の燃料、酸化剤、生成物の質量分率が入る。

完全平衡モデル

使用可能時間ステップ	global, local
使用可能時間スキーム	euler, RK2, LU-SGS, NR, precon, dual, preconLU-SGS

store/therm.lib/NASA/cea 内に格納されており、このディレクトリには flow, mono, plot, reduction がある。mono, plot, reduction は 0 次元計算用のライブラリであり、流体計算には flow のみが使われる。

flow には自動生成ファイル用のファイル以外には thermal_model.f90 のみが存在する。これは実質的には sub_chem.f90 の cea を呼び出すためだけのルーチンとなり、メインの計算は cea が行うこととなる。

完全平衡モデルでは酸化剤と燃料はどちらかがなくなるまで反応を行う。組成は酸化剤と燃料の相対的な量で決まる。完全平衡モデルには、GUI.py で一段総括反応モデル (flame sheet model) を選ぶと得られる chem.inp と control_chem.raw.inp を用いる。control_chem.raw.inp は適切な修正をした後に名前を control_chem.inp とし、chem.inp と共に実行ファイルがあるディレクトリ直下に入れなければならない。control_chem.inp の中身は例えば以下のようにになっている。

```
#Control parameters for chemistry
Oxidizer Pressure(Pa) : 1.e5
Oxidizer Temperature(K) : 300.
Oxygen Composition(mole ratio):
N2 4
O2 1
end
```

```

Fuel Pressure(Pa)      : 1.e5
Fuel Temperature(K)    : 300.
Fuel Composition(mole ratio):
H2 2
end
Products Composition(mole ratio):
N2 4
H2O 2

```

酸化剤と燃料はそれぞれの圧力と温度、そのモル比での組成を入力する必要がある。圧力と温度はそれぞれ単位 Pa,K で入力する。このインプットファイルではそれぞれ 1.e5, 300. となっている。また、一段総括反応モデルと同様に、生成物の組成も定義する必要がある。純粋な完全平衡モデルではこの条件は必要ではない。生成物の組成は自由エネルギー最小という条件から求まるためである。本プログラムでこの量を定義しているのは、必要以上に計算負荷があがることを防ぐために、ほとんど酸化剤の場合やほとんど燃料の場合には一段総括反応を使用することにしているためである。この”ほとんど”を定義するために、許容温度誤差 (K) と切り替えパラメータをチェックする周期を control.inp に定義する必要がある。完全平衡モデルを用いる際には、以下を control.inp で定義する必要がある。

```

#FOR CFH
Temp. Diff Limit(K)   : 1.e-3
cfh check freq.       : 500

```

cfh とは”cea-flame sheet hybrid scheme”の略で、”cea”は完全平衡モデル、”flame sheet”は一段総括反応モデルを表している。この入力例では 500 ステップ毎にどのくらい”ほとんど”燃料または酸素であれば温度の誤差が 10^{-3} K 以下になるかを計算し、他のステップではその閾値を用いて完全平衡モデルと一段総括反応の切り替えを行っている。

組成は”化学種名 モル分率”の順で一化学種に一行用いて入力し、最後の行には”end”のみの行を入力する。このインプットファイルでは、酸化剤に N2:O2=4:1 の混合気体(空気)、燃料に H2 純粋気体を用いることになっている。

計算のアルゴリズムには基本的に NASA-CEA と同じものが使われる。特に、緩和係数は一段総括反応モデルのような固定ではなく、様々なパラメータの増減をみた値となっている。また、ほぼ燃料または酸化剤の場合、係数行列に逆行列がなくなる場合がある。これは、存在しない原子の量に関する拘束条件が、収束近くになると係数がすべて 0 になるからである。よって、これらのときにその拘束条件を削除するアルゴリズムを実装しており、その際、nef,elistf,nelistf,maskf,maskbf などの配列が使われる。また、複数の原子がある同一の化学種のみに含まれるような平衡状態が解の場合、それらの原子の量に関する拘束条件が数学的に同値となり、係数行列のランクが落ちる。この場合、NASA CEA では特殊な処理が行われるらしいが、本プログラムではこのような状態をさけるため、拘束条件に 10^{-30} 程度の微小量を加えている。この数は使用するモデルによって変更せねばならない可能性がある。

また、nV=ns であり、Yv にはそれぞれの化学種の質量分率が入る。

また、chem.inp には使用する原子種と化学種のリストが入る。よって、この化学種のリストを変更すれば、簡略化モデルとなる。この簡略化については 0 次元化学計算の章で言及する。

4.9.3 store/therm_lib/chemkin

特有モジュール	nY	nV	with-nV or no-nV
const_chem, chem, chem_var	化学種数	nY	no-nV

このモデルを選択する際には chem.inp が存在しなければならない。

”thermo all”を用いた thermo.inp を必要としない形式の chemkin.inp の使用を前提とした熱力学ライブラリである。凍結流と素反応モデルを用意している。

気体グループは一つ一つが単一の化学種に対応する。つまり、 ρ_i の次元は化学種数となる。

全てのモデルで使用される熱力学モデルは core 内に格納されており、以下のファイルがある。

- MW.inp... それぞれの原子の名前と原子量の一覧
- trans.inp...NASA-CEA 内に格納されている熱輸送データ.
- dvode.f...odepack のパッケージの一つである dvode.f に修正を加えたもの
- mod_chem.raw.f90...chem.inp を用いた計算に必要なモジュール.(上述特有モジュール) 変数後述.
- sub_chem.f90...mod_chem 内の変数初期化や input ファイル読み込み、上述反応モデルの全てのコア部のサブルーチン. NASA データベースでの func_chem.f90 の内容も統一して格納している.

なお、control_chem.inp について、NASA データベースを用いた場合にはモデル作成時に生成するようになっているが、chemkin.inp の場合、テンプレートを”control_chem.raw.inp”を他のファイルと同じ場所に生成するようにしている。

dvode.f に加えた修正について dvode.f は OpenMP に直接用いるとエラーを起こす。これは、common 文に含まれる変数がプロセッサ間で共有されてしまうからである。そのため、本ライブラリに含む dvode.f には threadprivate 属性を付け加えている。

また、警告メッセージを標準出力に出力すると標準出力がそれで埋め尽くされてしまうので、標準出力への警告出力も抑制してある。

mod_chem.f90 の変数について

const_chem 定数と重要な値 (化学種数) を格納している.

- erg2cal=1d0/4.184e7...erg から cal への単位変換に用いる定数.erg の値にかけると cal になる
- cal2erg=1d0/erg2cal...cal から erg への単位変換に用いる定数。
- b2Pa=0.1d0...barye から Pa への単位変換に用いる定数。
- Pa2b=1d0/b2Pa...Pa から barye への単位変換に用いる定数。
- Ru=8.3144621d7...cgs 単位系での気体定数。
- invRc=1d0/(1.987e0)...cal/(mol*K) で表した気体定数の逆数. 活性化エネルギーに対して用いる。
- pst=1.01325d6... 単位系 dyn/cm² での標準状態圧力
- logPR=-4.407418526701446...=ln(Pst/Ru)
- n_eps = 1d-40...n(mol/cm³) に用いる微小量. 初期値がこれ以下である場合、この値にして計算を回す。
- rtol_user,atol_user... それぞれユーザー定義の相対許容誤差、絶対許容誤差
- isColdFlow... 凍結流の計算で true、それ以外で false が設定される。true が設定されると、温度計算 反復で化学種を反応物に含まれるものしか考慮しなくなる。

- max_ne,ne... 最大原子種数及び原子種数
- max_ns,ns... 最大化学種数及び化学種数
- max_nr,nr... 最大素反応数及び素反応数
- num_reac... 反応物で使われている化学種数
- LRW=22+9*(max_ns+1)+2*(max_ns+1)**2...dvode 用に確保する実数型作業用配列の次元
- LIW=30+ (max_ns+1)...dvode 用に確保する整数型作業用配列の次元
- max_recalc=50...dvode でこの回数以上再計算を行う場合、計算を止める。(一回の計算で 500 回の内部反復を dvode は行うので、500*50=25,000 回の内部反復となる。)

chem 計算のコアとなるデータ.

- nt... 粘性係数の計算に使われる化学種数。化学組成の計算をする際に使われる化学種の内、trans.inp に含まれる化学種の数となる。
- ns_tocalc...isColdFlow=.true. の場合 num_reac, isColdFlow=.false. の場合 ns。
- SYM_ELM... それぞれの原子種の記号。(HE や H,O など)
- SYM_SPC... それぞれの化学種の記号。(CO2 や H2O など)
- SYM_RCT... それぞれの素反応の記号。(‘CH2CO+H=CH3+CO’ など)
- ES... それぞれの化学種に含まれる原子の数. 例えば原子 H,C,O のインデックスがそれぞれ 1,2,3、分子 H2O のインデックスが 1 の場合、ES(1,1)=2,ES(2,1)=0,ES(3,1)=1 である。
- Tthre... それぞれの区間の始めと終わりの温度. 例えば化学種 i の j 番目の区間は Tthre(1,j,i)K から Tthre(2,j,i)K まで.
- coeff... それぞれの区間の熱力学関数の多項式の係数. 例えば化学種 i の j 番目の区間の多項式の係数は coeff(1:7,j,i)。
- MWs... それぞれの化学種の分子量
- invMW...MWs の逆数
- Rstate... 素反応の分類
 - Rstate(1,:)... 双方向反応か一方向反応か
 - * Rstate(1,:)=0: 双方向反応 (‘=’ や ‘<=>’ など)
 - * Rstate(1,:)=1: 一方向反応 (‘=>’ で表される反応)
 - Rstate(2,:)... 反応係数のモデル
 - * Rstate(2,:)=0: 逆反応を平衡係数から求める.
 - * Rstate(2,:)=1: 双方向の反応係数の係数がそれぞれ存在する (rev)
 - * Rstate(2,:)=2:Lindemann 式 (low)
 - * Rstate(2,:)=3:TROE 式 (troe)
 - * Rstate(2,:)=4: 第三体が必ず存在する反応 (第三体に括弧がつかない反応) で逆反応を平衡係数から求める.
 - * Rstate(2,:)=5: 第三体が必ず存在する反応 (第三体に括弧がつかない反応) で双方向の反応係数の係数がそれぞれ存在する.
- NumNu... 右辺左辺それぞれの反応体数 (素反応 j が 2H2O2=2H2O+O2 なら NumNu(1,j)=2, NumNu(1,j)=3)
- IndNu... 右辺左辺それぞれの反応体のインデックス (素反応 j が 2H2O2=2H2O+O2 で H2O2,H2O,O2

のインデックスがそれぞれ 1,2,3 なら、IndNu(1,1,j)=1; IndNu(2,1,j)=1 ;IndNu(1,2,j)=2; IndNu(2,2,j)=2; IndNu(3,2,j)=3)

- snu...snu(j)=NumNu(2,j)-NumNu(1,j)
- duplicate...chem.inp で duplicate 属性がついているなら .true.、ついていないなら .false.。
- exist_M... 第三体を含む反応なら true、含まないなら false
- NumM...enhance factor が設定されている場合、それを持つ化学種数 (H₂O/2 / O₂/3/なら 2)
- IndM...enhance factor をもつ化学種のインデックス (素反応 k の enhance factor が H₂O/2 / O₂/3/かつ H₂O,O₂ のインデックスがそれぞれ 2,3 なら、IndM(1,k)=2;IndM(2,k)=3)
- Men...enhance factor から 1 を引いたもの (素反応 k の enhance factor が H₂O/2 / O₂/3/なら、Men(1,k)=1;Men(2,k)=2)
- ABE... 反応係数 $k = AT^\beta \exp(-E/RT)$ のとき、ABE(1)=ln A, ABE(2)= β , ABE(3)=E/R.
- cABE...rev や low の ABE
- TROE...troe フォームの係数. TROE(1)=a, TROE(2)=1/T***, TROE(3)=1/T*, TROE(4)=T**
- species_name_trans... 热力学関数の SYM_SPC に対応
- Trange_trans... 輸送係数の区間の端の値。热力学関数の Tthre に対応。
- trans... 輸送係数の多項式の係数。热力学関数の coeff に対応。
- num_sctn_trans... それぞれの化学種の輸送係数関数多項式の区間の数。
- tr2th... 輸送係数のインデックスを热力学関数のインデックスの変換するもの。例えば H₂O のインデックスが輸送係数では i、热力学係数が j(このとき species_name_trans(i) と SYM_SPC(j) は共に'H₂O'である。)の場合、tr2th(i)=j である。

chem_var 境界条件で使うための燃料、酸化剤のそれぞれの各種値。

- qf,wf... それぞれ燃料の速度が 0 のときの保存量、基本量。
- rhof,pf,Tf,Ef,Hf,MWf,kappaf,muf... それぞれ燃料の密度、圧力、温度、質量あたり内部エネルギー、質量あたり内部エンタルピー、分子量、比熱比、粘性係数
- vrhof...qf(1:ns)
- vwf...wf(5:ns+4)
- vhif... 燃料の vhi
- of... 当量での o/f.0 次元計算のときのみ使用。

また、上記には燃料のみ記してあるが、”f”を”o”に変えれば酸化剤のそれを表すことになる。

uvhp について store/therm.lib/chemkin には uvhp というディレクトリがあり、uv と hp というディレクトリがその中に含まれている uv と hp というディレクトリには FJ.f90 が含まれており、それぞれ uv 一定, hp 一定条件のサブルーチン群が含まれている。圧縮性流体の場合、保存量が uv 一定であるため、自動的に uv ディレクトリが使用される設定になっている。

境界条件での入力について 境界条件での入力に関して。例えば酸化剤のみのセルで条件を入力する場合、保存量、基本量や vhi は以下のように表される。ただし、u, v は x,y 方向速度である。DH_i については set_thermo_prop で直接入力されるため、手作業で入力されることはない。

q(:)=qo

```

q(nY+1)=rho*u
q(nY+2)=rho*v
q(nY+3)=qo(nY+3)+0.5d0*rho*(u**2+v**2)

```

```

w(:)=wo
w(2)=u
w(3)=v
w(indxht)=wo(indxht)+0.5d0*(u**2+v**2)
vhi=vhio

```

燃料では”o”を”f”に変える。

以下、凍結流と素反応モデルについてそれぞれ説明する。

凍結流モデル

使用可能時間ステップ	global, local
使用可能時間スキーム	euler, RK2, LU-SGS, NR, precon, dual, preconLU-SGS

store/therm.lib/chemkin/flow/cold に格納されている。

凍結流モデルでは、酸化剤と燃料の混合は扱うが、その反応は考えない。つまり、酸化剤と燃料は混合されたところで反応しない。よって凍結流となる。凍結流モデルを選んだ流体コードを生成すると、checkout ディレクトリに control_chem.raw.inp が生成される。これは適切な修正をした後に名前を control_chem.inp とし、chem.inp と共に実行ファイルがあるディレクトリ直下に入れなければならない。control_chem.inp の中身は以下のようになっている。

```

#Control parameters for chemistry
Oxidizer Pressure(Pa) : 1.e5
Oxidizer Temperature(K) : 1200.
Oxygen Composition(mole ratio):
O2 2
end
Fuel Pressure(Pa) : 1.e5
Fuel Temperature(K) : 1200.
Fuel Composition(mole ratio):
CH4 1
end
Relative Tolerance VODE : 1.e-8
Absolute Tolerance VODE : 1.e-20

```

酸化剤と燃料はそれぞれの圧力と温度、そのモル比での組成を入力する必要がある。圧力と温度はそれぞれ単位 Pa,K で入力する。このインプットファイルではそれぞれ 1.e5, 1200. となっている。

組成は”化学種名 モル分率”の順で一化学種に一行用いて入力し、最後の行には”end”のみの行を入力する。モル分率の和が 1 である必要はない。このインプットファイルでは、酸化剤に O₂ 純粹気体、燃料に CH₄ 純粹気体を用いることとなっている。

また、VODE に用いる相対許容誤差と絶対許容誤差も定義しなければならない。(凍結流では実際にこの値は使われることはない。) 例に載せてある値は SENKIN に使われている値である。これらの値は大きくするとそれだけ計算が早くなるが、大きくしすぎると計算が破綻する(発散する)。

境界条件には calc_boundary(p,T,Yf, wt,vhit) を用いる。仕様は NASA データベースで用いられている YPT2w(Yf,p,T,wt,vhit) と同様である。

また凍結流の計算では、化学種は反応物に含まれるものしか考慮されない。

素反応モデル

使用可能時間ステップ	global
使用可能時間スキーム	euler, RK2

store/therm.lib/chemkin/flow/reactive に格納されている。現段階で唯一 point implicit を用いる熱力学ライブラリである。

素反応モデルでは、化学種の組成は素反応を直接積分して求める。そのため、最も計算コストが高いが、最も詳細な、熱力学モデルとなる。素反応モデルを選んだ流体コードを生成すると、checkout ディレクトリに control_chem.raw.inp が生成される。これは適切な修正をした後に名前を control_chem.inp とし、chem.inp と共に実行ファイルがあるディレクトリ直下に入れなければならない。control_chem.inp の中身は以下のようにになっている。

```
#Control parameters for chemistry
Oxidizer Pressure(Pa)      : 1.e5
Oxidizer Temperature(K)    : 1200.
Oxygen Composition(mole ratio):
O2 2
end
Fuel Pressure(Pa)          : 1.e5
Fuel Temperature(K)        : 1200.
Fuel Composition(mole ratio):
CH4 1
end
Relative Tolerance VODE : 1.e-8
Absolute Tolerance VODE : 1.e-20
```

酸化剤と燃料はそれぞれの圧力と温度、そのモル比での組成を入力する必要がある。圧力と温度はそれぞれ単位 Pa,K で入力する。このインプットファイルではそれぞれ 1.e5, 1200. となっている。

組成は”化学種名 モル分率”の順で一化学種に一行用いて入力し、最後の行には”end”のみの行を入力する。モル分率の和が 1 である必要はない。このインプットファイルでは、酸化剤に O₂ 純粹気体、燃料に CH₄ 純粹気体を用いることとなっている。

また、VODE に用いる相対許容誤差と絶対許容誤差も定義しなければならない。例に載せてある値は SENKIN に使われている値である。これらの値は大きくするとそれだけ計算が早くなるが、大きくしそぎると計算が破綻する（発散する）。

境界条件には calc_boundary($p, T, Y_f, wt, vhit$) を用いる。仕様は NASA データベースで用いられている YPT2w($Y_f, p, T, wt, vhit$) と同様である。

素反応モデルでは、時間積分ライブラリとして chemeq2 と dvode が用いられる。これらの切り替えは、chemeq2 で 10 回以上の内部反復が行われると、dvode が呼び出されることとなっている。これらの時間積分の前後で、 n_{eps} 以下の化学種の量はすべて n_{eps} に置き換えられる。

4.10 store/cond

condition.raw.f90 の生成を行う。

condition.f90 では、初期条件と境界条件の設定を行うこととなっているが、境界条件については、condition.raw.f90 生成時に境界内にある切断面を探索し、切断面については入力欄をなくしており、手動ではなく自動的に境界値が設定されるようになっている。

condition.raw.f90 生成時には以下のような動作を行っている。

4.10.1 store/cond/core

- mod_cond.py... 以下のプログラムで用いる class の定義。
- gen_cond.py... 主プログラムであり、checkout.py から呼び出しを受ける。以下のファイルに含まれる関数をドライブする。
- read_plot3d.py...plot3d ファイル (*.x という名前を持つグリッドファイル) を読み込み、各ブロックの境界の座標をリスト化する。
- set_cond_var.py... 各ブロックの境界のうち、他のブロックと接触している部分といない部分、つまりユーザーが境界条件を定義する必要がない部分とある部分をそれぞれリスト化する。
- fetchMPI.py...grid_separation.inp からグリッドのプロセッサへの分割線を計算し、ブロック間での境界条件の通信でどのプロセッサがどのプロセッサにどこのデータを送るべきかを計算する。
- out_cond.py... 上述計算結果の condition.raw.f90, cut_copro*.inp, MPIcomm*.inp へのアウトプット。

4.10.2 store/cond/no-nV, store/cond/with-nV

それぞれ no-nV, with-nV に対応する。with-nV では Y_v の MPI 転送が必要となるためファイルが異なる。condition.head.f90 と condition.tail.f90 という condition.raw.f90 の head と tail が含まれている。

5 NASA 熱力学データベース

checkout_model.inp を記入し、checkout.py を起動すると生成される。実処理は checkout.py から呼び出された”store/checkout/checkout_model.py”によって生成される。ファイルは chem.inp と control_chem.raw.inp が checkout.py と同じディレクトリに生成される。これらのデータベースを用いる計算ではこの control_chem.raw.inp を編集し、control_chem.inp として作業ディレクトリ (checkout または checkout_chem) にコピーしなければならない。編集方法は 4.9.2 に記してある。

checkout_model.inp は以下のようになっている。

```
thermal model:0 #flame sheet of nasa(0) / NASA CEA(1)
Species Selection:
CH4 02
end
```

thermal model の部分で一段総括反応 (0) または完全平衡モデル (1) を選ぶことができる。Species Selection では一行に一つの化学種を記録する (スペース空けて一行に複数の化学種を記入してもよい) 最後には’end’ の行を記入する。

なお、GUI.py でも checkout_model.inp 生成及び checkout.py 実行ができるが、この場合、”Not Selected” の窓内の選択肢を選択し、Ctrl+F を押すと入力窓が立ち上がり、そこに化学種名を入力すると該当する化学種が検索される。選択のとき便利なので利用するとよい。

一段総括反応モデル 選択された化学種のみが chem.inp に記録され、また control_chem.raw.inp には”Production Composition”の欄が生成される。

完全平衡モデル 選択された化学種が含む原子を含む全ての化学種が chem.inp に記録され、また control_chem.raw.inp には”Production Compsition”の欄は生成されない。

6 0次元化学コード

checkout_chem.inp を記入し、checkout.py を起動すると生成される。実処理は checkout.py から呼び出された”store/checkout/checkout_chem_nasa.py”によって生成される。(現在は nasa のみではなく、chemkin 用のファイルもこの python スクリプトで生成できる。) ファイルはディレクトリ checkout_chem に生成される。実行には chem.inp が必要である。

化学コードでは化学モデルの選択や、uv 一定条件か hp 一定条件の選択、単一条件か複数条件の選択、更には化学モデルの簡略化までを行うことができる。

コード生成には checkout_chem.inp を用いる。中身は以下のようになっている。

```
thermal model:1 # flame sheet of nasa(0) / NASA CEA(1) / chemkin(2)
calc model   :2 # mono(0) / plot(1) / reduction(2)
constants    :1 # uv(0) / hp(1)
```

flame sheet of nasa, NASA CEA, chemkin はそれぞれ一段総括反応、完全平衡モデル、化学素反応モデルを表している。該当する数字を入力すれば所望のコードを得られる。例えばこの入力ファイルでは、圧力エンタリピー一定完全平衡モデルの簡略化コードを得られる。

実行ファイルは driver であり、./driver と入力することで、実行される。

化学コードはディレクトリ構造が複雑である。

NASA データベースを用いるものでは共通の計算を行うファイルが

```
store/therm_lib/NASA/core
```

にまとめてあり、それぞれの計算の種類によって以下のような構造のディレクトリのいずれかを選ぶこととなる。

```
store/therm_lib/NASA/{flame_sheet,cea}/{mono,plot,reduction}/{uv,hp}
```

ただし{a,b,c}と書いてあるところには a か b か c が入る。後述するように全ての組み合わせが存在するわけではない。

chemkin ファイルを用いるものでは共通の計算を行うファイルが

```
store/therm_lib/chemkin/core
```

にまとめてあり、uv 一定条件か hp 一定条件かで

```
store/therm_lib/chemkin/uvhp/{uv,hp}
```

計算の種類によって

```
store/therm_lib/chemkin/{mono,plot,reduction}
```

を選ぶことになる。ただし{a,b,c}と書いてあるところには a か b か c が入る。これも後述するように全ての組み合わせが存在するわけではない。

また、chem_drive の中に、ドライバである driver.f90 をコンパイルするための Makefile の部品が存在し、

0次元化学コード生成時には必ず用いられる。流体コードの store/core にあたる。

また、化学素反応モデル (chemkin) では”平衡温度”と”着火時刻”を用いているが、定義はそれぞれ

- 着火時間...0次元自己着火計算において初期状態から初期温度から 400K あがるまでの時間
- 平衡温度...0次元自己着火計算において初期状態から着火時間の 10 倍経ったときの時間

である。

6.1 store/therm_lib/NASA

6.1.1 mono

単条件化学計算を行うものであり、出力としては

燃料のみ、酸化剤のみ、当量

の

圧力、密度、温度、内部エネルギーまたはエンタルピー、平均分子量、比熱比、粘性係数

と o/f 比を出力する。

control.inp は必要はないが、control_chem.inp で、そのままのモル分率で混合すると当量になるよう、燃料と酸化剤のそれぞれの化学種の量を設定しなければならない。

特有のファイルは”store/therm_lib/NASA/{flame_sheet,cea}/mono/{uv,hp}”の中に収められており

- Makefile.var
- driver.f90

のみである。driver.f90 は sub_chem.f90 のルーチンを呼び出し、画面に表示する役割をもつ。

6.1.2 plot

複数条件の 0 次元化学計算を行うものであり、出力としては各条件での平衡温度となる。

燃料と酸化剤の定義は control_chem.inp で流体計算と同様に行う。control.inp は次のような入力になる。

```
Pressure (Pa)          range and Ntics : 1.e5 1.e6  3
Temperature (K)        range and Ntics : 300. 300.  1
mass fraction of fuel range and Ntics :   0.   1. 11
variable for x axis           : Yf # P or T or Yf. Yf is mass fraction of fuel
```

入力は初期圧力、初期温度、燃料質量分率及びそのサンプル数、加えてプロット時に x 軸とする変数である。

実行が完了すると、”done”と表示される。

また、プロット時には、ある変数については色をわけ同じファイルに、ある変数についてはファイル自体をわけることにしており、ファイルを分ける変数は x 軸とする変数以外でサンプル数が少ない変数が自動的に選ばれる。

上記入力ファイルの場合、x 軸を燃料質量分率、y 軸に平衡温度、異なる初期圧力が色分けされて描かれて、

初期温度 300K を固定した一つのグラフが生成される。

データファイルとしては、plot.001.002.dat のようなファイルが生成される。一つ目の数字(例では'001')はファイルの違い(例では初期温度の違い)、二つ目の数字(例では'002')は色わけの違い(例では初期圧力の違い)を表す。例にあげた入力ファイルだと plot.001.001.dat, plot.001.002.dat, plot.001.003.dat が生成される。

gnuplot plot.plt というコマンドを実行すると out.*.eps というグラフが生成される。'*'には plot.*.*.dat の一つ目の数字が入る。

また、plot.*.*.dat には温度だけではなく各種物性値も記録される。どの列にどのデータがあるかはコメント文をデータファイルに直接記してあるので参照のこと。

特有のファイルは”store/therm.lib/NASA/{flame_sheet,cea}/plot/{uv,hp}”の中に収められており

- Makefile.main
- Makefile.var
- conditions.f90
- control.part.inp
- driver.f90

である。conditions.f90 では control.inp の読み込みや各条件に用いるエネルギーと密度の定義、plot.*.*.dat などの書き出しを行っている。

6.1.3 reduction

複数条件の 0 次元化学計算による簡略化を行う。モニターする値は温度のみであり、温度があっていれば、どのような簡略化でも許すようになっている。また、簡略化は化学種の削減により行われる。定圧定エンタルピー条件の完全平衡モデルの簡略化のみを行うことができる。つまり、一段総括反応モデルはそもそも簡略化を行う必要はなく、また体積エネルギー一定条件の完全平衡モデルにおける簡略化は未実装である。

燃料と酸化剤の定義は control_chem.inp で流体計算と同様に行う。control.inp は次のような入力になる。

```
Pressure (Pa)          range and Ntics : 1.e5 1.e6 7
Temperature (K)        range and Ntics : 300. 300. 1
mass fraction of fuel range and Ntics : 0. 1. 11
Allowable tolerance of temperature : 0.05
```

入力は初期圧力、初期温度、燃料質量分率及びそのサンプル数、許容する温度誤差(割合)である。上記例であれば、許容温度誤差 5% となる。

実行が完了すると、作成されたモデルの実際の温度誤差、及び使用される化学種数が表示される。

新しく作成されたモデルは chem.inp.new とファイルに保存される。これを chem.inp とすれば、流体計算や 0 次元化学計算に用いることができる。

簡略化は全条件において最も量が少なかった化学種から順に行われる。この量は”sort.dat”というファイルに記録されている。以下にファイル冒頭を抜粋する。

```
1 9.9999811E-01 # CH4
```

```

2 7.7786694E-01 # N2
3 2.2213306E-01 # O2
4 1.9663044E-01 # CO
5 9.0467837E-02 # H2O
6 8.5527640E-02 # CO2
7 3.1036181E-02 # H2
8 1.1552881E-04 # NH3
9 4.3948638E-05 # C2H6
10 2.1079976E-06 # HCN

```

一列目は化学種順位、二列目は最大の質量分率、三列目は#、四列目は化学種名となっている。化学種順位は反応物に使われているものが高く、次に量が多いものの順に並んでいる。よって最大質量分率が大きい生成物が、最大質量分率が小さい反応物より化学種順位が小さくなる場合もある。また、このファイルは化学種名がコメントとなっているので、以下のように gnuplot でプロットも可能である。

```

set logscale y
plot "sort.dat" with lines

```

特有のファイルは”store/therm_lib/NASA/cea/reduction/hp”の中に収められており

- Makefile.main
- Makefile.var
- conditions.f90
- control.part.inp
- driver.f90

である。conditions.f90 では control.inp の読み込みや各条件に用いるエネルギーと密度の定義、簡略化サブルーチンの定義を行っている。

6.2 store/therm_lib/chemkin

6.2.1 mono

単条件化学計算を行うものであり、当量における自己着火計算の温度と指定化学種の質量分率の時間履歴を出力する。初期温度、初期圧力、”当量”は control_chem.inp で指定されたモル分率として定義される。例えば control.inp が

```

#Control parameters for chemistry
Oxidizer Pressure(Pa)    : 1.01325e5
Oxidizer Temperature(K)  : 1300.
Oxygen Composition(mole ratio):
O2 2
end

```

```

Fuel Pressure(Pa)      : 1.01325e5
Fuel Temperature(K)    : 1300.
Fuel Composition(mole ratio):
CH4 1
end
Relative Torelance VODE : 1.e-8
Absolute Torelance VODE : 1.e-20

```

の場合は圧力 1 気圧、温度 1300K、CH₄:O₂=1:2 の計算が行われる。また、VODE で用いる相対許容誤差 (Relative Torelance VODE) と絶対許容誤差 (Absolute Torelance VODE) も定義しなければならない。(多くの場合上記デフォルト値で十分。落ちた場合はこれらを小さくとること。)

control.inp は以下のようなファイルである。

```

final time (s)          : -1.e0
time step (s)           : -1.e0
species to include:
CH4
O2
OH
CH
CO2
CO
H2O
end

```

final time, time step は自分で決めることもできるが、final time に負値を指定すると、final time が着火時間の 1.5 倍、time step が final time の千分の一に設定される。また、100 秒以内に着火しない場合は着火時間計算不能と判断され、プログラムは異常終了する。

”species to include” 以下には時間履歴を記録したい化学種を記す。一行一化学種であり、最後に”end”だけの行が必要である。

データは”plt.dat”に出力される。

```
gnuplot plot.plt
```

を実行すると、まず温度の時間履歴が表示される。グラフのウィンドウを閉じ ('q' 押下で閉じれる)、ターミナル上でもう一度 Enter キーを押すと、指定した化学種の質量分率の時間履歴が表示される。これも 'q' 押下で閉じれ、ターミナル上でもう一度 Enter キーを押すと、実行は終了する。

plt.dat における各値の列は plot.plt を参照のこと。

特有のファイルは”store/therm.lib/chemkin/mono”の中に収められており

- control.part.inp
- Makefile.var

- driver.f90

のみである。driver.f90 は sub_chem.f90 のルーチンを呼び出し、画面に表示する役割をもつ。

6.2.2 plot

複数条件の 0 次元化学計算を行うものであり、出力としては各条件での平衡温度及び着火時間となる。

燃料と酸化剤の組成定義および VODE のパラメータ定義は control_chem.inp で流体と同様に行う。

control.inp は次のような入力になる。

```
Pressure (Pa)          range and Ntics : 1.e6   1.e6   1
Temperature (K)        range and Ntics : 1300. 1500. 2
mass fraction of fuel range and Ntics : 0.15   0.25   3
variable for x axis      : T # P or T or Yf. Yf is mass fraction of fuel
```

入力は初期圧力、初期温度、燃料質量分率及びそのサンプル数、加えてプロット時に x 軸とする変数である。

着火計算の場合、必ずしも全ての条件で着火することはない。よって、まず上記条件の中で”計算可能”なものと”計算不可能”だったものが分類される。この分類は実行時に

1	1	1 OK
2	1	1 OK
1	1	2 OK
2	1	2 OK
1	1	3 OK
2	1	3 OK
(calculated points)/(all points)=		6 / 6

のように表示される（番号は左から順に初期温度、初期圧力、燃料質量分率）他、gnuplot で

```
splot "calculatable.out", "not_calculatable.out"
```

とコマンドをうつと、計算可能な点が赤、不可能な点が緑で表示される。

出力ファイルの、色わけする変数やファイルを分ける変数の選び方や、plot.*.*.dat の名前規則は NASA データベースを用いる計算と同じである。ただし、gnuplot plot.plt を実行した際には、out.*.eps の代わりに Teq.*.eps と Tign.*.eps が出力される。'*' には NASA データベースの際と同じ数字が入る。

また、Tref.out には、全条件での初期圧力、燃料質量分率、初期温度、着火時刻、平衡温度がこの順で入力されている。

特有のファイルは”store/therm.lib/chemkin/plot”の中に収められており

- Makefile.main
- Makefile.var
- conditions.f90
- control.part.inp
- driver.f90

である。conditions.f90 では control.inp の読み込みや各条件に用いるエネルギーや密度の定義、plot.*.*.dat などの書き出しを行っている。

6.2.3 reduction

複数条件の 0 次元化学計算による簡略化を行う。モニターする値は平衡温度と着火時刻のみであり、この二者があつていれば、どのような簡略化でも許すようになっている。また、簡略化は化学種の削減と素反応の削減のみにより行われる。

燃料と酸化剤の定義は control_chem.inp で流体計算と同様に行う。control.inp は次のような入力になる。

```
Pressure (Pa)      range and Ntics : 1.e6   1.e6   1
Temperature (K)    range and Ntics : 1300. 1500. 2
mass fraction of fuel range and Ntics : 0.15   0.25   3
Allowable tolerance of temperature : 0.05
```

入力は初期圧力、初期温度、燃料質量分率及びそのサンプル数、許容する温度誤差(割合)である。上記例であれば、許容温度誤差 5% となる。

実行は

1. 計算可能な条件点の判定
2. 化学種の並び替え及び重要度の少ない化学種からの削減 (reduct_species_in_order)
3. 全ての化学種のチェック (reduct_every_species)
4. 素反応の並び替え及び重要度の少ない素反応からの削減 (reduct_reaction_in_order)
5. 全ての素反応のチェック (reduct_every_reaction)
6. 化学種の並び替え及び重要度の少ない化学種からの削減 (reduct_species_in_order)
7. 全ての化学種のチェック (reduct_every_species)

のように行われる。

化学種の重要度は全条件全時間におけるその化学種の最大質量分率の大きさで評価される。初回並び替え結果は”sort.dat”に保存されている。

素反応の重要度は CSP 法により評価されている。全時間の CSP の値は各時間の CSP の値の最大値をとることで評価されている。初回並び替え結果は”sort_r.dat”に保存されている。

”sort.dat”と”sort_r.dat”は NASA データベースの際と同様に gnuplot でプロットすることができる。

実行中には現在の化学種数および素反応数が表示される。

新しく作成されたモデルは chem.inp.new とファイルに保存される。これを chem.inp とすれば、流体計算や 0 次元化学計算に用いることができる。

特有のファイルは”store/therm_lib/chemkin/reduction”の中に収められており

- Makefile.main
- Makefile.var
- conditions.f90
- control.part.inp

- driver.f90

である。conditions.f90 では control.inp の読み込みや各条件に用いるエネルギー や密度の定義、簡略化サブルーチンの定義を行っている。