

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。

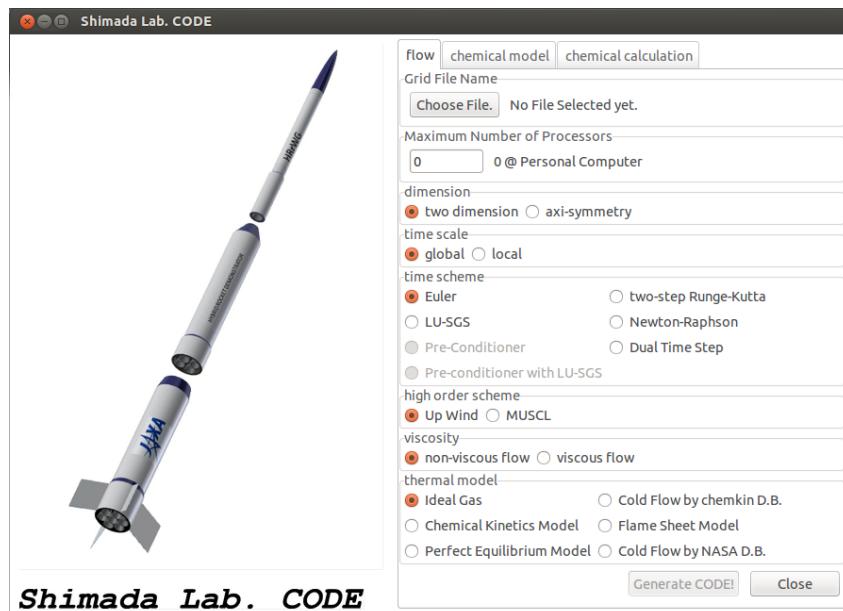
よって、コード生成は GUI.py を通して行うことをお勧めする。

GUI.py は以下のコマンドで起動される。

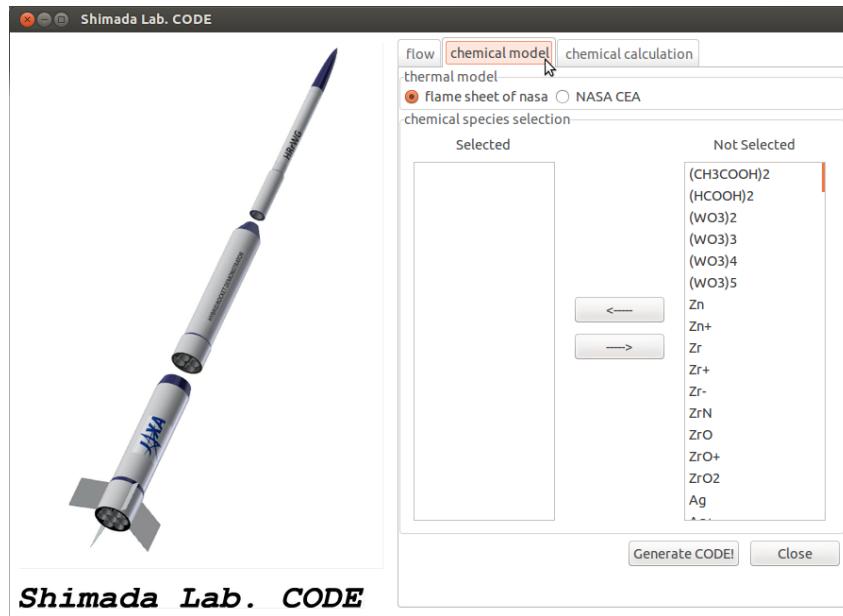
重要

./GUI.py

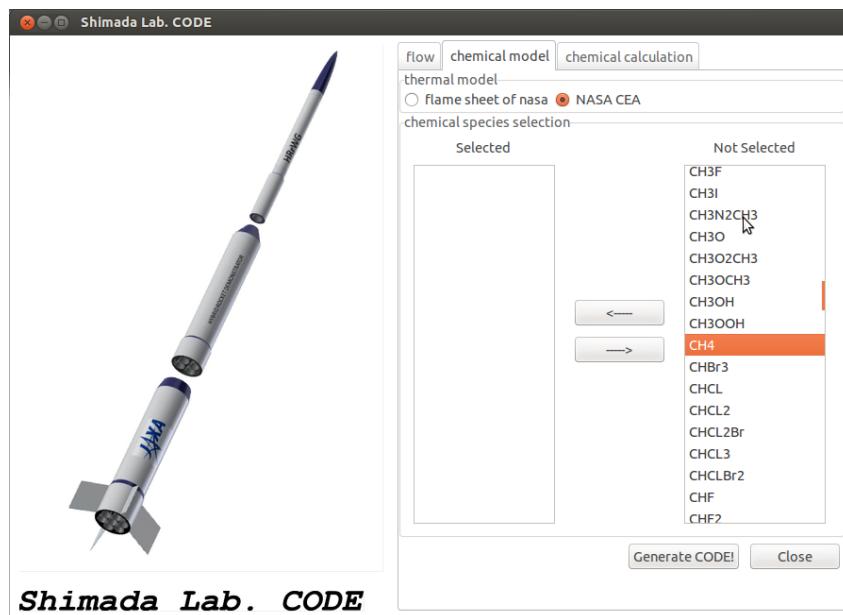
起動されると以下の画面が立ち上がる。

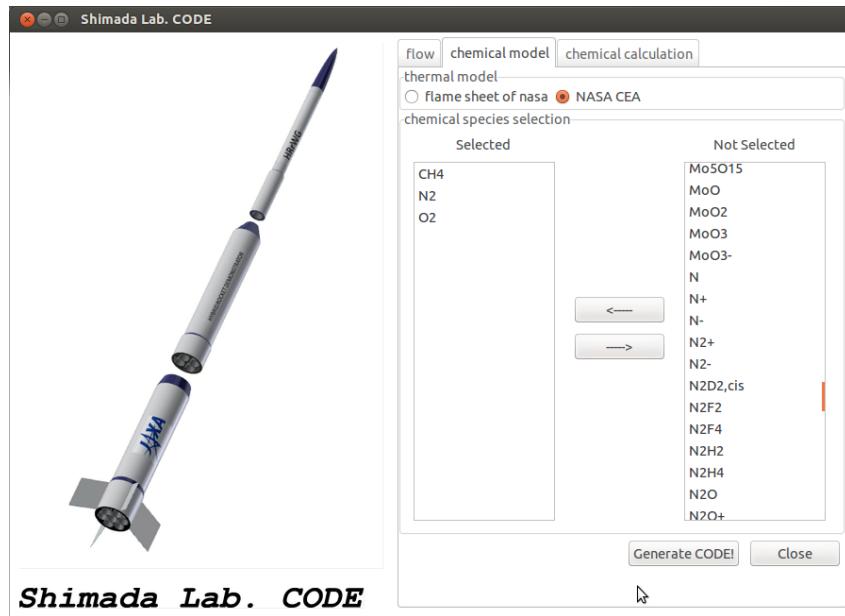


今回はまず元モデルの生成を行う。まずは上のタブで”chemical model”を選択する。

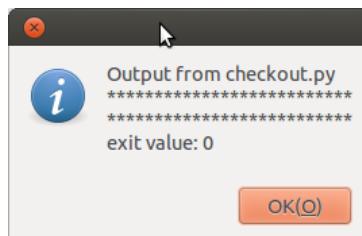


起動されたら、まず thermal model で完全平衡モデル”NASA CEA”を選び、“Not Selected”の中から生成物として使いたい化学種をダブルクリックすることで選択する。今回はメタン空気の燃焼なので、CH₄, N₂, O₂の三つを選べばよい。なお、この際”Not Selected”内で Ctrl+L をうつと検索モードとなり、所望の化学種を素早くみつけることができる。





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。

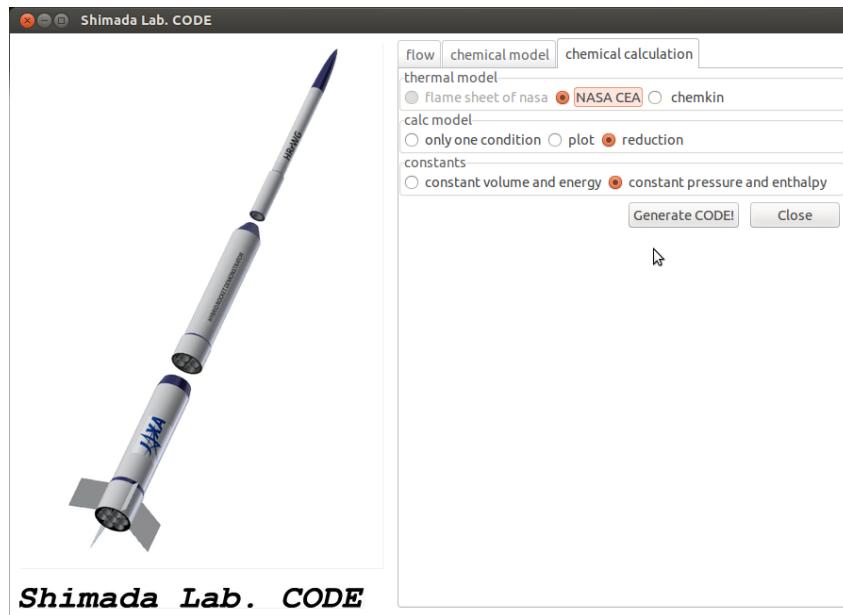
重要

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。



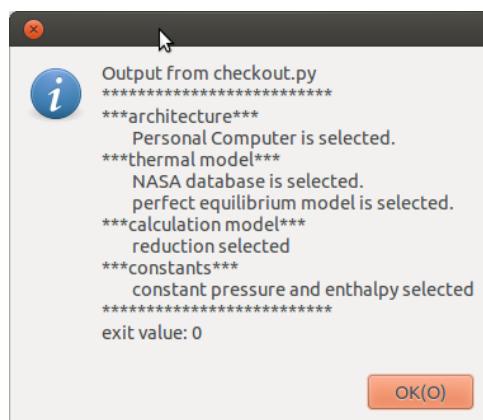
図のように

- 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。

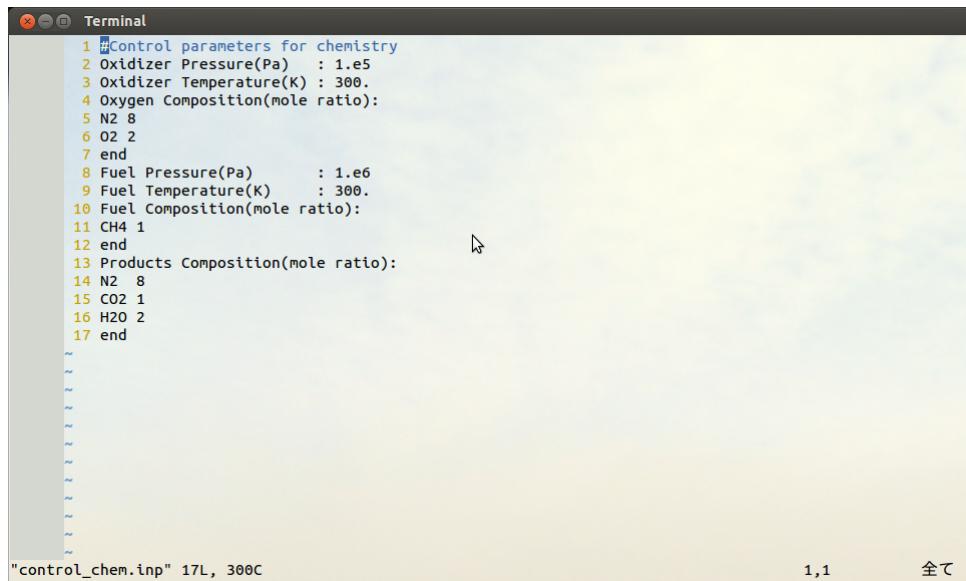
以上の動作を行うと、ディレクトリは以下のようになる。



```
$ ls
GUI.py  checkout.py  checkout_chem.inp  control_chem.raw.inp  postprocess  store
Makefile  checkout_chem  chem.inp        doc                  sample       trunk_fs
$
```

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。



```
1 #Control parameters for chemistry
2 Oxidizer Pressure(Pa)      : 1.e5
3 Oxidizer Temperature(K)   : 300.
4 Oxygen Composition(mole ratio):
5 N2 8
6 O2 2
7 end
8 Fuel Pressure(Pa)         : 1.e6
9 Fuel Temperature(K)       : 300.
10 Fuel Composition(mole ratio):
11 CH4 1
12 end
13 Products Composition(mole ratio):
14 N2 8
15 CO2 1
16 H2O 2
17 end
~
```

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??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。

```
Terminal
$ 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.raw.inp func_chem.f90  mod_mpi_dummy.f90 thermo.inp
$
```

ここにも control.raw.inp という”raw”がつくファイルがあるので、適切に編集したあと control.inp に変更しなければならない。

- 重要

このように、”raw”がつくファイルが生成した場合には、適切に編集の上”raw”をとったファイル名に名前変更しなければならない。

このようなファイルには他に condition.raw.f90 がある。

control.inp は以下のように変更する。

詳細な書式は??参照のこと。以上の編集がおわったら、control.raw.inp を control.inp に変更する。

```
$ 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
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
```

これから、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。

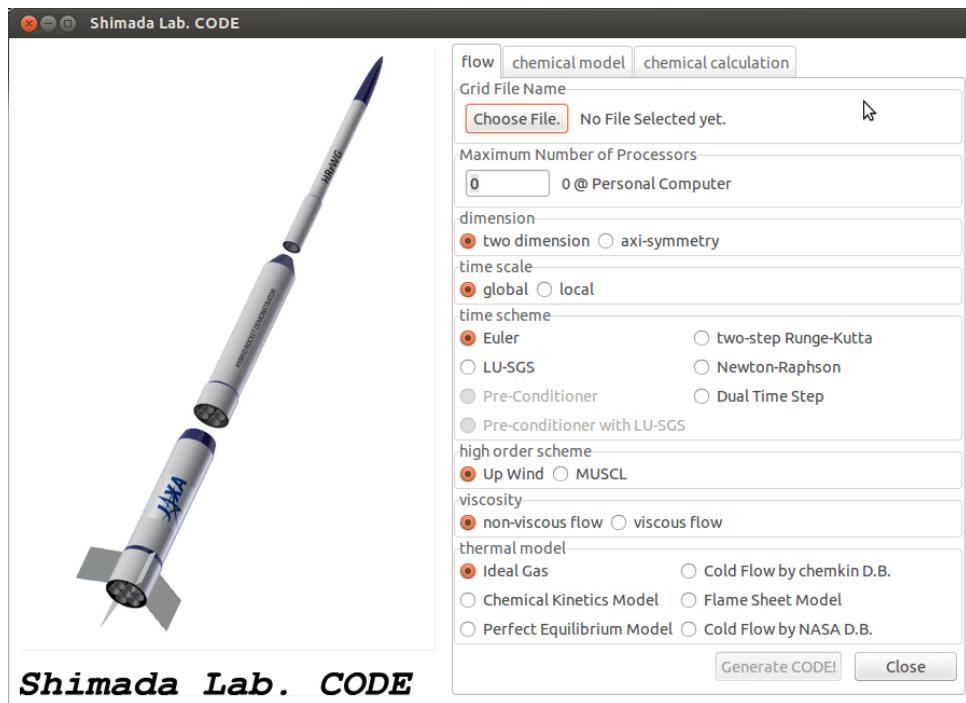
2.2 簡略モデルを用いた作動流体にメタンと空気を用いた衝撃波管問題

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

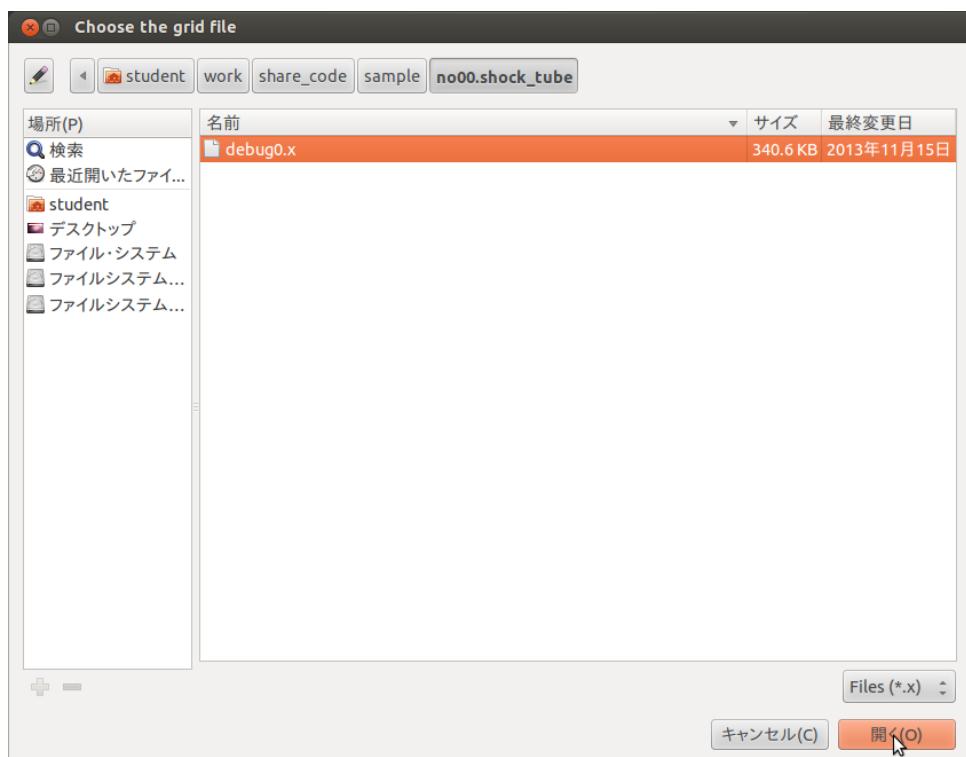
```
cp checkout_chem/chem.inp.new chem.inp
```

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。



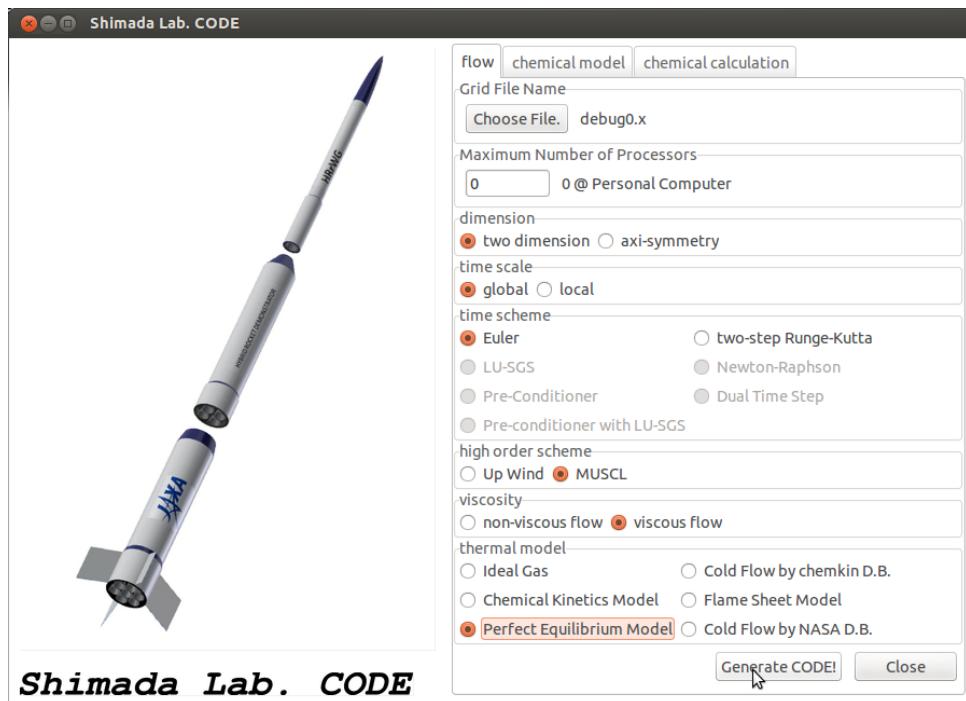
まずは Grid File Name でグリッドを選ぶ。”Choose File”を押下すれば、ファイル選択画面が現れる。本チュートリアルでは sample/no00.shock_tube/debug0.x を選択する。



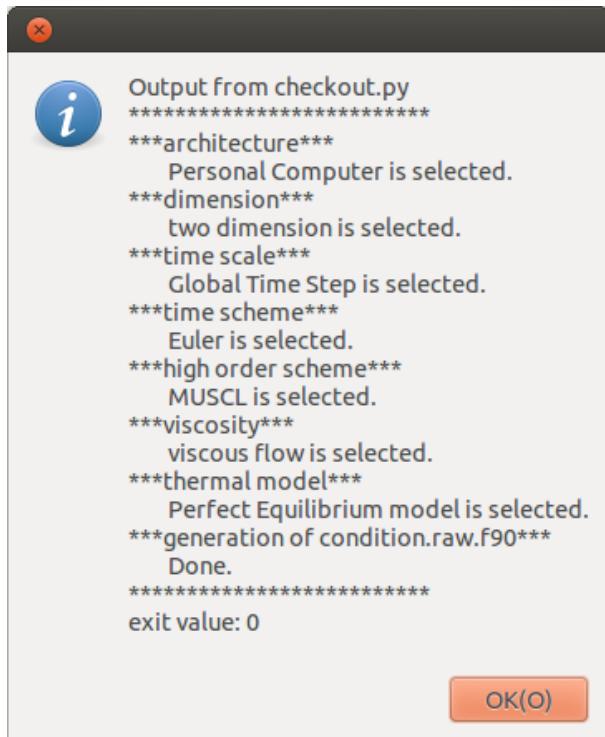
グリッドファイル選択後は

- 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 ?? 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。

```

$ 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 ライブラリを使っているので、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。

```
Terminal
$ 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.inp 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。

```
Terminal
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 ?? 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 ??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。

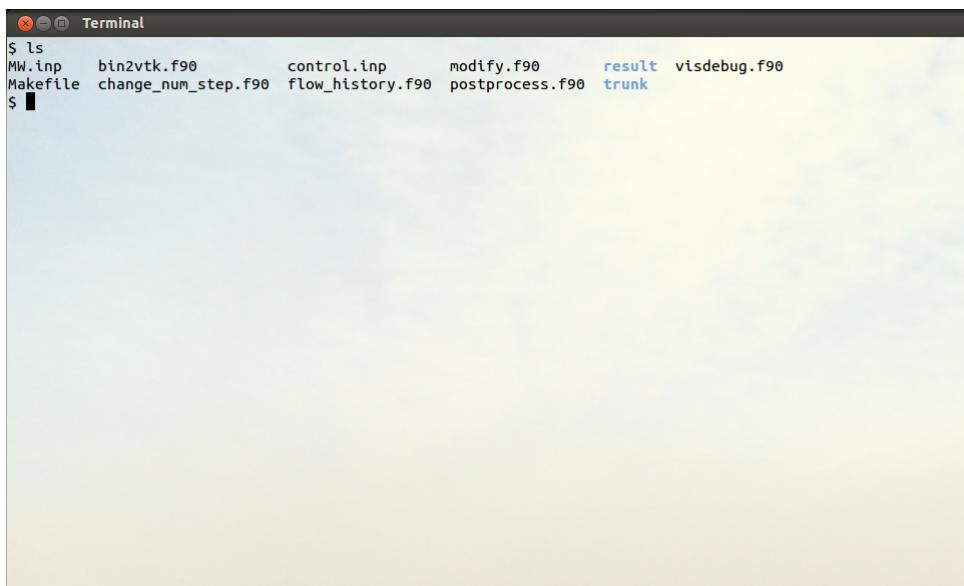
```

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。

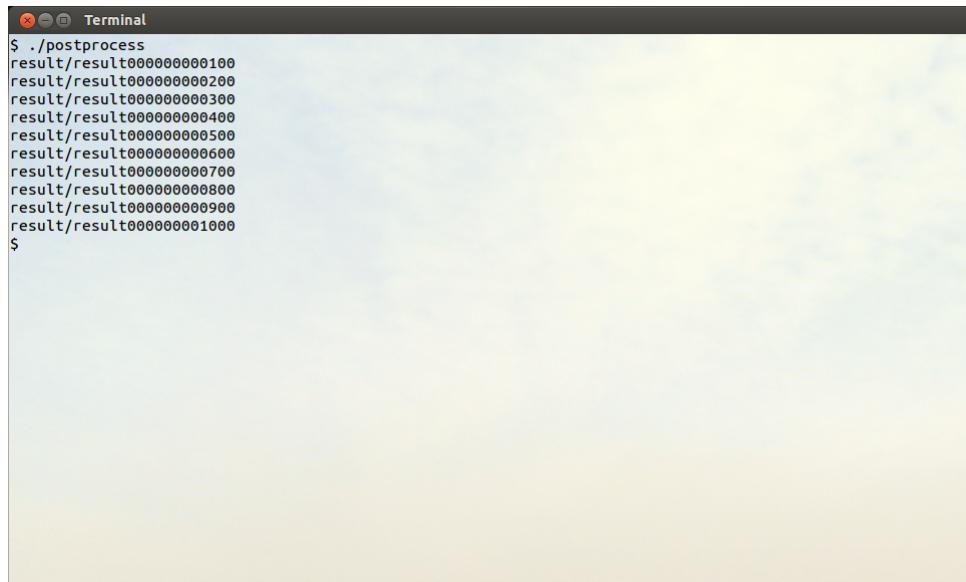
make を実行すると、実行ファイル postprocess が生成される。この入力ファイル control.inp は以下のように設定する。

それぞれの意味は section ?? 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。

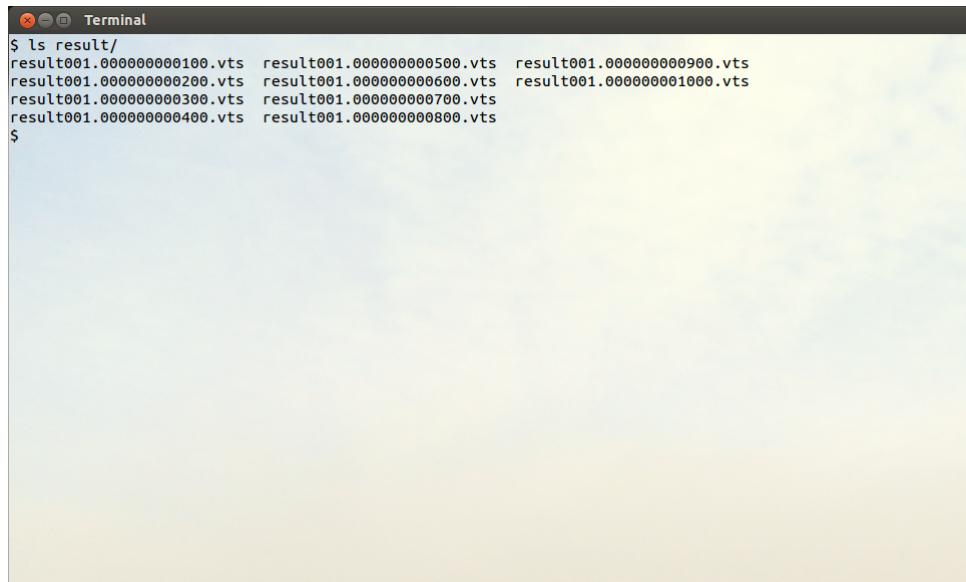
```
Terminal
1 #Input File for PostProcessing
2 File Read Freq.          : 100
3 From Where               : -1
4 Database Place           : ../checkout/
5 OverWrite? Yes(1)/No(0) : 0
6 1st Plane i start index : 0
7 1st Plane i end   index : -1
8 1st Plane j start index : 0
9 1st Plane j end   index : -1
10 1st Plane i start index : 210
11 1st Plane i end   index : 400
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。



```
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。

