

# iSALE

## ～ソースコード改良入門編～

末次竜  
産業医科大





# イントロダクション

- ・iSALEは基本的にソースコードの内容を知らなくても動くように設計されている
- ・インストール時に作成された実行ファイル(iSALE2D)が初期条件(asteroid.inp)、物質の情報(material.inp)を読み込み計算
  - 毎回、コンパイルしなくてよい
- ・計算結果もpySALEPlot、VIMoDを使うことで図として表示することができる



# イントロダクション

- ・一方で、設定されている物理量以外を出力することはできない
- ・計算結果はバイナリで出力されるため、テキスト化や他の描画ソフト(gnuplot)で見るには一手間必要
- ・コード内でどういう計算が行われているのかがわからない

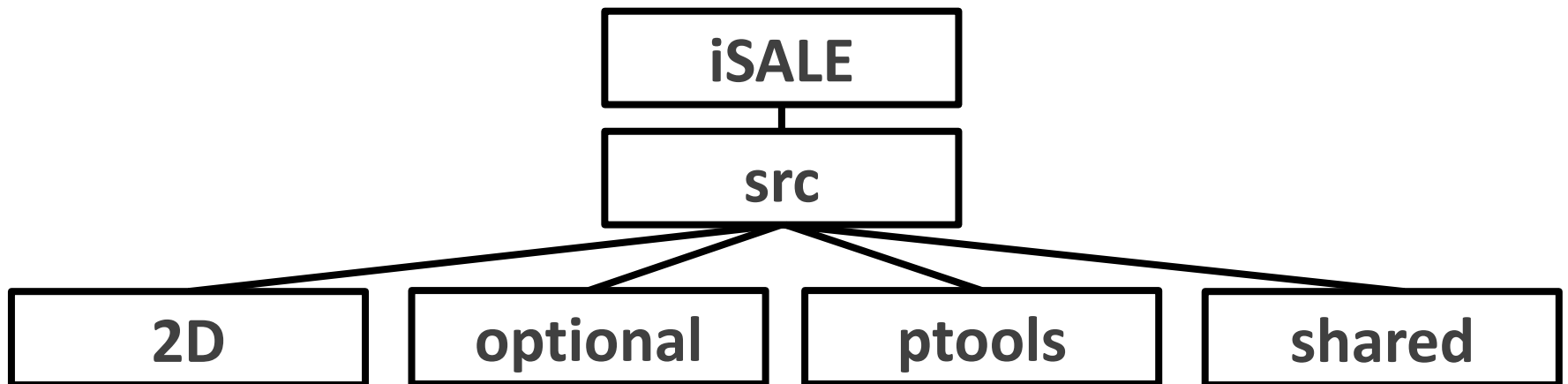


# イントロダクション

- ・今回の発表では、ソースコードの場所やコードの変更の仕方を簡単に紹介

# ソースコードの場所

- ・ソースコードはダウンロードしたディレクトリ内の  
/iSALE/src/に保管されている  
(私の場合~/iSALE-Dellen/work/iSALE/src/)
- /iSALE/src/には以下の四つのディレクトリ





# ソースコードの場所

**2D**

実行ファイルiSALE2Dを作成するためのプログラムを保管

**shared**

iSALEの2Dと3Dで共通して使われるプログラムを保管

**ptools**

主にiSALEMatとiSALEParを作成するためのプログラムを保管(一部iSALE2Dでも使用)

**optional**

不明(iSALE2Dの並列計算プログラム？ANEOSのtable作成プログラム？)





# ソースコードの場所

## 2D

- ・実行ファイルiSALE2Dを作成するためのプログラムを保管  
(Makefileやメイン関数(iSALE2D.F90)など)
- ・出力を変更したい場合は、write.F90などの中身を変更すればいい

## shared

- ・iSALEの2Dと3Dで共通して使われるプログラムを保管  
(.inpファイルを読み込むプログラムなど)
- ・モジュールはこのディレクトリに保管されている  
変数の多くはmod\_isale.F90内で宣言

# ソースコードの場所

・ディレクトリ2D内の実行ファイルはデモ計算の実行ファイルとは別物(bin以下のiSALE2Dにリンク)

例 demo2D内で以下のコマンド入力

```
$ ls -l
```

```
suetsugu@pc1606:~/iSALE-Dellen/install/share/examples/demo2D$ ls -l
合計 6188
drwxrwxr-x 2 suetsugu suetsugu 4096 10月 26 22:06 Plots
drwxrwxr-x 2 suetsugu suetsugu 4096 10月 26 21:54 Plotting
-rw-r--r-- 1 suetsugu suetsugu 4610 10月 26 21:54 asteroid.inp
drwxrwxr-x 5 suetsugu suetsugu 4096 10月 26 22:03 demo2D
lrwxrwxrwx 1 suetsugu suetsugu 45 10月 26 21:54 eos -> /home/suetsugu/iSALE-Dellen/install/share/eos
lrwxrwxrwx 1 suetsugu suetsugu 47 10月 26 21:54 iSALE2D -> /home/suetsugu/iSALE-Dellen/install/bin/iSALE2D
lrwxrwxrwx 1 suetsugu suetsugu 48 10月 26 21:54 iSALEMat -> /home/suetsugu/iSALE-Dellen/install/bin/iSALEMat
lrwxrwxrwx 1 suetsugu suetsugu 48 10月 26 21:54 iSALEPar -> /home/suetsugu/iSALE-Dellen/install/bin/iSALEPar
```

・更新したiSALE2Dをbin以下にコピーすることで、はじめて更新したiSALE2Dの実行できる





# iSALE2D.F90の中身

- ・ディレクトリ2D以下のiSALE2D.F90がメイン関数  
(iSALE-chicxulubの場合はmain.F90)
- ・一応、各サブルーチンについて、短く説明が書かれているので、慣れるまでは、このプログラムから目的の物理量が計算されているプログラムまで辿るのがおすすめ

# iSALE2D.F90の中身

35行目まで

モジュールから使う変数の呼び出し

```
PROGRAM ISALE
```

```
USE MOD_ISALE, only : dumpname, &  
    out, ioerr, STDOUT, ioinit, ioener, iomass, iostep, &  
    time, tend, tlim, ale_mode, lagrangian, eulerian, &  
    tracer_motion, TR_VEL, stop_after_setup, verbose, vl, ncyc, eps_min  
use ptool_interface, only : io_out, io_oin, ptool_text, ptool_text_int  
use mod_parallel  
use mod_strength_routines, only : check_state_all  
use mod_identify_mat  
use mod_regrid, only : dynamic_regrid
```

```
implicit none
```

```
real :: time_start !< Wall time of simulation start  
real :: time_now   !< Wall time now  
real :: time_cpu   !< Duration of simulation (wall time)
```

```
out = STDOUT ! for the very first beginning...  
ioerr = STDOUT
```

```
! first start with initializing MPI (if required)...  
call initialize_mpi()
```

```
!DE flag analyzation must come before any text output  
! to allow the use of help2man to generate a manpage.  
call analyze_flags ! interpret user-defined command line statements
```

iSALE2D.F90でのみ使用する  
変数の宣言



# iSALE2D.F90の中身

35行目～50行目

- ・iSALE2Dを実行すると端末に表示されるもの  
→実習1ではここに名前加えます

```
write(*,*)
write(*,*)" ++++++ "
write(*,*)" +++          iSALE          +++ "
write(*,*)" +++      by Kai Wuennemann, Gareth Collins,  +++ "
write(*,*)" +++      Dirk Elbeshausen and Tom Davison  +++ "
if (verbose>0) then
  write(*,*)" +++          +++ "
  write(*,*)" +++ based on SALEB by Ivanov          +++ "
  write(*,*)" +++          SALES by Melosh          +++ "
  write(*,*)" +++          SALE by Amsden et al.    +++ "
  write(*,*)" +++          +++ "
endif
write(*,*)" +++      Contact: isale@imperial.ac.uk    +++ "
write(*,*)" ++++++ "
write(*,*)
```



# iSALE2D.F90の中身

50行目～73行目

```
call read_param()      ! generate log-files, open input-files
call setup_derive_param() ! compute derived parameters and prepare EOS, etc.
call setup_info()      ! acquire some information
call decomp()
if (dumpname(1:4).eq.'NONE') then ! No restart requested
  call celset          ! Generate Grid and set model
  call setup_check()  ! Perform setup sanity checks
else
  call restore        ! Check for restart
end if

! Identify the materials in each cell...
call identify_mat(eps_min)

! If just performing a set-up check end here
call info_stop_after_setup()

! acquire and store start time information
call CPU_TIME(time_start)
time_cpu=0.
```

astroid.inp、material.inp  
の読み込み、dumpfile  
の有無の確認、  
初期状態を計算、  
端末への表示など

## 73行目～108行目

```
! Loop until reached end time, or reached wall time
do while (time.lt.tend.and.time_cpu.lt.tlim)

  if (dynamic_regrid .eq. 1) call update_dynamic_regrid ! Check for dynamic regrid then apply
  call update_state ! Initialize some values at the beginning of cycle
  call check_state_all() ! check the state/phase of the materials
  call write_dump(0) ! dump data for restart
  call savedata ! Save data
  call update_timestep ! Calculate timestep
  if (tracer_motion .eq. TR_VEL) call movetracer ! move tracer according to velocities
  call update_cycle ! Calls for data dumps, screen prints ...
  call update_velocity_stability ! Calc. artf. viscosity and alternate node coupling
  call update_velocity_stress ! Calc. stress deviators
  call update_velocity_pressure ! Explicit lagrangian calc. with respect to p
  call update_energy ! Update Energy

  if (ALE_MODE /= EULERIAN) call update_mesh ! Move the mesh
  if (ALE_MODE /= LAGRANGIAN) call advect ! Flux quantities

  ! mark pressure field as stored
  ! (for calculation of differential pressure required)

  ! Compute cpu time since job began
  call CPU_TIME(time_now)
  time_cpu = time_now-time_start

end do
```

TimeがTEND以上になるまでループし、計算を続ける



# iSALE2D.F90の中身

103行目～116行目

- ・ループ終了後の後処理と計算終了

```
! Finish simulation...  
call savedata  
call write_dump(1) ! Make data dump on exit  
call finalcycle(time_cpu)  
call isale_finalize("<<>>")
```

```
END PROGRAM ISALE
```





# ソースコードの変更

変更の仕方は極めて原始的

1. iSALE2D.F90やマニュアルを見て、変更したい物理量を扱ってそうな変数及び、サブルーチン名をみつける
2. grepコマンドを使って当たりをつけた変数を含むプログラムを検索

```
$ grep "変数名" *.F90
```

# ソースコードの変更

3. 検索した文字を含むプログラムが表示される

例

```
$ grep "time_cpu" *.F90
```

```
suetsugu@pc1606:~/iSALE-Dellen/work/iSALE/src/2D$ grep "time_cpu" *.F90
iSALE2D.F90: real :: time_cpu !< Duration of simulation (wall time)
iSALE2D.F90: time_cpu=0.
iSALE2D.F90: do while (time.lt.tend.and.time_cpu.lt.tlim)
iSALE2D.F90:     time_cpu = time_now-time_start
iSALE2D.F90: call finalcycle(time_cpu)
```

“time\_cpu”が書かれているプログラム

4. 定義等が見つからなければ、別の変数等に変更してプログラムを絞り込んで見つける

# ソースコードの変更

## 5.プログラムを変更する

例 time\_cpuを端末に表示する

```
! Compute cpu time since job began
call CPU_TIME(time_now)
time_cpu = time_now-time_start

!+++++
time_output = time_output + dt           !---R.S.
if(time_output>0.1)then                 !---R.S.
  write(*,*) 'TIME[s]=' , time,'CPU TIME[s]=' , time_cpu !---R.S.
  time_output = 0.0                       !---R.S.
end if                                    !---R.S.
!+++++

end do

! Finish simulation...
call savedata
call write_dump(1) ! Make data dump on exit
call finalcycle(time_cpu)
call isale_finalize("<<<END OF SIMULATION>>>")
```

END PROGRAM ISALE



# ソースコードの変更

6. プログラムを更新したので、コンパイルする

```
$ make
```

7. 実行ファイル(iSALE2D)が更新されたので、  
実行ファイルをリンク先に移動する

```
$ cp iSALE2D /home/suetsugu/iSALE-Dellen/install/bin/
```

これでどのデモ計算を行った場合でも、  
更新したiSALE2Dが実行されることになる



# 実習

- 実習1 : 名前の追加
- 実習2 : output.txt0への出力回数の変更
- 実習3 : 端末への計算時間の表示(時間があれば)



# 実習1:名前の追加

- ・iSALE2Dを実行すると以下のように表示される

```
suetsugu@pc1606:~/iSALE-Dellen/install/share/examples/demo2D$ ./iSALE2D
+++++
+++          iSALE          +++
+++  by Kai Wuennemann, Gareth Collins,  +++
+++  Dirk Elbeshausen and Tom Davison    +++
+++                                     +++
+++  based on SALEB   by Ivanov          +++
+++                SALES by Melosh       +++
+++                SALE  by Amsden et al. +++
+++                                     +++
+++  Contact: isale@imperial.ac.uk       +++
+++++
```

コードをいじるので自分の名前を追加しよう





# 実習1:名前の追加

- ・ソースコードの保管されているディレクトリに移動

```
$ cd ~/iSALE-Dellen/work/iSALE/src/2D
```

人によって異なる

- ・iSALE2D.F90を変更するので、**変更前にバックアップをとっておく**

```
$ cp iSALE2D.F90 iSALE2D_bk.F90
```

(iSALE-chicxulubの場合はmain.F90なのでmain\_bk.F90などにする)

# 実習1:名前の追加

- ・適当なテキストエディタでiSALE2D.F90を開く

```
$emacs iSALE2D.F90 -nw
```

テキストエディタが  
emacsの場合

```
write(*,*)
write(*,*)" ++++++ "
write(*,*)" +++          iSALE          +++ "
write(*,*)" +++      by Kai Wuennemann, Gareth Collins,  +++ "
write(*,*)" +++      Dirk Elbeshausen and Tom Davison    +++ "
if (verbose>0) then
  write(*,*)" +++          +++ "
  write(*,*)" +++  based on SALEB  by Ivanov          +++ "
  write(*,*)" +++          SALES  by Melosh          +++ "
  write(*,*)" +++          SALE   by Amsden et al.    +++ "
  write(*,*)" +++          +++ "
endif
write(*,*)" +++      Contact: isale@imperial.ac.uk      +++ "
write(*,*)" ++++++ "
write(*,*)
```

# 実習1:名前の追加

- 適当なテキストエディタでiSALE2D.F90を開く

```
$emacs iSALE2D.F90 -nw
```

テキストエディタが  
emacsの場合

```
write(*,*)
write(*,*)" ++++++ "
write(*,*)" +++          iSALE          +++ "
write(*,*)" +++      by Kai Wuennemann, Gareth Collins,      +++ "
write(*,*)" +++      Dirk Elbeshausen and Tom Davison      +++ "
if (verbose>0) then
  write(*,*)" +++          ここに名前を入れる          +++ "
  write(*,*)" +++      based on SALEB   by Ivanov          +++ "
  write(*,*)" +++          SALES   by Melosh          +++ "
  write(*,*)" +++          SALE     by Amsden et al.      +++ "
  write(*,*)" +++ "
endif
write(*,*)" +++      Contact: isale@imperial.ac.uk      +++ "
write(*,*)" ++++++ "
write(*,*)
```

# 実習1:名前の追加

## ・例えば

```
write(*,*)
write(*,*)" ++++++ "
write(*,*)" +++          iSALE          +++ "
write(*,*)" +++      by Kai Wuennemann, Gareth Collins,  +++ "
write(*,*)" +++      Dirk Elbeshausen, Tom Davison      +++ "
write(*,*)" +++          and Ryo Suetsugu          +++ "
if (verbose>0) then
  write(*,*)" +++          +++ "
  write(*,*)" +++      based on SALEB   by Ivanov          +++ "
  write(*,*)" +++          SALES   by Melosh          +++ "
  write(*,*)" +++          SALE     by Amsden et al.      +++ "
  write(*,*)" +++          +++ "
endif
write(*,*)" +++      Contact: isale@imperial.ac.uk      +++ "
write(*,*)" ++++++ "
write(*,*)
```

## ・追加できたら保存する(emacsだとCtrl + X, Ctrl + S)



# 実習1:名前の追加

- ・iSALE2D.F90を更新したので、コンパイルする

```
$make
```

- ・実行ファイル(iSALE2D)が更新されたので、  
実行ファイルをリンク先に移動する

```
$cp iSALE2D /home/suetsugu/iSALE-Dellen/install/bin
```



# 実習1:名前の追加

- ・確認のためdemo2Dを実行してみると、

```
suetsugu@pc1606:~/iSALE-Dellen/install/share/examples/demo2D$ ./iSALE2D
```

```
+++++  
+++          iSALE          +++  
+++  by Kai Wuennemann, Gareth Collins,  +++  
+++  Dirk Elbeshausen, Tom Davison      +++  
+++  and Ryo Suetsugu          +++  
+++                                     +++  
+++  based on SALEB   by Ivanov          +++  
+++          SALES   by Melosh          +++  
+++          SALE    by Amsden et al.    +++  
+++                                     +++  
+++  Contact: isale@imperial.ac.uk      +++  
+++++
```



# 実習2:出力する頻度の変更

- output.txt0には、様々な情報が記録される  
demo2Dの場合、demo2D/demo2D/以下

```
Initial total energy and mass:
IE: 7.1180726904816678E+020 KE: 2.0086850477740737E+019
TE: 7.3189411952590756E+020 M: 1799741142126422.2

CYC= 0 T= 0.0000E+00 DT= 1.000E-03 FLAG=G
TE= 100.0000 KE= 100.0000 IE= 0.0000 TM= 100.0000
GMESH= 0.00E+00 GTARG= 0.00E+00 PERCENT COMPLETE: 0.000%

LIMITING CELL: I= 1 J= 111
CONC= 1.0000 1.0000
VERTEX 1..4= 1.0000 1.0000 1.0000 1.0000
C_SOUND= 4012.53 C_MAT= 6500.00

-----
SAVE DATA (jpeg-format): 10 1.2577892535548835E-002
wrote 'Cm1' : ( 113: 141), min/max = 0.00000E+00 1.00000E+00
wrote 'Den' : ( 113: 141), min/max = 0.00000E+00 3.43755E+03
wrote 'Pre' : ( 113: 141), min/max = -8.26891E+06 2.98189E+10
wrote 'Tmp' : ( 113: 141), min/max = 0.00000E+00 5.22904E+03
wrote 'Yld' : ( 113: 141), min/max = 0.00000E+00 1.84336E+09
wrote 'Dam' : ( 113: 141), min/max = 0.00000E+00 9.97620E-01
wrote 'Ert' : ( 113: 141), min/max = 0.00000E+00 1.62985E+01
wrote 'Vib' : ( 113: 141), min/max = 0.00000E+00 2.00000E+02
wrote 'YAc' : ( 113: 141), min/max = 0.00000E+00 1.00000E-08
wrote 'PVb' : ( 113: 141), min/max = 0.00000E+00 3.54556E+09
wrote 'V_x' : ( 113: 141), min/max = -1.93512E+01 9.26335E+02
wrote 'V_y' : ( 113: 141), min/max = -6.64471E+03 2.81579E-01
----- finished writing cycle -----
```

# 実習2:出力する頻度の変更

- output.txt0には、様々な情報が記録される  
demo2Dの場合、demo2D/demo2D/以下

```
Initial total energy and mass:
IE: 7.1180726904816678E+020 KE: 2.0086850477740737E+019
TE: 7.3189411952590756E+020 M: 1799741142126422.2


CYC= 0 T= 0.0000E+00 DT= 1.000E-03 FLAG=G
TE= 100.0000 KE= 100.0000 IE= 0.0000 TM= 100.0000
GMESH= 0.00E+00 GTARG= 0.00E+00 PERCENT COMPLETE: 0.000%

LIMITING CELL: I= 1 J= 111
CONC= 1.0000 1.0000
VERTEX 1..4= 1.0000 1.0000 1.0000 1.0000
C_SOUND= 4012.53 C_MAT= 6500.00
```

50ステップに  
1回出力  
ソースコード内で  
決められており  
変更不可

```
SAVE DATA (jpeg-format): 10 1.2577892535548835E-002
wrote 'Cm1' : ( 113: 141), min/max = 0.00000E+00 1.00000E+00
wrote 'Den' : ( 113: 141), min/max = 0.00000E+00 3.43755E+03
wrote 'Pre' : ( 113: 141), min/max = -8.26891E+06 2.98189E+10
wrote 'Tmp' : ( 113: 141), min/max = 0.00000E+00 5.22904E+03
wrote 'Yld' : ( 113: 141), min/max = 0.00000E+00 1.84336E+09
wrote 'Dam' : ( 113: 141), min/max = 0.00000E+00 9.97620E-01
wrote 'Ert' : ( 113: 141), min/max = 0.00000E+00 1.62985E+01
wrote 'Vib' : ( 113: 141), min/max = 0.00000E+00 2.00000E+02
wrote 'YAc' : ( 113: 141), min/max = 0.00000E+00 1.00000E-08
wrote 'PVb' : ( 113: 141), min/max = 0.00000E+00 3.54556E+09
wrote 'V_x' : ( 113: 141), min/max = -1.93512E+01 9.26335E+02
wrote 'V_y' : ( 113: 141), min/max = -6.64471E+03 2.81579E-01
----- finished writing cycle -----
```

asteroid.inp内の  
DTSAVEで  
出力頻度の  
変更可



# 実習2:出力する頻度の変更

・PROGRESS以下には保存量の時間変化が記録

demo2Dの場合、demo2D/demo2D/以下

\* 質量保存はmass.txt0

1行目:時間

2行目:現在の質量

3行目:(現在の質量-初期質量)/初期質量

\* エネルギー保存はenergy.txt0

1行目:時間

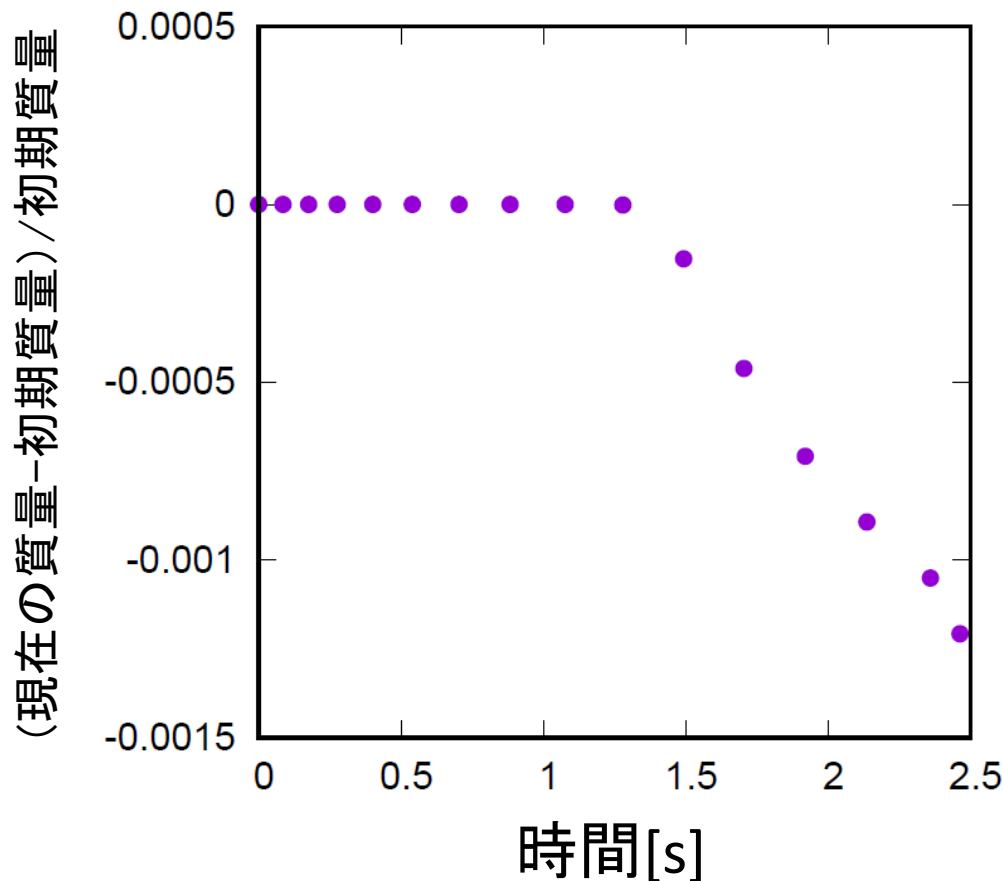
2行目:現在の全エネルギー - 初期の全エネルギー

3行目:現在の内部エネルギー - 初期の内部エネルギー

4行目:現在の運動エネルギー - 初期の運動エネルギー

# 実習2:出力する頻度の変更

・例: demo2Dのmass.txt0の一行目と三行目



これらの出力頻度も  
50ステップに1回で固定


グラフの表示

demo2D/demo2D/PROGRESS  
内でgnuplotを立ち上げる

```
$ gnuplot
```

以下のコマンドを入力

```
plot "mass.txt0" u 1:3
```



## 実習2:出力する頻度の変更

- ・50ステップに1回から10ステップに1回に変更する
- ・ディレクトリ2D内のupdate\_cycle.F90をコピーする

```
cp update_cycle.F90 update_cycle_bk.F90
```

(iSALE-chicxulubの場合はnewcyc.F90なので、newcyc\_bk.F90などにする)

- ・テキストエディタでupdate\_cycle.F90を開く

```
emacs update_cycle.F90 -nw
```







```
use mod_timestep
use mod_grind
use mod_conservation
implicit none
integer i,j,step,m
```

```
! ++++++
```

```
step=50
```

← この変数の値を変える

```
! Calculate initial energy and mass
!
if (ncyc   
 call step=10 alize grind calculation
```


今回は10にする

```
! compute initial total energy and mass
call conservation_calc(1)
write(OUT,*) 'Initial total energy and mass:'
write(OUT,*) 'IE: ',toti0,' KE: ',totk0
write(OUT,*) 'TE: ',tote0,' M: ',totm0
endif
```

```
! Calculate the grind (mean processing time per cell)
!
call grind_calc()
```

```
! Produce some monitor print to assess progress and stability...
```

```
if (mod(ncyc,step).eq.0) then ステップ数が50で割りきれた  
時に出力している  
call conservation_calc(0)  
call conservation_record(ioener,iomass,time)
```



## 実習2:出力する頻度の変更

- step=10に変更したら保存(emacsだとCtrl + X, Ctrl + S)
- update\_cycle.F90を更新したのでコンパイルする

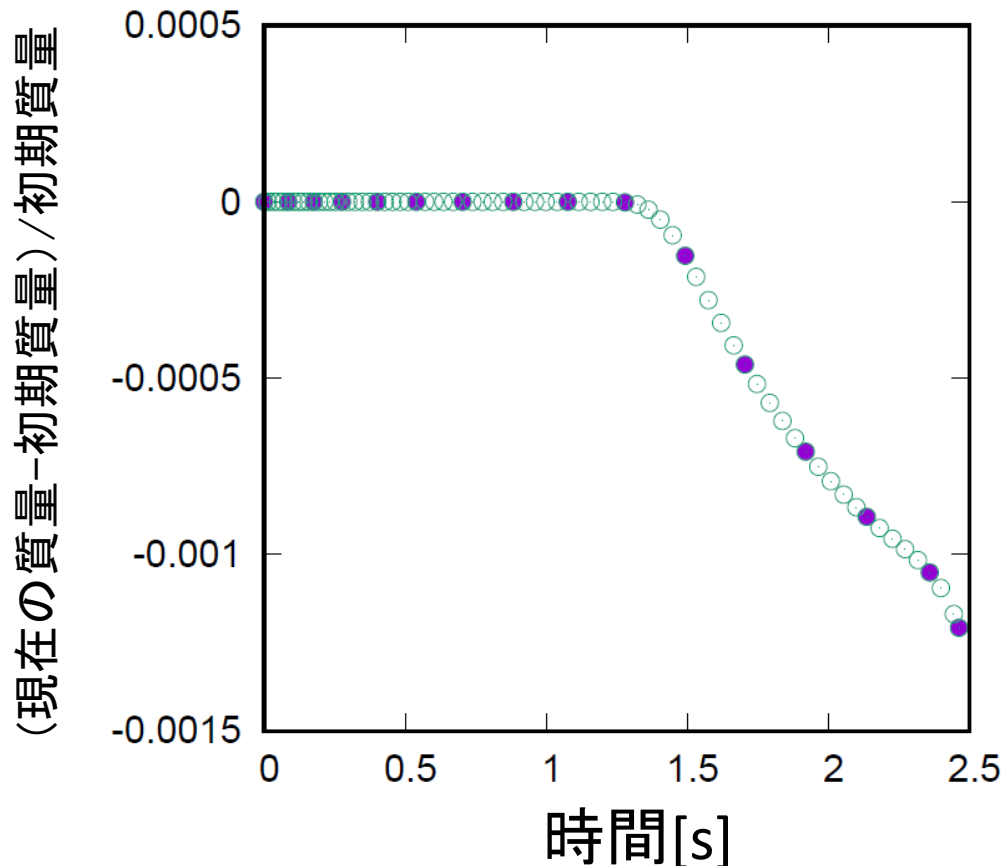
```
make
```

- 実行ファイルをリンク先に移動する

```
cp iSALE2D /home/suetsugu/iSALE-Dellen/install/bin
```

# 実習2:出力する頻度の変更

▪ 例: demo2Dのmass.txt0の一行目と三行目



出力頻度が  
10ステップに1回に増加

グラフの表示

demo2D/demo2D/PROGRESS  
内でgnuplotを立ち上げる

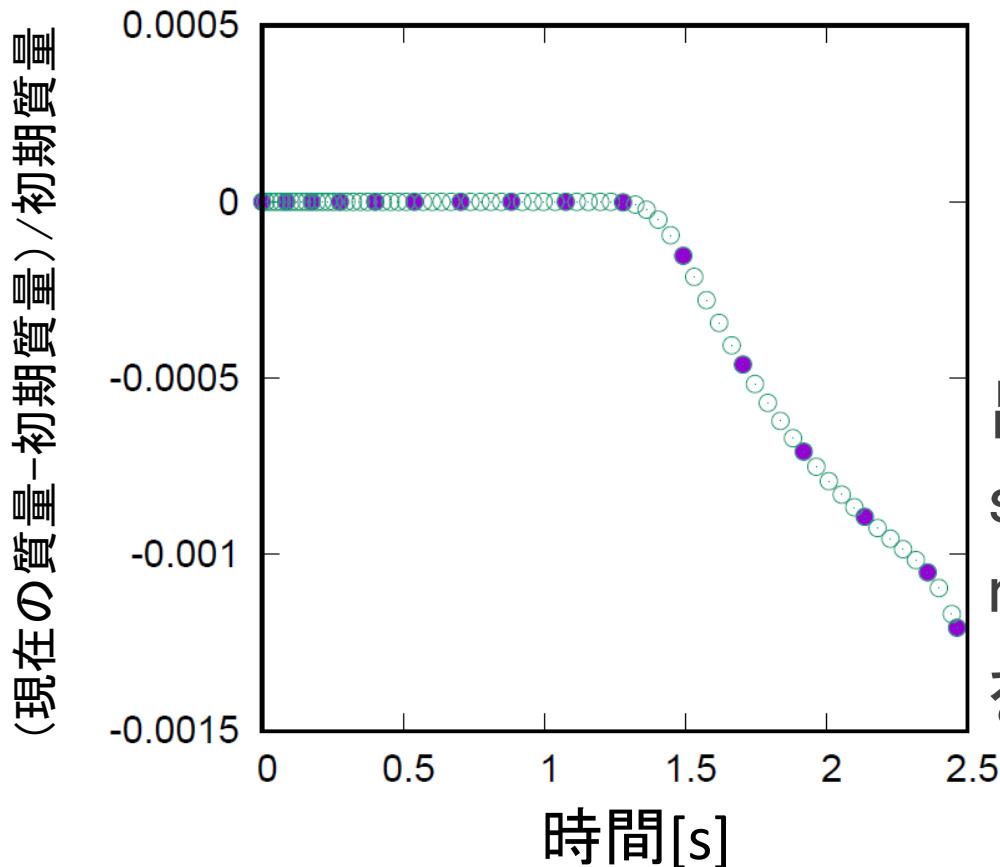
```
$ gnuplot
```

以下のコマンドを入力

```
plot "mass.txt0" u 1:3
```

# 実習2:出力する頻度の変更

▪ 例: demo2Dのmass.txt0の一行目と三行目



出力頻度が  
10ステップに1回に増加

出力自体を変更するなら  
src/shared/以下の  
mod\_conservation.F90  
を変更する

# 実習3:端末への時間表示

- ・端末への計算時間とCPU時間の表示  
適当な時間間隔で計算時間を端末に表示するよう  
出力文を追加する

```
iSALE2D.F90
```

```
初期状態を計算  
do while(計算時間<TEND)
```

```
結果の出力&保存  
タイムステップの計算  
様々な物理量の計算
```

```
end do  
計算終了の処理
```

出力用の時間を宣言

```
出力用の時間にdtを足す  
if(出力用の時間>0.1) then  
計算時間を出力する文  
出力用の時間をリセット  
endif
```



## 実習3:端末への時間表示

- time, time\_cpuはiSALE2D.F90内ですでに宣言済み
- dtもモジュール内で宣言されている
- time\_outputのみ新たに宣言する

```
real :: time_output=0.0
```

```
time_output = time_output + dt
```

```
if(time_output>0.1) then
```

```
    write(*,*) 'TIME[s]=', time, 'CPUTIME[s]=', time_cpu
```

```
    time_output = 0.0
```

```
endif
```



```
PROGRAM ISALE
```

```
USE MOD_ISALE, only : dumpname, &  
    out, ioerr, STDOUT, ioinit, ioener, iomass, iostep, &  
    time, tend, tlim, ale_mode, lagrangian, eulerian, &  
    tracer_motion, TR_VEL, stop_after_setup, verbose, vl, ncy, eps_min, dt  
use ptool_interface, only : io_out, io_oin, ptool_text, ptool_text_int  
use mod_parallel  
use mod_strength_routines, only : check_state_all  
use mod_identify_mat  
use mod_regrid, only : dynamic_regrid  
implicit none  
real :: time_start !< Wall time of simulation start  
real :: time_now    !< Wall time now  
real :: time_cpu    !< Duration of simulation (wall time)  
real :: time_output = 0.0 !---R.S.
```

```
! Compute cpu time since job began  
call CPU_TIME(time_now)  
time_cpu = time_now-time_start
```

```
!+++++  
time_output = time_output + dt !---R.S.  
if(time_output>0.1)then !---R.S.  
    write(*,*) 'TIME[s]=', time, 'CPU TIME[s]=', time_cpu !---R.S.  
    time_output = 0.0 !---R.S.  
end if !---R.S.  
!+++++
```

```
end do
```

```
! Finish simulation...  
call savedata  
call write_dump(1) ! Make data dump on exit  
call finalcycle(time_cpu)  
call isale_finalize("<<<END OF SIMULATION>>>")
```

```
END PROGRAM ISALE
```



# 実習3:端末への時間表示

```
SETTINGS FINISHED.....START JOB

TIME[s]= 0.10155292062315720      CPU TIME[s]= 2.05200005
TIME[s]= 0.20169808489705784      CPU TIME[s]= 3.95199990
TIME[s]= 0.30301174513443768      CPU TIME[s]= 5.61999989
TIME[s]= 0.40357199351875195      CPU TIME[s]= 6.98799992
TIME[s]= 0.50510114347969681      CPU TIME[s]= 8.29599953
TIME[s]= 0.60581518550657010      CPU TIME[s]= 9.47599983
TIME[s]= 0.70693318863838173      CPU TIME[s]= 10.5599995
TIME[s]= 0.80720307829126348      CPU TIME[s]= 11.5880003
TIME[s]= 0.91020272480366005      CPU TIME[s]= 12.6199999
TIME[s]= 1.0105720429103993       CPU TIME[s]= 13.5839996
TIME[s]= 1.1115367795138853       CPU TIME[s]= 14.5880003
TIME[s]= 1.2127105170616108       CPU TIME[s]= 15.5439997
TIME[s]= 1.3141941368832579       CPU TIME[s]= 16.4759998
TIME[s]= 1.4178545230496820       CPU TIME[s]= 17.4440002
TIME[s]= 1.5210108024769766       CPU TIME[s]= 18.4080009
TIME[s]= 1.6250520907059414       CPU TIME[s]= 19.3720016
TIME[s]= 1.7256110494423331       CPU TIME[s]= 20.3439999
TIME[s]= 1.8287578784077527       CPU TIME[s]= 21.3240013
TIME[s]= 1.9290049996850289       CPU TIME[s]= 22.3040009
TIME[s]= 2.0326928220552705       CPU TIME[s]= 23.2520008
TIME[s]= 2.1331031904730793       CPU TIME[s]= 24.2400017
TIME[s]= 2.2367526772855197       CPU TIME[s]= 25.2000008
TIME[s]= 2.3412216656210267       CPU TIME[s]= 26.1680012
TIME[s]= 2.4456811493628536       CPU TIME[s]= 27.2639999
```



# まとめ

- ・iSALE2Dの小さな改良は比較的簡単にできる
- ・今回は割愛したが出力先の変更やテキストでの出力も出力先の番号にさえ注意すれば可能  
([shared/mod\\_io.F90](#)を参照)

*iSALEの開発者であるGareth Collins, Kai Wünnemann, Boris Ivanov, H. Jay Melosh, Dirk Elbeshausenの各氏に感謝致します*