Exercise 1: Blast waves and the ISM

Troels Haugbølle, Jon Ramsey, and Neil Vaytet

In this exercise you will be using the `RAMSES` code to model blast waves in the interstellar medium (ISM).

Prerequisites:

You will need the following tools:

- `git` for downloading of material and code
- `gfortran` or another Fortran compiler for compiling RAMSES
- The `RAMSES` code for running your simulations
- `python` and `osiris` for data analysis

`git`, `gfortran`, and `python` should all be installed as part of your system installation your laptop. Follow the instructions on the homepage:

https://indico.nbi.ku.dk/internalPage.py?pageId=9&confId=933

if you didn't do it already.

To keep things organised, make a folder where you will keep everything related to this exercise (for example):

```
mkdir PhD-School-2017-exercise-1
```

Go to the folder and download the necessary material to start the exercise by following the next three steps. You may already have done some of it as part of the code clinic yesterday.

Obtain the exercise material:

- Download it via git from bitbucket:
  `git clone https://bitbucket.org/thaugboelle/NBI-PhD2017-ex-1.git`

Obtain the RAMSES code:

- It is publicly available and can be downloaded using git from bitbucket:
  `git clone https://bitbucket.org/rteyssie/ramses.git`

Obtain the osiris plotting tools

- It is publicly available and can be downloaded using git from bitbucket:
  `git clone https://bitbucket.org/nvaytet/osiris.git`

To have access to `osiris`, add it to your PYTHONPATH shell variable (this has to be repeated in every new shell you start up). If, for example, the full path to where osiris is stored is called

```
/home/user/PhD-School-2017-exercise-1/osiris
```

Then add it to the PYTHONPATH:

```
export PYTOHNPATH=/home/user/PhD-School-2017-exercise-1/osiris:$PYTHONPATH
```

Blast waves

Releasing a large amount of energy into a background fluid creates an explosion characterized by a strong shock wave, which then expands radially outwards from the point where the energy was released. In astrophysics, this happens, for example, in the case of supernovae.

As you have already heard in the morning lecture, this is called a Sedov-Taylor blast wave. Sedov and Taylor first solved the problem in the context of atomic bomb explosions, but the solution is also useful in astrophysics. It can be used to test and validate hydrodynamical computer codes since an analytical solution exists, and is also a good illustration of the power of the adaptive mesh refinement (AMR) technique. Even without the exact analytical solution, one can already test the correctness using the scaling behaviour of the expanding shock.

In the morning lecture, you learned that the radius of an expanding blast wave in a homogeneous medium scales as

$$\text{Radius} \sim \text{Energy}^{1/5}\ \text{Density}^{-1/5}\ \text{Time}^{2/5}$$

In the first part of this exercise, you will run `RAMSES` with a Sedov-Taylor blast wave set up in two dimensions and test this scaling behaviour.

Blast waves, Part I: Scaling behaviour and structure

Once you have the exercise repository, `osiris`, and `RAMSES` available locally, go to the folder where you have stored exercise 1. You should see the following subfolders:

```
PhD-School-2017-exercise-1% ls
NBI-PhD2017-ex-1  osiris  ramses
```

Inside the exercise folder, you have the files needed to carry out the exercise:

```
PhD-School-2017-exercise-1% ls NBI-PhD2017-ex-1
Makefile partII ramses_ug.pdf  sedov2d.nml  sedov2d.py
```

Note that the PDF file `ramses_ug.pdf` contains the official `RAMSES` documentation. First, you have to compile `RAMSES` for hydrodynamical experiments in two dimensions. To accomplish this, we change to the `bin` subfolder of the `RAMSES` installation:

```
PhD-School-2017-exercise-1% cd ramses/trunk/ramses/bin
PhD-School-2017-exercise-1/ramses/trunk/ramses/bin%
```

Then copy the `Makefile` from the exercise folder to the `bin` subfolder:

```
...nk/ramses/bin% cp ../../../../NBI-PhD2017-ex-1/Makefile .
```

You can see which changes have been made compared to the public version by using the `git diff` command

```
...nk/ramses/bin% git diff Makefile
diff --git a/trunk/ramses/bin/Makefile b/trunk/ramses/bin/Makefile
index f56fcd0..460f259 100644
--- a/trunk/ramses/bin/Makefile
+++ b/trunk/ramses/bin/Makefile
@@ -3,7 +3,7 @@
 ########################################################################
 # Compilation time parameters
 NVECTOR = 32
-NDIM = 1
+NDIM = 2
 NPRE = 8
 NVAR = 5
 NENER = 0
```

You see that we have changed the `Makefile` so that `RAMSES` will be compiled for two dimensional simulations. If you use the `gfortran` compiler you are now ready to compile the code. However, if you use another compiler you will have to open the `Makefile` in an editor, comment out the lines setting the compiler, and comment in the other compiler (normally the Intel `ifort` compiler). Now you can compile the code:

```
...nk/ramses/bin% make
```

This will provide a lot of output about the compilation. If everything is successful it should end with a (long) linking line similar to:

```
gfortran -O3 -frecord-marker=4 -fbacktrace -ffree-line-length-none -g amr_parameters.o amr_commons.o random.o pm_parameters.o pm_commons.o poisson_parameters.o poisson_commons.o hydro_parameters.o hydro_commons.o cooling_module.o bisection.o sparse_mat.o clfind_commons.o gadgetreadfile.o write_makefile.o write_patch.o write_gitinfo.o read_params.o init_amr.o init_time.o init_refine.o adaptive_loop.o amr_step.o update_time.o output_amr.o flag_utils.o physical_boundaries.o virtual_boundaries.o refine_utils.o nbors_utils.o hilbert.o load_balance.o title.o sort.o cooling_fine.o units.o light_cone.o movie.o init_hydro.o init_flow_fine.o write_screen.o output_hydro.o courant_fine.o godunov_fine.o uplmde.o umuscl.o interpol_hydro.o godunov_utils.o condinit.o hydro_flag.o hydro_boundary.o boundana.o
```

read_hydro_params.o synchro_hydro_fine.o init_part.o output_part.o rho_fine.o synchro_fine.o move_fine.o newdt_fine.o particle_tree.o add_list.o remove_list.o star_formation.o sink_particle.o feedback.o clump_finder.o clump_merger.o flag_formation_sites.o init_sink.o output_sink.o init_poisson.o phi_fine_cg.o interpol_phi.o force_fine.o multigrid_coarse.o multigrid_fine_commons.o multigrid_fine_fine.o multigrid_fine_coarse.o gravana.o boundary_potential.o rho_ana.o output_poisson.o ramses.o -o ramses2d
rm write_makefile.f90
rm write_patch.f90
..nk/ramses/bin%

The `RAMSES` code executable is called "`ramses2d`". You can now copy this executable to the exercise folder and go there:

..nk/ramses/bin% cp ramses2d ../../../../NBI-PhD2017-ex-1/
..nk/ramses/bin% cd ../../../../NBI-PhD2017-ex-1/
PhD-School-2017-exercise-1/NBI-PhD2017-ex-1%

This step by step procedure:

- go to the `bin` sub-folder of `RAMSES;`
- change or copy the `Makefile;`
- make a new executable with the command `make;`
- copy the executable to the folder where you will run `RAMSES`,

has to be followed every time you need to produce a new executable. Note that before compiling a new executable, it is good practice to first have to clean up from the last time by invoking "`make clean`" in the `bin` subfolder. In this exercise you can reuse your executable for all substeps, but in the Thursday exercise you will have to create a new `RAMSES` executable with support for three dimensions and star formation feedback.

To run the code you need an input text file. An input file contains small blocks of code, called namelists, that specify parameters for the code.

For this first exercise, we have prepared a ready made namelist called `sedov2d.nml`. An example of such an input parameter block is (from `sedov2d.nml`):

```
&INIT_PARAMS
nregion=2
region_type(1)='square'
region_type(2)='point'
x_center=0.5,0.0
y_center=0.5,0.0
length_x=10.0,1.0
length_y=10.0,1.0
exp_region=10.0,10.0
d_region=1.0,0.0
u_region=0.0,0.0
v_region=0.0,0.0
p_region=1e-5,0.4
/
```

It specifies that the initial conditions for the experiment contain two regions: A "`square`" extended region (that fills the whole box), and a "`point`" region at the corner (0,0,0) with a much higher pressure (`p_region=0.4`) compared to the surrounding medium, which has a pressure of 10^{-5}. You can read more about the namelists and the options in chapter 3 of the official `RAMSES` user guide. A copy is stored in the exercise folder. The file is called `ramses_ug.pdf`.

You are now ready to execute the code. To execute the code, type:

```
PhD-School-2017-exercise-1/NBI-PhD2017-ex-1% ./ramses2d sedov2d.nml
```

It should take somewhere between 30 seconds and 2 minutes to run the code on a reasonably modern laptop. You will see *a lot* of output scrolling across the screen while the code runs. It basically gives the status of how many cells are in use at each AMR refinement level at a given timestep, what is the current time, how much memory is in use, what is the size of the timestep etc. See Chapter 2 in the user guide for a description of what some of this output means. At the end you will see something like this:

```
Level  1 has         1 grids (      1,      1,      1,)
Level  2 has         4 grids (      4,      4,      4,)
Level  3 has        16 grids (     16,     16,     16,)
Level  4 has        64 grids (     64,     64,     64,)
Level  5 has       177 grids (    177,    177,    177,)
Level  6 has       425 grids (    425,    425,    425,)
Level  7 has       917 grids (    917,    917,    917,)
Level  8 has      1832 grids (   1832,   1832,   1832,)
Level  9 has      3311 grids (   3311,   3311,   3311,)
Level 10 has      4652 grids (   4652,   4652,   4652,)
Main step=   1483 mcons=-6.66E-16 econs= 8.55E-15 epot= 0.00E+00 ekin= 2.50E-01
Fine step=   2966 t= 1.00066E+00 dt= 7.496E-04 a= 1.000E+00 mem=12.3%
Run completed

   seconds        %    STEP (rank=      1)
     0.107      0.2    coarse levels
     1.125      2.4    refine
     0.157      0.3    load balance
     0.273      0.6    io
     1.446      3.1    courant
     0.391      0.8    hydro - set unew
    37.995     82.1     hydro - godunov
     0.321      0.7    hydro - set uold
     0.531      1.1    hydro upload fine
     3.834      8.3    flag
    46.254    100.0    TOTAL
Note: The following floating-point exceptions are signalling: IEEE_UNDERFLOW_FLAG
IEEE_DENORMAL
..I-2017-exercise-1/NBI-PhD2017-ex-1%
```

Note also that `output_XXXXX` sub-folders have been created where data from the run is stored

```
..l-2017-exercise-1/NBI-PhD2017-ex-1% ls
Makefile      output_00001    output_00002    output_00003    output_00004    output_00005
output_00006  partII          ramses2d        ramses_ug.pdf   sedov2d.nml     sedov2d.py
```

You can use the provided python script, "`sedov2d.py`", to analyse the data. It loops over a variable `nout` that indicates which snapshot will be analysed. The rest of the script is commented and should be relatively self-explationary. You can adapt the range for `nout` if you add snapshots. We recommend you also look at the osiris home page for a description of the capabilities of this package. It can be found here:

https://bitbucket.org/nvaytet/osiris/wiki/Home

To execute the python analysis routine either load it in your favourite python development environment or execute it from the command line:

```
..l-2017-exercise-1/NBI-PhD2017-ex-1% python sedov2d.py
```

If you get the output

```
Traceback (most recent call last):
  File "sedov2d.py", line 3, in <module>
    import osiris as pp
ImportError: No module named osiris
```

It is because you forgot to add osiris to the PYTHONPATH variable! See above and check your current environment with "echo $PYTHONPATH". Upon successful execution, you will see some basic printout on the terminal about the code ending with:

```
Time=1.0006641446 Max density=19.120367253 at radius=0.999657810792
```

In addition, PDFs "`sedov_XX.pdf`" are produced that visualise the structure of the fluid at the different time steps. The setup is a point explosion at (0,0) and reflecting boundaries at all edges of the box. This allow us to save a factor of 4 in computational time compared to running with an explosion at the center of the box. You can read more about boundary conditions in Sections 3.6 and 5.4 of the user guide.

Q1: Dimensional analysis

You have now had a very thorough walkthrough of how to compile and run `RAMSES`, and analyse the output. We would like to test if we can reproduce the scaling of the shock radius that we saw this morning in the lecture. To do this, execute the analysis script and make note of the radius of the shock as a function of time (which is printed to the terminal). Does it match the expectation, or else how does it scale with time? Produce a plot of the shock radius as a function of time. Check in the output PDFs that the printout actually matches the edge of the shock and that the maximum of the density is not at some other place.

The dimensional analysis assumes a blast wave expanding in a three dimensional medium with a density having dimensions (Mass / Length^3). But we are running in two dimensions. Derive the scaling law for two dimensions by figuring out the unique combination of time (T), density (M / L^2), and energy (M L^2 / S^2) that gives the appropriate length scale. Does this now match with the shock radius observed in the simulation?

Q2: Adding Mass

A supernova exploding in the interstellar medium does not only have a high pressure, it also contains a significant amount of mass. To simulate this, try adding mass to your experiment. Start by making a new sub-folder, copy in the needed files to run a new experiment, and then change to the folder:

```
..I-2017-exercise-1/NBI-PhD2017-ex-1% mkdir Q2
..I-2017-exercise-1/NBI-PhD2017-ex-1% cp ramses2d sedov2d.nml sedov2d.py Q2/
..I-2017-exercise-1/NBI-PhD2017-ex-1% cd Q2
..017-exercise-1/NBI-PhD2017-ex-1/Q2%
```

Edit the namelist file to add mass to the experiment by changing the `d_region` in the `INIT_PARAMS` block of the namelist:

```
d_region=1.0,0.0
```

The second number in the line indicates how much additional density there is at the point of explosion. Add mass by changing 0.0 to something reasonable, like 0.4. Run the simulation followed by the analysis again. What is the impact of adding mass? How does the expansion change? How does the radius as a function of time change? What about the structure of the shock: Does it look similar to what you saw in the first run? If not, why not?

Q3: Playing with quality of the solution

The reverse shock of a supernova blast wave is Rayleigh-Taylor (R-T) unstable. This is not readily obvious in the solution you found in Q2. To resolve the instability, you either need a higher quality solution or a higher resolution model. The quality of the solution is controlled by parameters in the `HYDRO_PARAMS` namelist block. In particular, the type of Riemann solver (`riemann`) impacts how well R-T plumes are resolved, and the slope interpolation

from cell centers to cell interfaces (`slope_type`), where the fluid dynamics is solved, impacts the sharpness of the shocks.

Look in the RAMSES user guide and find the section about HYDRO_PARAMS. The default option is to use the `LLF` (local Lax-Friedrichs) solver and a very conservative slope interpolation, which gives a stable, but highly diffusive solution. Other options are to increase the resolution by adding more refinement levels (`levelmax`), or change the refinement strategy (see section 3.10 in the RAMSES user guide).

Make a new subfolder, e.g., Q3, and copy `sedov2d.nml`, `sedov2d.py`, and `ramses2d` from Q2 to Q3. Play around with the Riemann solver and the slope interpolation, and store the PDFs (`sedov_XX.pdf`) and namelist file (`sedov2d.nml`) in subfolders while you try out different options so that you can compare the results later. Try using more accurate solvers, like “exact” or “hllc”, increase the value of `slope_type` and/or increase the resolution. Are you able to spatially resolve the instabilities? At what cost (i.e. time to solution) relative to the reference solution (i.e. Q2)?

This illustrates how the number of cells is not always a good gauge for the quality of a numerical solution to a problem, and how the AMR strategy is something that you choose. Indeed, AMR can have a large impact on the quality of (certain aspects of) the solution to a problem, even though this may be difficult to quantify.

Blast waves, Part II: Explosions in an inhomogeneous medium

Real supernovae can explode both inside and outside molecular clouds, depending on the lifetime of the progenitor. The most massive stars may live only for a few million years and do not have time to move far from their natal site, whereas the majority of supernovae have progenitor masses ~8 M_{Sun} and supernovae delay times of up to 40 Myr. Therefore, prior to a star going supernova, the molecular cloud that the star formed in could have dispersed, or the star may have moved beyond the cloud. It is therefore not entirely clear how well correlated supernovae explosions are with molecular clouds.

To investigate this, in this part of the exercise we will place blast waves at different locations relative to a (very) simplified molecular cloud, in pressure equilibrium with a surrounding, low density ambient medium. The density ratio of the cloud with respect to the ambient medium is chosen to be 100.

RAMSES has a simple mechanism for setting up regions in space. We already used it in Part I of the exercise to combine a homogeneous medium with a point explosion. It is controlled by the `INIT_PARAMS` namelist block.

In Part II, you will use it to set off supernovae in an inhomogeneous medium and look at the impact on the simplified molecular cloud. To begin, go to the “`partII`” subfolder and copy the executable `ramses2d` from Part I of the exercise to this folder.

Inside the folder is yet another namelist file called `sedov2d.nml`. If you compare it with the namelist from the Part I, you will notice that there are now 3 regions:

```
&INIT_PARAMS
nregion=3
region_type(1)='square'
region_type(2)='square'
region_type(3)='point'
x_center=0.5,0.5,0.6
y_center=0.5,0.0,0.0
length_x=10.0,0.4,1.0
length_y=10.0,0.4,1.0
exp_region=10.0,2.,10.0
d_region=0.1,10.0,0.0
u_region=0.0,0.0
v_region=0.0,0.0
p_region=1e-5,1e-5,0.4
/
```

The three regions are: the outside medium, the cloud, and the explosion, respectively. Note that, in the current setup, the explosion will occur inside the cloud. The `length_x` and `length_y` parameters indicate the *diameter* of the cloud in the x- and y-directions. The radius of the cloud is 0.3/2 = 0.15, while the explosion is displaced 0.05 from the center of the cloud.

Run the code with the new namelist and describe what happens. For example, how does it compare with a blast wave in a single medium? Is it still possible to plot the distance of the shock wave as a function of radius in a simple fashion?

Create a subfolder called "`inside`" and move the output folders, PDF files and a copy of the input file there. Next, modify the original input file and place the explosion on the *edge* of the cloud. Run the code again and store the results in another new folder, "`edge`". Finally, place the explosion outside the cloud and run the code one more time. In these two cases, how does the explosion develop relative to when it was "`inside`" the cloud. For that matter, how is the cloud impacted differently in the three cases? Note that, in the last two cases, you may have to lower the end time (i.e. the time of the final output) to 0.05 in order to have the runs finish in a reasonable amount of time (e.g. < 1 hour). Or use two laptops to run the setups "in parallel"! ;D

Note that, when loading data using `osiris`, one can specify the origin for the plots via the "`center`" keyword. In order for the plot of density as a function of radius to remain sensible, remember to change the position of the origin in `sedov2d.py` when moving the center of the blast wave.

On Thursday, you will set up exploding supernovae at a much lower resolution, but with an otherwise more realistic setup. If you are interested, you can also have a look at
http://adsabs.harvard.edu/abs/2015A%26A...576A..95I
where a more realistic initial structure for the cloud is used.