Computational Astrophysics Report 2b

Jakob Schauser, pwn274

December 2023

1 Complete the 2-D LLF and HLL solvers (20p)

The important parts of the implementation here below:

```
def primitive_to_conservative(q):
    # ...
    if q.gamma!=1.0:
        U.Etot = q.P/(q.gamma-1.) + 0.5*q.D*(q.vperp)**2 + 0.5*q.D*(q.vpar)**2

def Hydro_Flux(q,U):
    # ...
    if q.gamma!=1.0:
        F.Etot = (U.Etot + q.P) *(q.vperp)

def LLF(ql,qr):
    # ...
    if ql.gamma!=1.0:
        Flux.Etot = 0.5*(Fl.Etot + Fr.Etot - cmax*(Ur.Etot - Ul.Etot))
```

The HLL looks as follows:

```
def HLL(ql,qr):
    # sound speed for each side of interface (l==left, r==right), and maxmimum sound speed (
    c_max)
    c_left  = (ql.gamma*ql.P/ql.D)**0.5
    c_right = (qr.gamma*qr.P/qr.D)**0.5
    c_max = np.maximum (c_left, c_right)

    # maximum wave speeds to the left and right (guaranteed to have correct sign)
    SL = np.minimum(np.minimum(ql.vperp,qr.vperp)-c_max,0) # <= 0.
    SR = np.maximum(np.maximum(ql.vperp,qr.vperp)+c_max,0) # >= 0.

  # Hydro conservative variable
    Ul = primitive_to_conservative(ql)
    Ur = primitive_to_conservative(qr)

    # Hydro fluxes
    Fl = Hydro_Flux(ql,Ul)
    Fr = Hydro_Flux(qr,Ur)

    # HLL flux based on wavespeeds. If SL < 0 and SR > 0 then mix state appropriately
    # The general form is
    #    (SR * F_left - SL * F_right + SL * SR *(U_right - U_left)) / (SR - SL)
    # where U is the state vector of the conserved variables
    Flux = empty_class()
    Flux.D = (SR*Fl.D - SL*Fr.D + SL*SR*(Ur.D - Ul.D)) / (SR - SL)
    Flux.Mperp = (SR*Fl.Mperp - SL*Fr.Mperp + SL*SR*(Ur.Mperp - Ul.Mperp)) / (SR - SL)
    Flux.Mpar  = (SR*Fl.Mpar  - SL*Fr.Mpar  + SL*SR*(Ur.Mpar  - Ul.Mpar )) / (SR - SL)
    if ql.gamma != 1:
       Flux.Etot = (SR*Fl.Etot - SL*Fr.Etot + SL*SR*(Ur.Etot - Ul.Etot)) / (SR - SL)
    return Flux
```

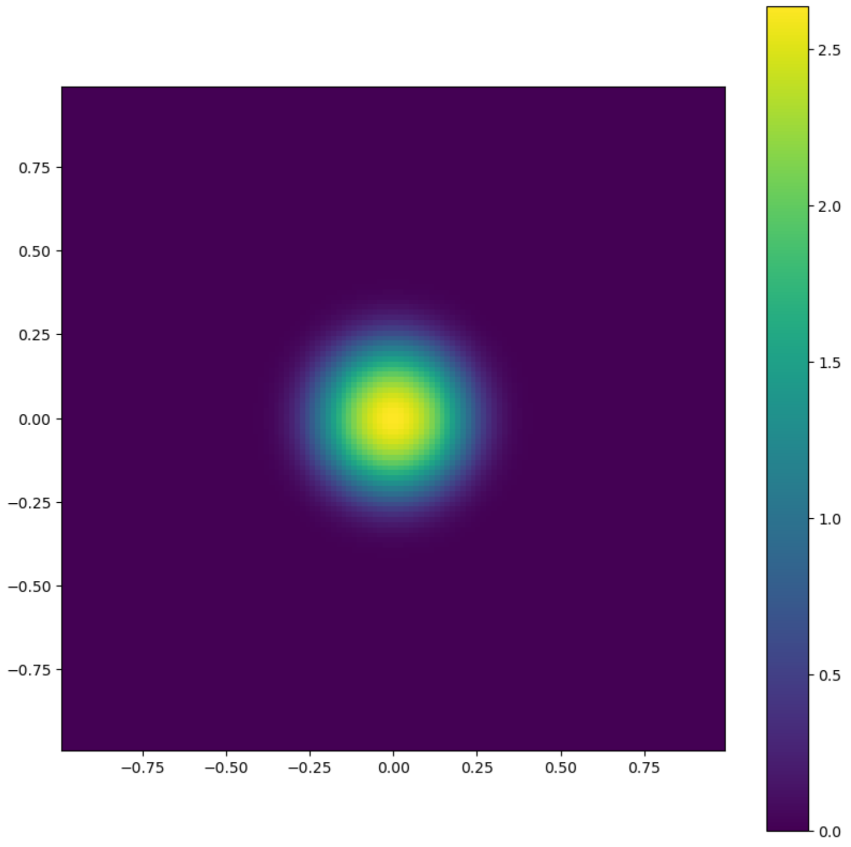
2 Add a profile that defines the blast (10p)

Defining the profile in code:

```
def __init__(exp,n=64,gamma=1.4,e0=1e3,d0=0.0,power=2,w=3.,eps=0.05):
        hd.__init__(exp,n)
        exp.gamma = gamma
        exp.D  = np.ones((n,n))

        dr = exp.ds   # grid size
        profile = e0*np.exp(-np.power(exp.r/(w*dr),power)) # the exponential part
        B = np.sum(profile)*dr*dr # the normalization factor
        exp.Etot = np.ones((n,n)) + profile / B
```

This profile looks as follows:



3 Scaling relations (20p)

Doing a dimensional analysis,

$$t^\alpha \cdot \Sigma^\beta \cdot E^\gamma = L^1 \leftrightarrow [s]^\alpha \cdot [kg/m^2]^\beta \cdot [kgm^2/s^2]^\gamma = m^1$$

This corresponds to solving the following system of equations:

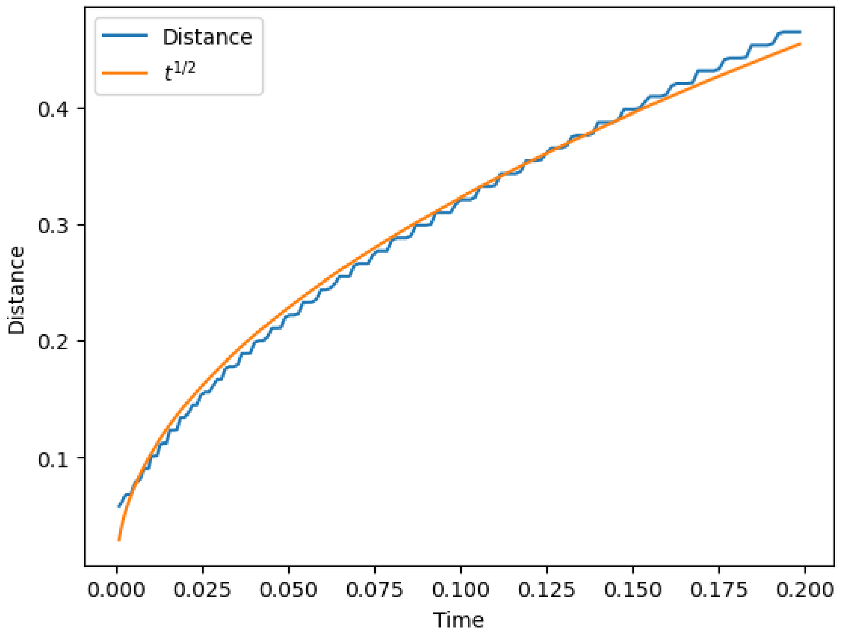
$$\begin{aligned} \beta + \gamma &= 0 \\ -2\beta + 2\gamma &= 1 \\ \alpha - 2\gamma &= 0 \end{aligned}$$

Giving us the only solution:

$$E^{1/4}\rho^{-1/4}t^{1/2} \propto L \tag{1}$$

We wrote up a way to estimate the width by finding the N biggest peaks and averaging over their locations.

Compared to the data, the dimensional analysis gave us the following:

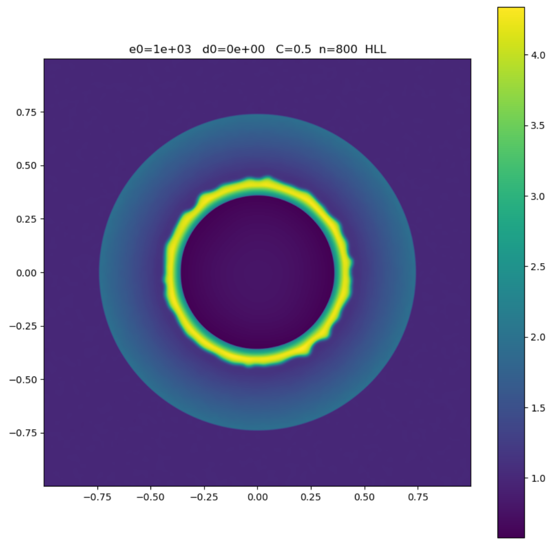


It seems like our experiment somewhat reasonably fit the theoretical prediction. And when we plot the found peaks (red crosses), it also looks reasonable (Appendix 1)

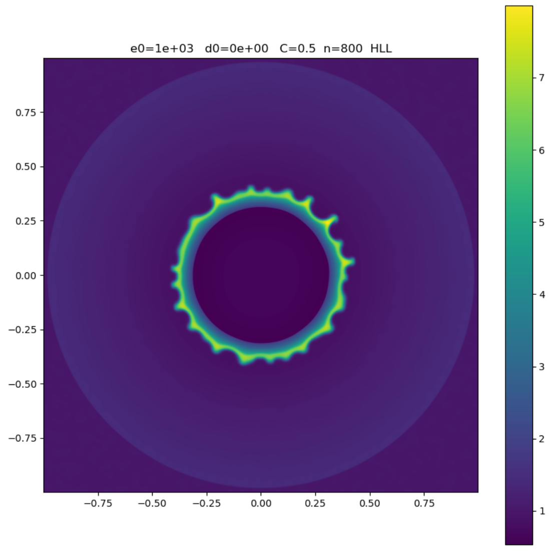
4 Rayleigh-Taylor instabilities in blast waves (30p)

We added mass using the same profile as before. As we are running out of space, you can look at the `.ipynb` for the implementation. In general we see, that the expansion is slower, which makes sense, as the pressure has to push through it. The chock-front also has a radically different structure. This can be seen as the difference between `no_mass.mp4` and `with_mass.mp4` where you also might notice the random noise we have added to prime instabilities. In short the blast-wave is more diffusive (ie. less sharp/pronounced) and the wave is delayed when the simulation includes initial matter. In Appendix 2 the delay can also clearly be seen, using the analysis tool of Sec. 3.

When looking for instabilities got a lot of pretty pictures:



(a)



(b)

5 Extra Tasks for the interested

5.1 What happens if your blastwave collide with another one

In the attached video (`two_explosions.mp4`) it can be seen that their blast waves cross each other.

5.2 What happens if a blastwave explodes outside a "cloud" (e.g. a dense circular region)

As can be seen in the second video (`explosion_and_cloud.mp4`) the gas cloud is compressed and then blown away when the blast wave hits it.

5.3 What happens if it explodes on the edge, or inside?

The cloud is smeared out over the surface of the blast sphere. Again, this can be seen in the last video (`on_edge.mp4`)

Appendix

1

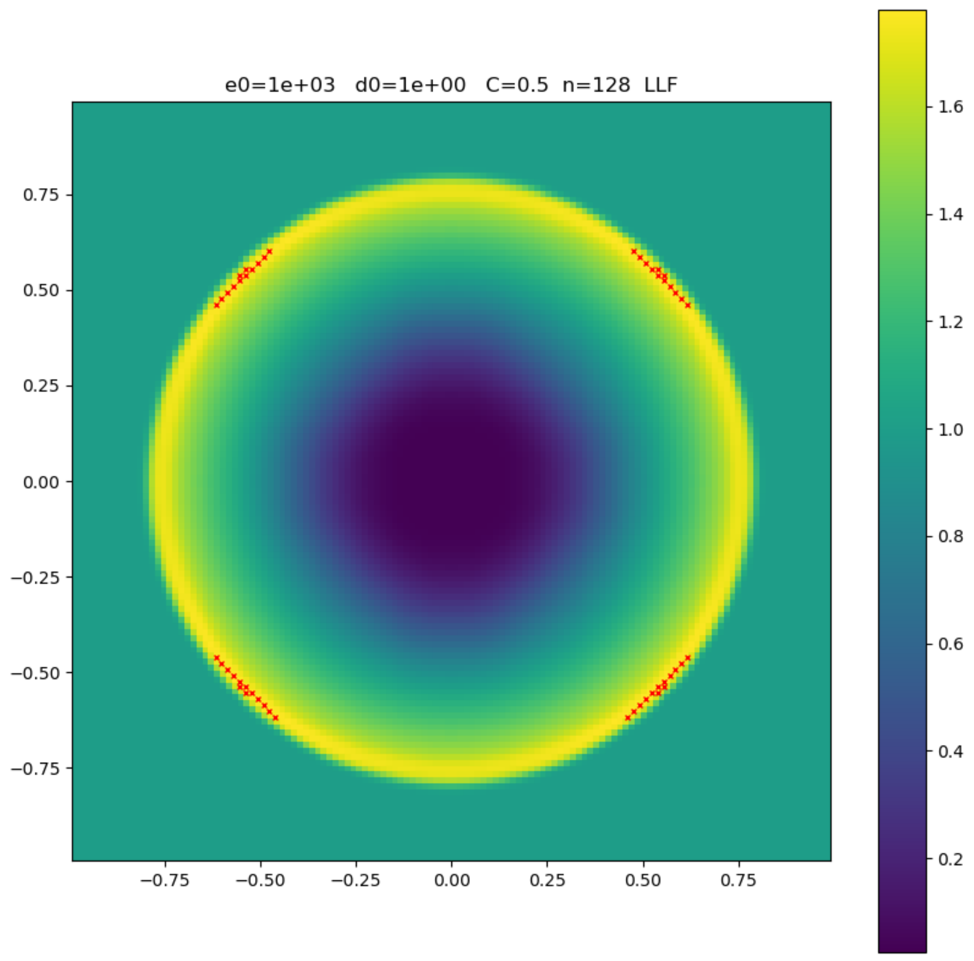


Figure 1: Plot of peak energy positions. This is done by numpy arg-maxing, so we have no idea what they are ”clumped together”

2

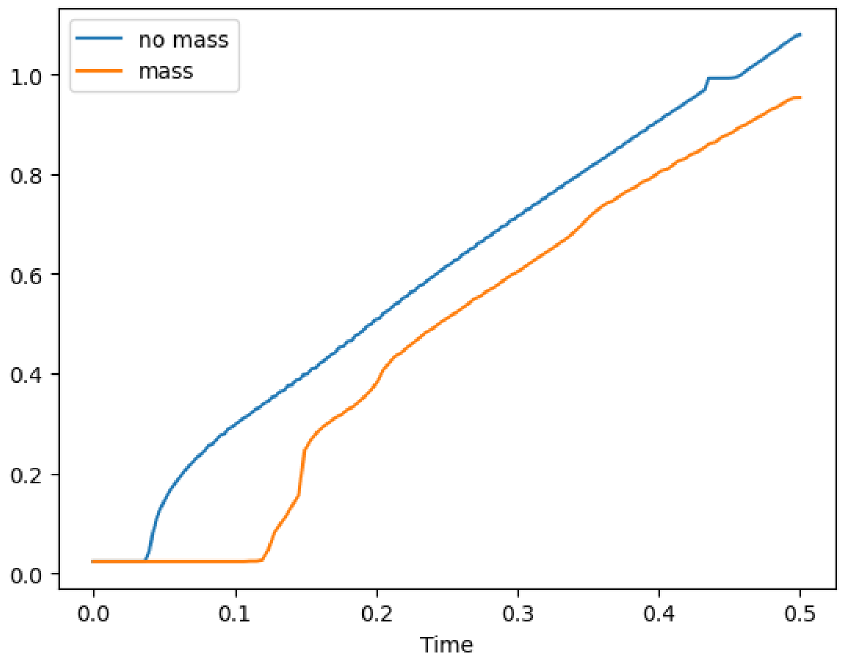


Figure 2: Plot of peak energy position over time