Frozen

March 28, 2020

1 Determine whether one needs to worry about frozen in ionization in a Python model

```
In [75]: import os
         os.getcwd()

Out[75]: '/Users/long/Projects/Python/bug407_frozen'

In [76]: from astropy.io import ascii
         import numpy as np
         import matplotlib.pyplot as plt
```

Note: One needs to run windsave2table on a model first, so one has access to the heating and cooloing

```
In [77]: root='cv_1e7'
         x=ascii.read(root+'.0.heat.txt')

In [78]: x.info()

<Table length=900>
      name        dtype
--------------- -------
              x float64
              z float64
              i   int64
              j   int64
         inwind   int64
       converge   int64
            v_x float64
            v_y float64
            v_z float64
            vol float64
            rho float64
             ne float64
            t_e float64
            t_r float64
              w float64
```

```
        heat_tot float64
       heat_comp float64
      heat_lines float64
         heat_ff float64
      heat_photo float64
      heat_auger float64
        cool_tot float64
       cool_comp float64
       lum_lines float64
         cool_dr float64
          lum_ff float64
         cool_rr float64
      cool_adiab float64
      heat_shock float64
heat_lines_macro float64
heat_photo_macro float64
```

The idea here is that the ionization/recombination time scale is about

$$t_c = \frac{3/2(n_e + n_{ion})kT_eVolume}{Cool_{tot}}$$

while the flow time scale is roughly

$$t_{flow} = \frac{\delta r}{v_r}$$

where δr is a characteric distance and v_r is the radial distance through the cell. A really gross way to do this would be simply to calculate δr from the center.

```
In [79]: k=1.38e-16
         nion=x['rho']/(1.26*1.67e-24)

         e=1.5*(x['ne']+nion)*k*x['t_e']*x['vol']
         e=np.array(e)
         i=0
         while i <len(e):
             if e[i]>0:
                 e[i]/=x['cool_tot'][i]

             i+=1

         x['Cool_time']=e

In [80]: x['v_r']=v_r=np.array(np.sqrt(x['v_x']*x['v_x']+x['v_z']*x['v_z']))

         n=0
         while n<len(x):
             if x['j'][n]==0:
```

```
            x['v_r'][n]=v_r[n]=v_r[n+1]
        n+=1
```

Take δr to be the geometric mean of the size of the cell in the x z directon, that is of the x-sectional area of a grid cell. This can be calculated from the volume of the cell divided by the circumference.x

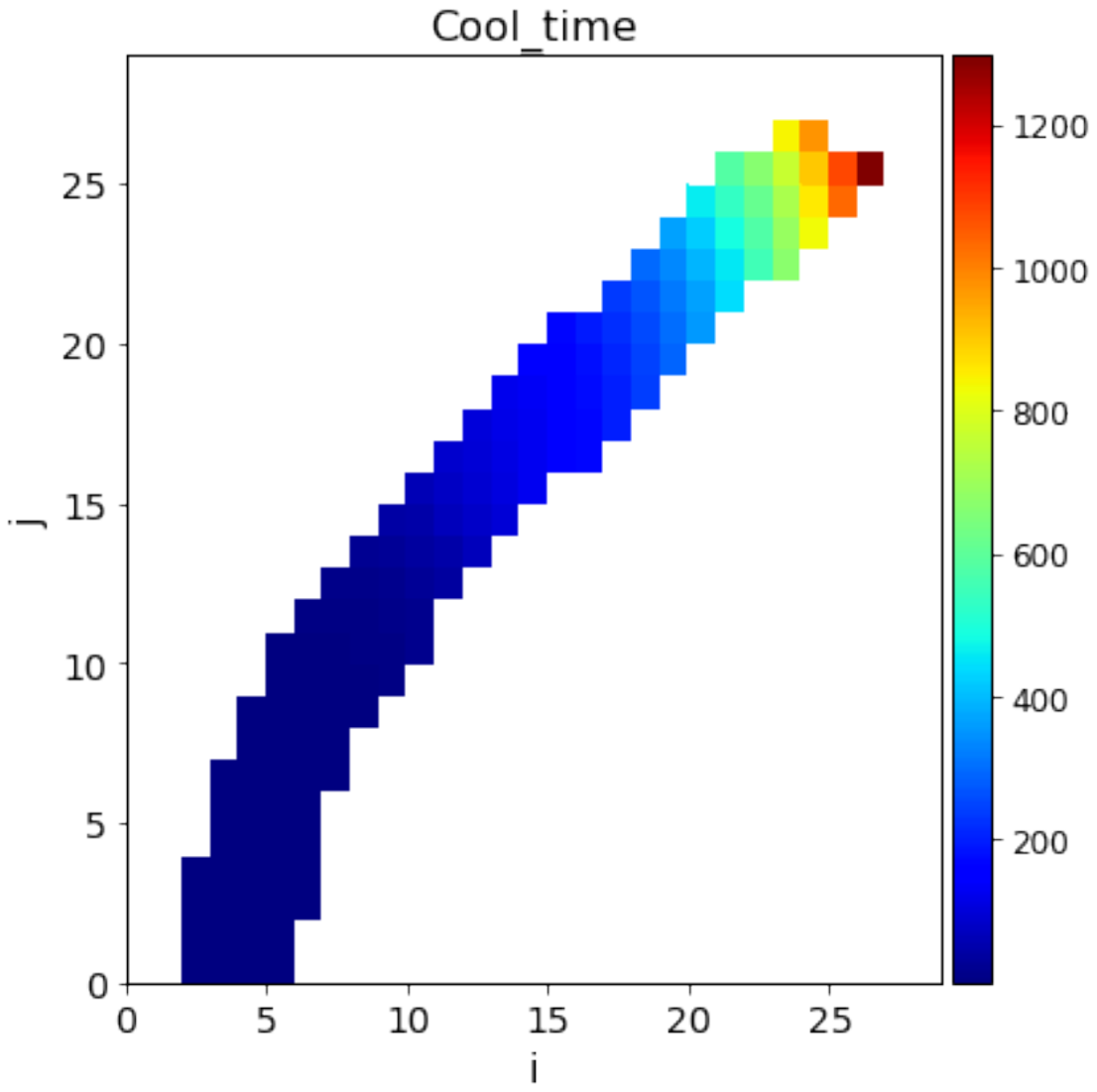
```
In [81]: delta_r=np.array(x['vol']/(2*3.14*x['x']))
         delta_r=np.sqrt(delta_r)

In [82]: x['Flow_time']=delta_r/v_r

In [83]: x['Ratio']=x['Flow_time']/x['Cool_time']

/Users/long/anaconda3/envs/astroconda/lib/python3.7/site-packages/ipykernel_launcher.py:1: Runti
  """Entry point for launching an IPython kernel.


In [84]: x.info()

<Table length=900>
      name        dtype  n_bad
--------------- ------- -----
              x float64     0
              z float64     0
              i   int64     0
              j   int64     0
        inwind   int64     0
       converge   int64     0
            v_x float64     0
            v_y float64     0
            v_z float64     0
            vol float64     0
            rho float64     0
             ne float64     0
            t_e float64     0
            t_r float64     0
              w float64     0
       heat_tot float64     0
      heat_comp float64     0
     heat_lines float64     0
        heat_ff float64     0
     heat_photo float64     0
     heat_auger float64     0
       cool_tot float64     0
      cool_comp float64     0
      lum_lines float64     0
        cool_dr float64     0
         lum_ff float64     0
```

```
         cool_rr float64     0
      cool_adiab float64     0
      heat_shock float64     0
heat_lines_macro float64     0
heat_photo_macro float64     0
       Cool_time float64     0
             v_r float64     0
       Flow_time float64     0
           Ratio float64   661
```

```
In [85]: for one in x:
             if one['vol']==0:
                 one['Flow_time']=0
                 one['Ratio'] = 0
         x.write('test.txt',format='ascii.fixed_width_two_line')

WARNING: AstropyDeprecationWarning: test.txt already exists. Automatically overwriting ASCII fil
```

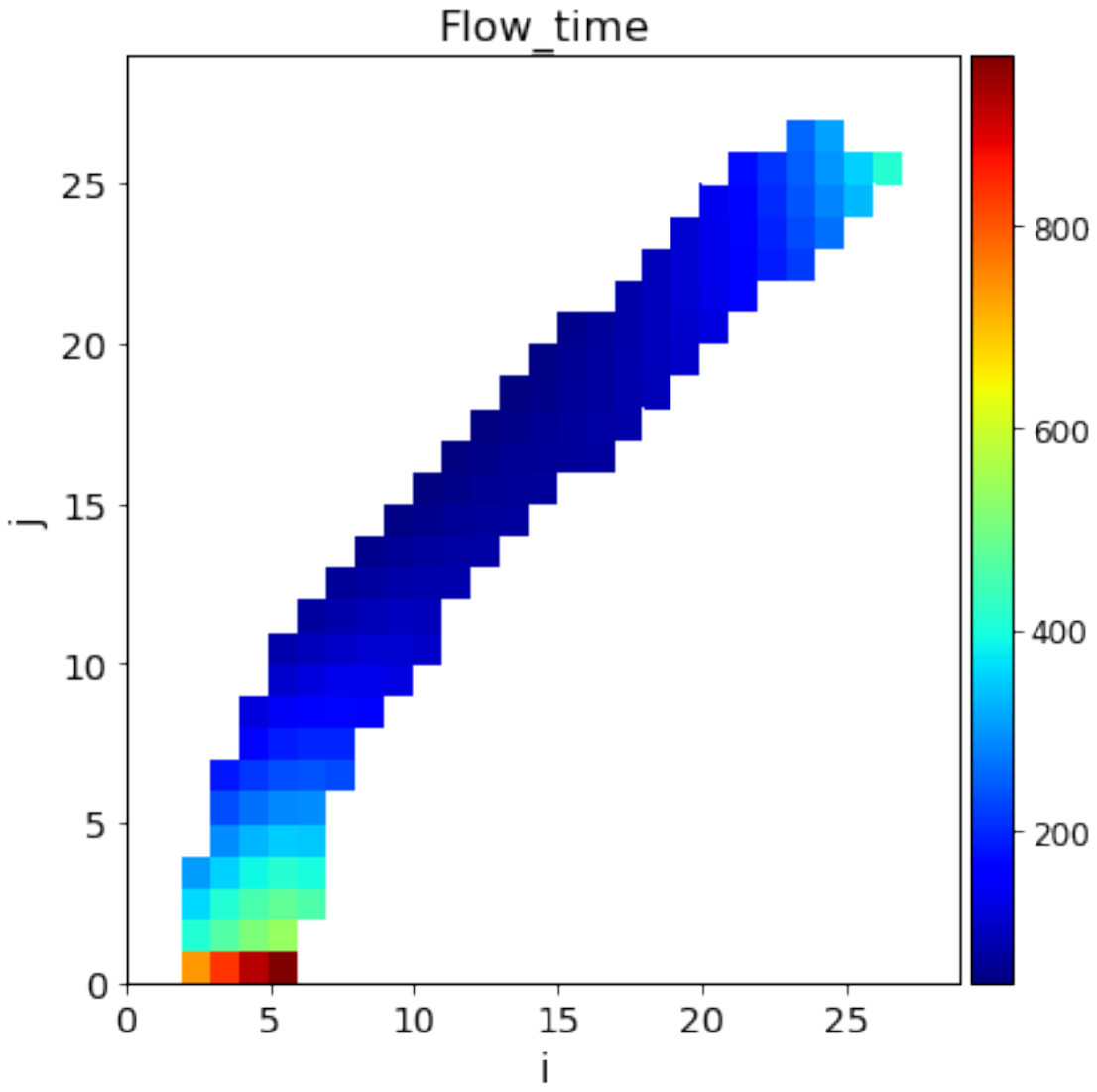
Now see if I can plot this for this model.
Note: plot_wind is in py_progs, and so should work if one has py_progs in one's path

```
In [86]: import plot_wind

In [87]: plot_wind.doit('test.txt',var='Cool_time')

Out[87]: 'test_Cool_time.png'
```




```
In [88]: plot_wind.doit('test.txt',var='Flow_time')
```

```
Out[88]: 'test_Flow_time.png'
```

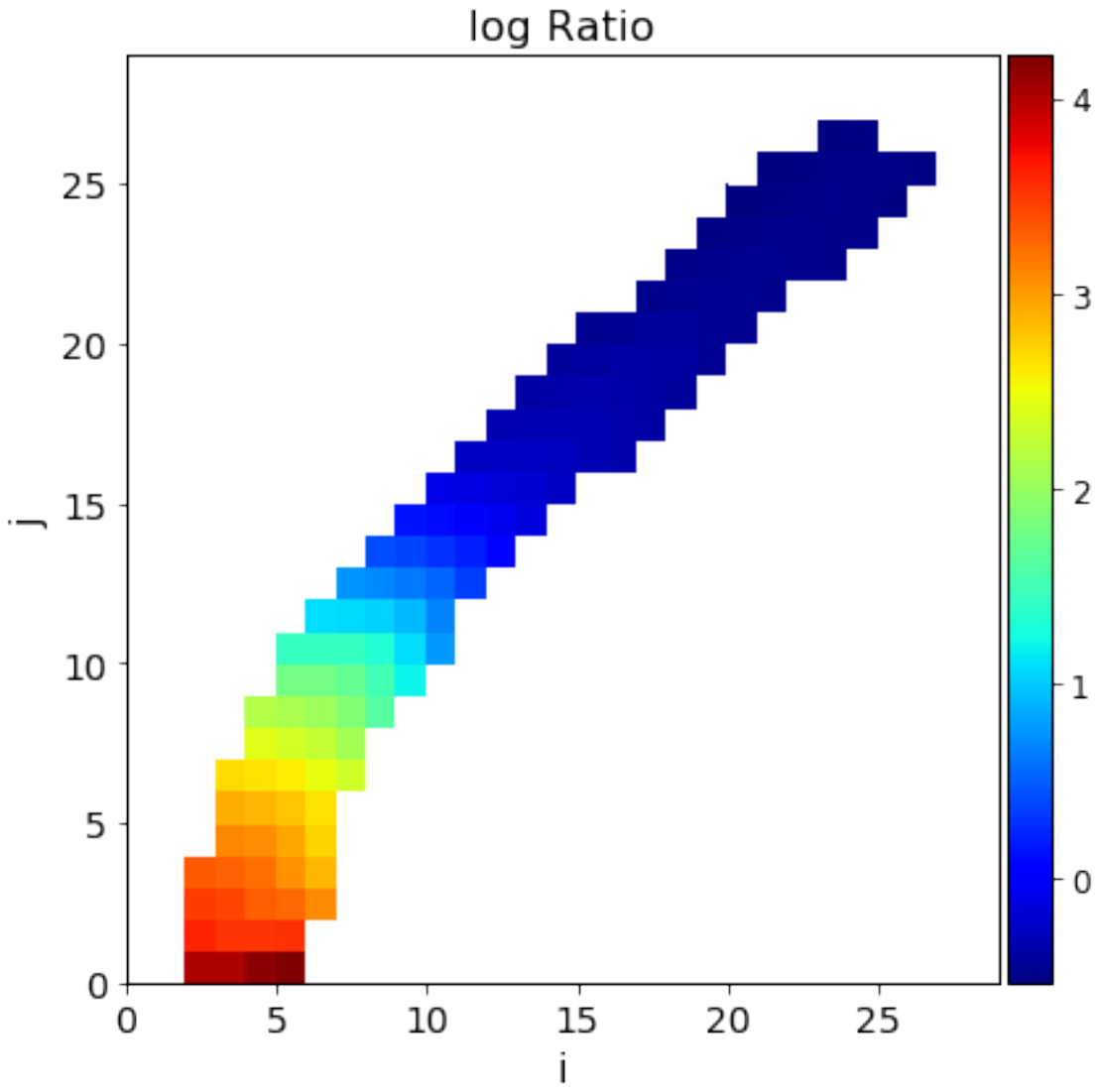


The next plot is the ratio of the cooling time to the flow time (through a cell). A large ratio should imply that the ionization fractions (in the absence of radiation) would change more rapidly in the cell than it would take for the atoms to traverse the cell. Since heating and cooling have to be the same in a converged model, a large value indicates that advected ionization is not a problem.

For the model below, which is a standard CV model, we see that at the base of the wind we are in good shape, but beyond once we get away from this region we have a problem.

Note that the way this criterion is written, the finer the model grid, the larger the region where there would be an issue. One would get a grid-independent answer if one took the distance to the central object.

```
In [89]: plot_wind.doit('test.txt',var='Ratio')

Out[89]: 'test_log_Ratio.png'
```


log Ratio
j
i
0
5
10
15
20
25
0
1
2
3
4