```
''' Read and analyse grafic files that resulted from converting ramses
 output
    $HOME/WIP/generate_IC/amr2grafic -inp output_00001 \
                                        -out grafics \
                                        -lma 8 \
                                        -siz $boxlen
'''

import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

from constants_lineRT import *
import grafic

''' Load simulation data
    PARAMS:
      sim = simulation tag ('box' or 'elena' or 'large_box')
      res = level max
      statistics = flag to plot statistics of data
    RETURNS:
      nx, ny, nz = number of cells in each direction
      dx = cell size (m)
      rho = density (kg/m3) '''
def load_sim_data(sim, res, statistics=False):

    # density
    rho, nx, ny, nz, dx = load_density(sim, res)
    if statistics:
        print 'nx =', nx, ', ny =', ny, ', nz =', nz, 'dx = ', dx
        print 'number of cells:', len(rho.flatten())
        print 'total size: [{}, {}, {}] pc'.format(nx*dx/PC, ny*dx/PC,
         nz*dx/PC)
        print 'total mass = {} Msun'.format(np.sum(rho)*(dx**3)/MSUN)
        plot_density(rho, 'rho_map_'+str(res)+'.png')
        #calc_PDF(rho, 'rho_pdf_'+str(res)+'.png', units='kg/m3')
        calc_PDF(rho, 'rho_pdf_hcc_'+str(res)+'.png', units='H/cc')

    # velocity
    vx = load_velocity(sim, res, 'vx')
    vy = load_velocity(sim, res, 'vy')
    vz = load_velocity(sim, res, 'vz')
    if statistics:
        calc_PDF(vx, 'vx_pdf_'+str(res)+'.png', units='km/s', density=rho)
        calc_PDF(vy, 'vy_pdf_'+str(res)+'.png', units='km/s', density=rho)
        calc_PDF(vz, 'vz_pdf_'+str(res)+'.png', units='km/s', density=rho)
        #calc_PDF(np.sqrt(vx**2+vy**2+vz**2), 'vtot_pdf_'+str(res)+'.png',
         units='km/s', density=rho)

    # velocity gradient
    grad_v = get_grad_vel(vx, vy, vz, nx, ny, nz, dx)
    if statistics:
        calc_PDF(grad_v, 'grad_v_pdf_'+str(res)+'.png', units='1/s',
         density=rho)
```

```
    return nx, ny, nz, dx, rho, vx, vy, vz, grad_v

#-----------------------------------------------------------------------------
 ------------------------
''' Get the right code units for the simulation
    PARAMS:
      sim = simulation tag ('box', 'large_box' or 'elena') '''
def code_units(sim):
    if sim == 'box':
        unit_l = 0.30857e19
        unit_d = 0.150492957435e-19
        unit_t = 0.31556926e14
        boxlen = 0.25 * unit_l * 1.e-2 #m
    elif sim=='large_box':
        unit_l = 0.30857e19
        unit_d = 0.150492957435e-19
        unit_t = 0.31556926e14
        boxlen = 100. * unit_l * 1.e-2 #m
    else: # sim == 'elena':
        unit_l = 0.617200e20
        unit_d = 0.438844e-20
        unit_t = 0.584383e14
        boxlen = 1.0 * unit_l * 1.e-2 #m

    return unit_l, unit_d, unit_t, boxlen

''' Load density grafic file
    PARAMS:
      sim = simulation tag ('box', 'large_box' or 'elena')
      res = level max '''
def load_density(sim, res):
    unit_l, unit_d, unit_t, boxlen = code_units(sim)

    grafics_dir = 'grafics' + str(int(res))
    density = grafic.Grafic()
    density.read(grafics_dir+'/ic_d')
    print density.header

    rho = density.data * unit_d * 1.e3 #kg/m3
    nx = density.header[0]
    ny = density.header[1]
    nz = density.header[2]
    dx =  density.header[3] * unit_l * 1e-2 #m
    if dx == 0.: #amr2cube doesn't output dx
        dx = boxlen/2**res

    return rho, nx, ny, nz, dx

''' Load velocity grafic file
    PARAMS:
      sim = simulation tag ('box', 'large_box' or 'elena')
      res = level max
      comp = component x, y or z'''
def load_velocity(sim, res, comp):
    unit_l, unit_d, unit_t, boxlen = code_units(sim)
```

```
    grafics_dir = 'grafics' + str(int(res))
    vel = grafic.Grafic()
    if comp=='vx':
        vel.read(grafics_dir+'/ic_u')
    elif comp=='vy':
        vel.read(grafics_dir+'/ic_v')
    else:
        vel.read(grafics_dir+'/ic_w')

    return vel.data * unit_l/unit_t * 1e-2 #m/s

''' helper function to get the indices of neighbouring cells (left and right
 from current cell)
    PARAMS:
      i = index of the current cell
      n_max = maximal index, number of cells in current direction
      periodic = flag to indicate if the boundaries are periodic in current
       direction
    RETURNS:
      l = index of the left cell or i if the cell has no left neighbour
      r = index of the right cell or i of the cell has no right neighbour
      neigh = number of neighbours'''
def check_indices(i, n_max, periodic):
    if i==(n_max-1): # cell at the right boundary
        l = i-1
        if periodic:
            r = 0
            neigh = 2
        else:
            r = i # -> this will give a 0 contribution
            neigh = 1
    elif i==0: # cell ath the left boundary
        r = i+1
        if periodic:
            l = n_max-1
            neigh = 2
        else:
            l = i
            neigh = 1
    else: # general case
        r = i+1
        l = i-1
        neigh = 2

    return l, r, neigh

''' Calculate the velocity gradient
    PARAMS:
      vx, vy, vz = datacubes with velocity field (m/s)
      nx, ny, nz = number of cells in each direction
      dx = cell size (m)
      periodic = flag to indicate if there are periodic boundaries or not
    RETURNS:
      grad = velocity gradient (1/s) '''
```

```
def get_grad_vel(vx, vy, vz, nx, ny, nz, dx, periodic=False):

    diff_v = np.zeros((nx,ny,nz))
    grad = np.zeros((nx,ny,nz))
    num_neighbours = np.zeros((nx,ny,nz))

    # vx
    for i in range(nx):
        l, r, neigh = check_indices(i, nx, periodic)
        diff_v[i,:,:] = np.abs(vx[r,:,:] - vx[i,:,:]) + np.abs(vx[l,:,:] -
         vx[i,:,:]) #right+left
        num_neighbours[i,:,:] = num_neighbours[i,:,:] +
         np.full((ny,nz),neigh)
    #print 'diff x', np.amin(diff_v), np.amax(diff_v)
    grad = grad + diff_v

    # vy
    for i in range(ny):
        l, r, neigh = check_indices(i, ny, periodic)
        diff_v[:,i,:] = np.abs(vy[:,r,:] - vy[:,i,:]) + np.abs(vy[:,l,:] -
         vy[:,i,:]) #upper+lower
        num_neighbours[:,i,:] = num_neighbours[:,i,:] +
         np.full((nx,nz),neigh)
    grad = grad + diff_v

    # vz
    for i in range(nz):
        l, r, neigh = check_indices(i, nz, periodic)
        diff_v[:,:,i] = np.abs(vz[:,:,r] - vz[:,:,i]) + np.abs(vz[:,:,l] -
         vz[:,:,i]) #back+front
        num_neighbours[:,:,i] = num_neighbours[:,:,i] +
         np.full((nx,ny),neigh)
    grad = grad + diff_v

    grad = grad/num_neighbours/dx
    #print num_neighbours
    return grad

#----
 diagnostics-------------------------------------------------------------
 --------------------
''' Plot the column density '''
def plot_density(rho, outname):
    rho_col = np.sum(rho, axis=0)
    data = np.log10(rho_col)
    fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(8, 8))
    imag = axes.imshow(data, interpolation='none',
                        #vmin=-18, vmax=-14,
                        #cmap='Greys_r',
                        cmap='inferno',
                        aspect='equal', origin='lower')
    divider = make_axes_locatable(axes)
    cax = divider.append_axes('right', size='5%', pad=0.05)
    cbar = fig.colorbar(imag, cax=cax)
    cbar.solids.set_rasterized(True)
```

```
    cbar.set_label('log(column density)')
    plt.savefig(outname)
    plt.close()

''' plot the PDF of field
    Assumes uni-grid
    If mass-weighted, density field as to be given '''
def calc_PDF(field, outname, units='kg/m3', density=None):

    # presets
    log = False
    if units=='H/cc':
        field = field/MH/1e6
        x_label = 'density [H/cc]'
        log = True
        density = field
        #bins = np.logspace(-5,10,35)
    elif units=='kg/m3':
        x_label = 'density [kg/m3]'
        log = True
        density = field
        #bins = np.logspace(-25,-10,35)
    elif units=='km/s':
        field = field * 1e-3 #km/s
        log = False
        x_label = 'velocity [km/s]'
        #bins = np.linspace(-10,10,35)
    elif units=='1/s':
        x_label = 'velocity gradient [1/s]'
    elif units=='/': #beta
        x_label='beta'
        log = False
    else:
        print 'invalid units option'
        return

    amin = np.amin(field)
    amax = np.amax(field)
    if log:
        bins = np.logspace(np.log10(amin), np.log10(amax), 35)
    else:
        bins = np.linspace(amin, amax, 35)

    field_flat = field.flatten()

    weights = np.full(len(field_flat), 1./len(field_flat))
    plt.hist(field_flat,
             weights=weights,
             bins=bins,
             label='volume weighted',
             histtype='step')

    if density is not None:
        total_density = np.sum(density)
        weights_mass = density.flatten() / total_density
```

```
        plt.hist(field_flat,
                 weights=weights_mass,
                 bins=bins,
                 label='mass weighted',
                 histtype='step')

    plt.xlabel(x_label)
    plt.ylabel('PDF')
    if log:
        plt.xscale('log')
        plt.yscale('log')
    plt.legend(loc='upper left')
    plt.savefig(outname)
    plt.close()

#-------test
 functions------------------------------------------------------------------
 -------------
def example():
    res = 9
    sim = 'large_box'
    load_sim_data(sim, res, statistics=True)

if __name__=='__main__':
    example()
```