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

from constants_lineRT import *

''' Calculate the mock image
    PARAMS:
      emis = emissivity (W/m3/sr/Hz) or (W/m3/sr) when integrated=True
      beta = escape probability (/)
      dx = cell size (m)
      line_freq = the center frequency of the line in the rest frame
      units = desired unit system, choose 'K' or 'W'
      axis = axis of the line of sight (along which emis is determined),
       choose 'vx', 'vy' or 'vz'
      distance (optional) = distance to the source in pc (needed when
       units='W')
      integrated (optional) = flag to indicate the emisivity is integrated
       over frequency W/m3/sr
      thin (optional) = flag to set beta to 1 everywhere, making the medium
       optically thin
    RETURNS:
      image = the intensity map in the desired units
      map_units = a string with the units of the image '''
def calc_image(emis, beta, dx, line_freq, units, axis, distance=None,
              integrated=False, thin=False):

    # emissivty in the direction of the observer
    if thin:
        intensity_cell = emis * dx
    else:
        intensity_cell = emis * beta * dx # W/m2/sr/Hz OR W/m2/sr

    #sum over line of sight
    if axis=='vx':
        intensity_pixel = np.sum(intensity_cell, axis=0)
    elif axis=='vy':
        intensity_pixel = np.sum(intensity_cell, axis=1)
    else: # axis=='vz'
        intensity_pixel = np.sum(intensity_cell, axis=2)

    # remark that the total luminosity of a cell in all directions is
    # lum = 4.*np .pi * emis * beta * dx**3 #W/Hz OR W
    # and then the flux from 1 cell at the observer is
    # F = lum/(4.*np.pi*distance**2) #W/sr/m2/Hz OR W/sr/m2

    if units=='K':
        # temperature units: use B_nu = 2 nu**2 k T / c**2
        image = intensity_pixel * c_si**2 /(2.*kb_si*(line_freq**2)) # K OR
         K * Hz
        if integrated:
            image = image * c_si / line_freq * 1.e-3 # K km/s
            map_units='K km/s'
        else:
            map_units='K'
```

```
    elif units=='W':
        # flux units
        # angular resolution of 1 pixel or cell
        dtheta = dx/(distance*PC) #sr
        image = intensity_pixel * dtheta**2 #W/m2/Hz OR W/m2
        if integrated:
            map_units = 'W/m2'
        else:
            map_units = 'W/m2/Hz'

    return image, map_units

''' Plot the image
    PARAMS:
      image = 2D array with the image
      units = units of the image
      outname = file name to save the image to
      log (optional) = flag to set logaritmic scale '''
def plot_image(image, units, outname, log=False):
    if log:
        final_image = np.log10(image)
    else:
        final_image = image
    fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(8, 8))
    imag = axes.imshow(final_image,
                       interpolation='none',
                       cmap='inferno',
                       #vmin=0, vmax=5.5,
                       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(units)
    plt.savefig(outname)
    plt.close()

''' Calculate the value of the spectrum corresponding to the given
 emissivity
    PARAMS:
      emis = emissivity (W/m3/sr/Hz)
      beta = escape probability (/)
      dx = cell size (m)
      line_freq = the center frequency of the line in the rest frame
      units = desired unit system, choose 'K' or 'W'
      axis = axis of the line of sight (along which emis is determined),
       choose 'vx', 'vy' or 'vz'
      distance (optional) = distance to the source in pc (needed when
       units='W')
      thin (optional) = flag to set beta to 1 everywhere, making the medium
       optically thin
    RETURNS:
```

```
      spectrum_value = the intensity in this wavelength bin in the desired
       units
      map_units = a string with the units of the value '''
def calc_spectrum_value(emis, beta, dx, line_freq, units, axis,
 distance=None, thin=False):

    if thin:
        intensity_cell = emis * dx**3 # W/sr/Hz total luminosity of the cell
         to the observer
    else:
        intensity_cell = emis * beta * dx**3 # W/sr/Hz

    intensity_total = np.sum(intensity_cell) #sum over cube

    if units=='K':
        # temperature units: use B_nu = 2 nu**2 k T / c**2
        spectrum_value = intensity_total * c_si**2 /
         (2.*kb_si*(line_freq**2)) # K * m2
        spectrum_value = spectrum_value / PC**2 # K * pc2
        value_units = 'K pc2'

    elif units=='W':
        # flux units
        # angular resolution of 1 pixel or cell
        dtheta = dx/(distance*PC) #sr
        spectrum_value = intensity_total * dtheta**2 #W/Hz
        value_units = 'W/Hz'

    return spectrum_value, value_units

''' Plot the spectrum
    PARAMS:
      spectrum = array containing the spectrum values
      units_spectrum = units of the spectrum
      bins = bins over which the spectrum is determined
      units_bins = units of the bins
      outname = file name to save the spectrum to
      log (optional) = flag to set logaritmic scale '''
def plot_spectrum(spectrum, units_spectrum, bins, units_bins, outname,
 log=False):

    plt.hist((bins[1:]+bins[:-1])/2., bins=bins, weights=spectrum,
             histtype='step')
    if log:
        plt.xscale('log')
        plt.yscale('log')
    plt.xlabel(units_bins)
    plt.ylabel(units_spectrum)
    plt.savefig(outname)
    plt.close()
```