Galaxy_Overgeneration_(Git)

November 11, 2022

[1]:
```
import skypy
print(skypy.__version__)
```

```
0.6.dev20+g5450d11
```

0.1 Setup:

Import all necessary modules and generate the catalogues

[1]:
```
#Imports
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as np_rand
from skypy.pipeline import Pipeline
from astropy.cosmology import FlatLambdaCDM
from astropy import units as u
from scipy.integrate import trapezoid as trap
```

[3]:
```
#Generate the catalogue

#Galaxy populations (red and blue, using values from Tortorelli et al 2020)
pipe_red_gal = Pipeline.read('red_galaxy.yml')
pipe_red_gal.execute()
pipe_blue_gal = Pipeline.read('blue_galaxy.yml')
pipe_blue_gal.execute()

#Extract the information
red_galaxy = pipe_red_gal['galaxy']
blue_galaxy = pipe_blue_gal['galaxy']

red_mag = red_galaxy['M_B'] #Absolute magnitudes
blue_mag = blue_galaxy['M_B']

print('Catalogues generated')
```

```
Catalogues generated
```

0.2 Analytical:

We generate the analytical integrals for the functions as comparison to the catalogue generated above. Since the catalogues are directly sampled from the analytical functions, they should match well excluding the Poission sampling. The function *mag_lumin* is added as it is cumbersome to use the equation everytime. Cosmology, sky and galaxy parameters are listed for the calculation of the integral and Schechter function.

[4]:
```
#Convert to absolute magnitude to solar luminosity
def mag_lumin(mag_value):
    return 10**(-0.4*(mag_value - 4.85))
```

[5]:
```
# Cosmology and other parameters
cosmology = FlatLambdaCDM(H0=70, Om0=0.3, Ob0=0.045, Tcmb0=2.7,␣
 ↪name='FlatLambdaCDM')
h0 = cosmology.h

n = 100 #Arbitary value
z = np.linspace(0.5,0.6,n)
sky_area = 2*u.deg**2
dVdz = (cosmology.differential_comoving_volume(z)*sky_area).to_value('Mpc3')
```

[6]:
```
# Galaxy parameters
#Compare both magnitude (raw output) and luminosity (used output)
mag_bin = np.linspace(-24, -12, 100)
lumin_bin = mag_lumin(np.flip(mag_bin))
const = 0.4*np.log(10)

#Tortorelli parameters converted
m_int_b = -20.475 + 5*np.log10(h0) # UNITS: Mag
m_int_r = -20.482 + 5*np.log10(h0)
phi_amp_b = np.e**(-5.326) # UNITS: 10**(3) h0**3 MPc**(-3) Mag**(-1)
phi_amp_r = np.e**(-5.683)

#Herbel et al 2017 parametrisation
a_m_b = -0.565 # M_* = a_m*z + b_m
b_m_b = m_int_b
a_phi_b = -0.093 # phi_* = b_phi*e**(a_phi*z)
b_phi_b = phi_amp_b
alpha_b = -1.3

a_m_r = -0.537
b_m_r = m_int_r
a_phi_r = -0.661
b_phi_r = phi_amp_r
alpha_r = -0.5
```

[7]:
```
# Calculate integral per bin

#Arrays
int_mag_b = []
int_mag_r = []
int_lum_b = [] #CONVERT
int_lum_r = []

for jj in range(0,len(mag_bin) - 1):
    mag_range = np.geomspace(mag_bin[jj], mag_bin[jj+1], n) #Create bin for␣
 ↪interval
    lumin_range = np.geomspace(lumin_bin[jj], lumin_bin[jj+1], n)

    dndV_b = [] #Reset for each bin
    dndV_r = []
    L_dndV_b = []
    L_dndV_r = []

    for ii in z:
        #Redshift dependent values
        M_star_b = a_m_b*ii + b_m_b
        M_star_r = a_m_r*ii + b_m_r
        L_star_b = mag_lumin(M_star_b) #Can change M_* to L_* using function
        L_star_r = mag_lumin(M_star_r)
        phi_star_b = b_phi_b*np.e**(a_phi_b*ii) #Assume phi_* is the same for␣
 ↪both L and M
        phi_star_r = b_phi_r*np.e**(a_phi_r*ii)

        #Main section of Schechter function
        m_m_b = 10**(0.4*(M_star_b - mag_range))
        m_m_r = 10**(0.4*(M_star_r - mag_range))
        L_L_b = lumin_range/L_star_b
        L_L_r = lumin_range/L_star_r

        #Generate Schechter for each z
        dndmdV_b = const*phi_star_b*m_m_b**(alpha_b + 1)*np.e**(-m_m_b)
        dndmdV_r = const*phi_star_r*m_m_r**(alpha_r + 1)*np.e**(-m_m_r)

        dndLdV_b = (phi_star_b/L_star_b)*L_L_b**(alpha_b)*np.e**(-L_L_b)
        dndLdV_r = (phi_star_r/L_star_r)*L_L_r**(alpha_r)*np.e**(-L_L_r)

        #Integrate over the magnitude/luminosity range for each z
        dndV_b.append(trap(dndmdV_b, mag_range))
        dndV_r.append(trap(dndmdV_r, mag_range))
        L_dndV_b.append(trap(dndLdV_b, lumin_range))
        L_dndV_r.append(trap(dndLdV_r, lumin_range))
```

```
    #Remove h0 unit
    dndV_b = np.array(dndV_b)*h0**3
    dndV_r = np.array(dndV_r)*h0**3
    L_dndV_b = np.array(L_dndV_b)*h0**3
    L_dndV_r = np.array(L_dndV_r)*h0**3

    #Replace integrand by z (to include the sky area)
    dn_b = dndV_b*dVdz #n(z)*V(z)
    dn_r = dndV_r*dVdz
    L_dn_b = L_dndV_b*dVdz
    L_dn_r = L_dndV_r*dVdz

    #Integral per bin
    int_mag_b.append(trap(dn_b, z))
    int_mag_r.append(trap(dn_r, z))
    int_lum_b.append(trap(L_dn_b, z))
    int_lum_r.append(trap(L_dn_r, z))

#Make into arrays
int_mag_b = np.array(int_mag_b)
int_mag_r = np.array(int_mag_r)
int_lum_b = np.array(int_lum_b)
int_lum_r = np.array(int_lum_r)

print('Integrals complete')
```

```
Integrals complete
```

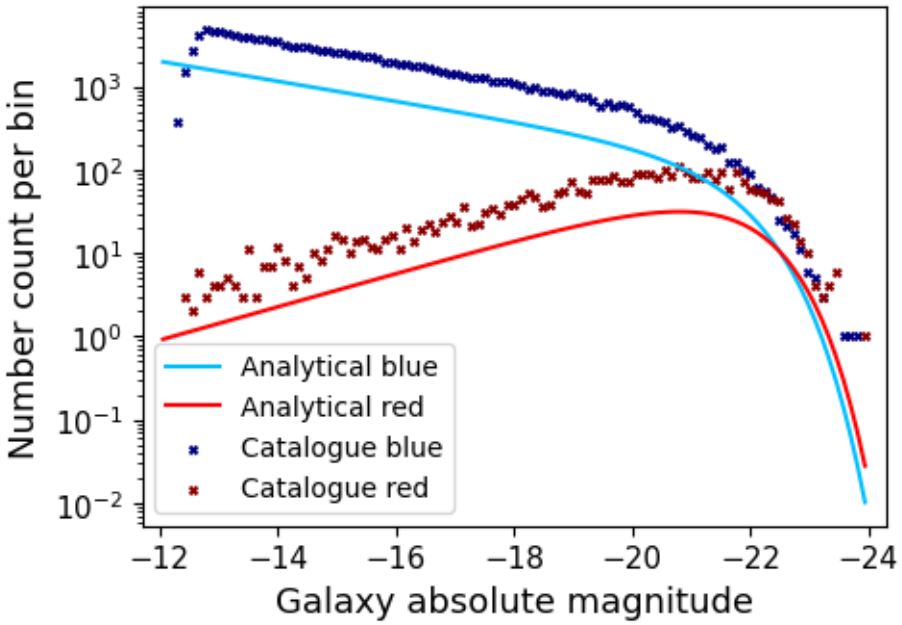
0.3 Problem:

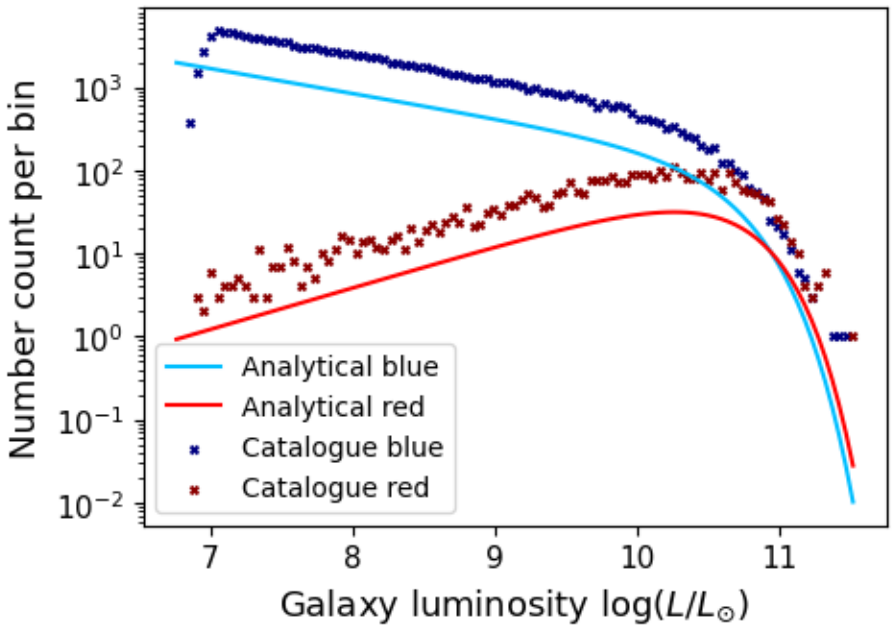
As seen below in the plots of magnitude and luminosity, unmodified SkyPy catalogues significantly over estimate the analytical functions (scatter vs lines) while retaining a similar distribution. This suggests that the issue is in the number calculation not the Schechter function itself or the parameters (these are the same in the YAML file and with the analytical integral).

[8]:
```
# Plots

#Mid points for histograms
mid_mag = (mag_bin[1:] + mag_bin[:-1])/2
mid_lumin = (lumin_bin[1:] + lumin_bin[:-1])/2

#Catalogue values
mag_ngr = np.histogram(red_mag, bins=mag_bin)[0]
mag_ngb = np.histogram(blue_mag, bins=mag_bin)[0]

L_b = mag_lumin(blue_mag)
L_r = mag_lumin(red_mag)
L_r_n = np.histogram(L_r, bins=lumin_bin)[0]
```

```
L_b_n = np.histogram(L_b, bins=lumin_bin)[0]

fig = plt.figure(figsize=(5,3.5))
plt.plot(mid_mag, int_mag_b, label='Analytical blue', c='deepskyblue')
plt.plot(mid_mag, int_mag_r, label='Analytical red', c='red')
plt.scatter(mid_mag, mag_ngb, label='Catalogue blue', c='navy', marker='x', s=8)
plt.scatter(mid_mag, mag_ngr, label='Catalogue red', c='darkred', marker='x',␣
 ↪s=8)
plt.xlabel(r'Galaxy absolute magnitude', fontsize = 13)
plt.ylabel(r'Number count per bin', fontsize = 13)
plt.yscale('log')
plt.xlim([mag_bin[-1]+0.3, mag_bin[0]-0.3]);
plt.xticks(fontsize = 11)
plt.yticks(fontsize = 11)
plt.legend(fontsize = 10)

fig = plt.figure(figsize=(5,3.5))
plt.plot(np.log10(mid_lumin), int_lum_b, label='Analytical blue',␣
 ↪c='deepskyblue')
plt.plot(np.log10(mid_lumin), int_lum_r, label='Analytical red', c='red')
plt.scatter(np.log10(mid_lumin), L_b_n, label='Catalogue blue', c='navy',␣
 ↪marker='x', s=8)
plt.scatter(np.log10(mid_lumin), L_r_n, label='Catalogue red', c='darkred',␣
 ↪marker='x', s=8)
plt.xlabel(r'Galaxy luminosity log($L/L_{\odot}$)', fontsize = 13)
plt.ylabel(r'Number count per bin', fontsize = 13)
plt.yscale('log')
plt.xticks(fontsize = 11)
plt.yticks(fontsize = 11)
plt.legend(fontsize = 10)
```

[8]: <matplotlib.legend.Legend at 0x7f8c92b56860>



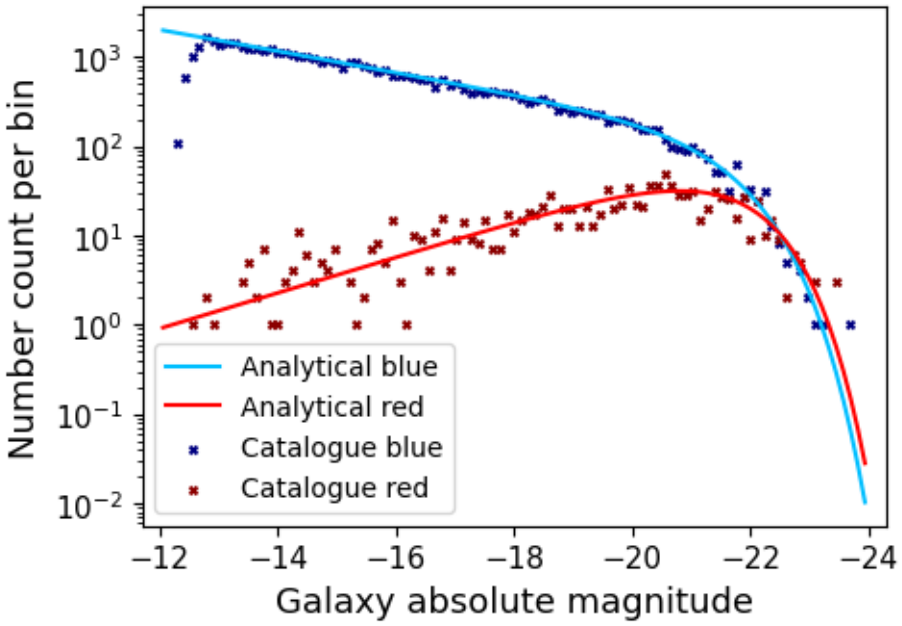


0.4 Fix:

We can fix this by either subsampling around a 1/3 of the generated galaxies...

[8]:
```
new_n_r = int(len(red_mag)/3)
new_n_b = int(len(blue_mag)/3)

rand_list_r = np_rand.randint(0, new_n_r, size=new_n_r) #Random elements to pull
rand_list_b = np_rand.randint(0, new_n_b, size=new_n_b)

new_red = []
new_blue = []

for ii in rand_list_r:
    new_red.append(red_mag[ii])

for jj in rand_list_b:
    new_blue.append(blue_mag[jj])

#Histograms for new catalogues
new_r = np.histogram(new_red, bins=mag_bin)[0]
new_b = np.histogram(new_blue, bins=mag_bin)[0]

#Plot
fig = plt.figure(figsize=(5,3.5))
plt.plot(mid_mag, int_mag_b, label='Analytical blue', c='deepskyblue')
plt.plot(mid_mag, int_mag_r, label='Analytical red', c='red')
plt.scatter(mid_mag, new_b, label='Catalogue blue', c='navy', marker='x', s=8)
plt.scatter(mid_mag, new_r, label='Catalogue red', c='darkred', marker='x', s=8)
plt.xlabel(r'Galaxy absolute magnitude', fontsize = 13)
plt.ylabel(r'Number count per bin', fontsize = 13)
plt.yscale('log')
plt.xlim([mag_bin[-1]+0.3, mag_bin[0]-0.3]);
plt.xticks(fontsize = 11)
plt.yticks(fontsize = 11)
plt.legend(fontsize = 10)
```

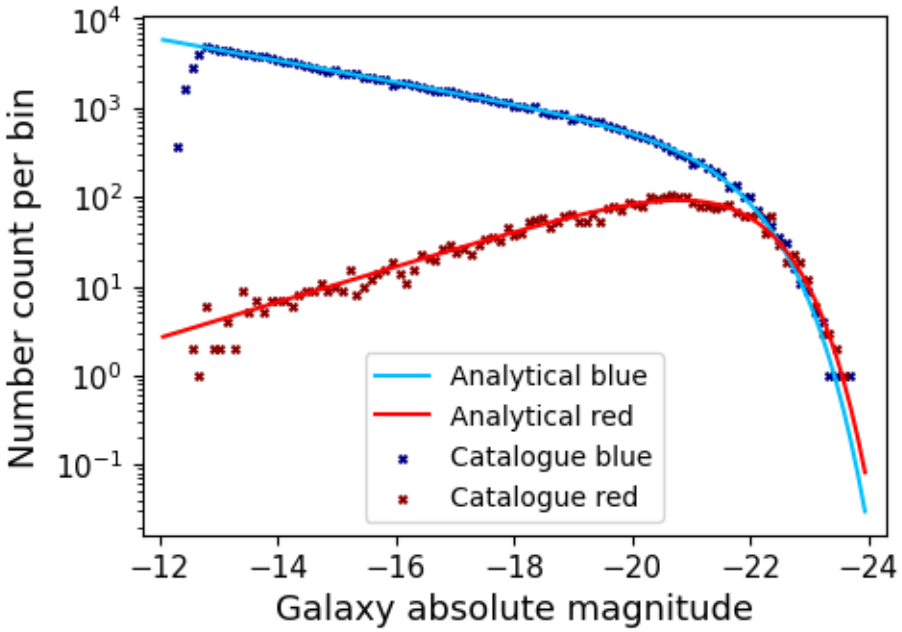
[8]:
```
<matplotlib.legend.Legend at 0x7f415620e320>
```



...or we can remove the multiplication by h_0 in the analytical integral, which lets the analytical values match the catalogue

[9]:
```
#Change analytic values
mag_r = int_mag_r/(h0**3)
mag_b = int_mag_b/(h0**3)

#Plot
fig = plt.figure(figsize=(5,3.5))
plt.plot(mid_mag, mag_b, label='Analytical blue', c='deepskyblue')
plt.plot(mid_mag, mag_r, label='Analytical red', c='red')
plt.scatter(mid_mag, mag_ngb, label='Catalogue blue', c='navy', marker='x', s=8)
plt.scatter(mid_mag, mag_ngr, label='Catalogue red', c='darkred', marker='x',␣
 ↪s=8)
plt.xlabel(r'Galaxy absolute magnitude', fontsize = 13)
plt.ylabel(r'Number count per bin', fontsize = 13)
plt.yscale('log')
plt.xlim([mag_bin[-1]+0.3, mag_bin[0]-0.3]);
plt.xticks(fontsize = 11)
plt.yticks(fontsize = 11)
plt.legend(fontsize = 10)
```

[9]:
```
<matplotlib.legend.Legend at 0x7f41563ae350>
```



0.5 Issue:

Both the galaxies (due to the normalisation constant ϕ_* units) and halos (due to how Colossus returns its value) have the same units of $h^3 Mpc^{-3}$ and therefore must be treated in the same way. The density value for the halos (calculated in *halos/_colossus.py colossus_mf_redshift*, line 135) is multiplied by h_0^3 before being passed onto the general redshift sampling function (*galaxies/redshift.py redshifts_from_comoving_density*). This does *not* happen in the same situation for the galaxies (*galaxies/redshift.py schechter_lf_redshift*, line 129), creating the over abundance.