Handin Assignment 3

Ioannis Koutalios s3365530

January 6, 2025

Abstract

Code and results for Handin assignment 3 for the course Numerical Recipes in Astrophysics.

1 Satellite galaxies around a massive central

In this assignment, we will work once again with the number density satellite profile from the previous assignment. The expression for this profile is given by:

$$n(x) = A\langle N_{sat}\rangle \left(\frac{x}{b}\right)^{a-3} \exp[-\left(\frac{x}{b}\right)^{c}] \tag{1}$$

where x is the radius relative to the virial radius($x = r/r_{vir}$) and a, b, c are free parameters. $\langle N_{sat}\rangle$ is the mean number of satellites in each halo, and A is a normalization factor.

1.1 Finding maximum

For this task, we want to explore the profile we were given and find the value of x ($x \in [0, 5)$) which gives the maximum number of galaxies. To get this value we need a function $N(x)$ which can be written as:

$$N(x) = \int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi} n(x)x^2 \sin(\theta)d\phi d\theta = 4\pi n(x)x^2 \tag{2}$$

For our purposes we assume $a = 2.4$, $b = 0.25$, $c = 1.6$, $\langle N_{sat}\rangle = 100$, and $A = 256/(5\pi^{3/2})$

We define two functions for this purpose. The function "n" and "f_N". The first one takes as input all the different parameters, while the latter only needs an input of x and returns the value $N(x)$ as it was described before.

To find the value of x that maximizes our function we will use two minimization routines that work together. We will minimize the function $-N(x)$ which means that $N(x)$ will be maximized.

The first function that we implemented is the "bracket_minimum" which utilizes the "bracketing a minimum" algorithm. We input two initial points and the algorithm then searches towards the direction of the smallest value of the two to find at least one local minimum. It uses both a golden ratio search and a parabola fitting to find three ordered points x_0, x_1, x_2 for which $f(x_1) < f(x_0)$ and $f(x_1) < f(x_2)$. The middle point has a smaller value than the two points at the edges of the bracket, which means that there is at least one local minimum inside the bracket $[x_0, x_2]$.

The final function that was implemented for this task is the "golden_search", which takes as an input two initial points and finds a local minimum. The two initial points are used by calling the "bracket_minimum" function to obtain a bracket that contains at least one local minimum. We then search inside the bracket to find the exact location, by tightening the bracket using the golden ratio. The code is an adaptation of the pseudocode that is provided in Press et al. (2007).

```
import numpy as np
import math
import matplotlib.pyplot as plt
from open_int import Fx

NORM = 256/(5*np.pi**(3/2))

```

```
def BracketMinimum(func: callable, a: float, b: float, limit: float = 110.0, maxiter: int = 100) -> tuple:
    '''
    function that utilizes the bracketing minimum algorithm
    to return 3 numbers with f2<f1 and f2<f3, which means that
    there is at least on local minimum in [x1,x3]
    input
    func: function to find bracket minimum
    a: lower value of bracket
    b: upper value of bracket
    limit: limit for the max distance of new point
    maxiter: maximum number of iterations
    returns
    3 numbers that bracket at least one minimum
    '''
    w=(1 + math.sqrt(5))/2
    fa, fb = func(a), func(b)

    if fb > fa:
        a, b = b, a
        fa, fb = fb, fa

    c = b + (b - a) * w
    fc = func(c)

    for i in range(maxiter):
        if fc >= fb:
            return a, b, c

        d = b - (1/2)*((b-a)**2*(fb-fc)-(b-c)**2*(fb-fa))/((b-a)*(fb-fc)-(b-c)*(fb-fa))
        fd = func(d)

        if (d - b) * (d - c) < 0: #check if in between
            if fd < fc:
                a, b, c = b, d, c
                fa, fb, fc = fb, fd, fc
                return a, b, c
            elif fd>fb:
                a, b, c = a, b, d
                fa, fb, fc = fa, fb, fd
                return a, b, c
            else:
                d = c + (c-b)*w
                fd = func(d)
        elif (d-c)*(c-b)>0:  #check if it is in the right direction
            if np.abs(d - b) > limit * np.abs(c - b):
                d = c + (c - b) * w
                fd = func(d)
        else: # if not in the right direction calculate new point
            d = c + (c-b)*w
            fd = func(d)

        a, b, c = b, c, d
        fa, fb, fc = fb, fc, fd

    raise ValueError("Maximum number of iterations exceeded.")

def GoldenSearch(func: callable, init1: float, init2: float, tol: float = 1e-5, maxiter: int = 1000) -> float:
    '''
    input
    func: function to find local minimum
    init1, init2: initial points to call bracket_minimum
    tol: relative precision of local minimum
    maxiter: maximum number of iterations
    returns
    x-value of a local minimum
    '''
    phi = (1 + math.sqrt(5))/2
    R = 1/phi
```

```python
    C=1-R
    a,b,d=BracketMinimum(func,init1,init2)

    if abs(d-b) > abs(b-a):
        c=b+C*(d-b)
    else:
        c=b
        b=b+C*(a-b)

    fb = func(b)
    fc= func(c)

    for i in range(maxiter):
        if abs(d-a)<tol*(abs(b)+abs(c)):
            if fb<fc:
                return b
            else:
                return c
        if fc<fb:
            a = b
            b = c
            c = R*c+C*d
            fb = fc
            fc = func(c)
        else:
            d = c
            c = b
            b = R*b+C*a
            fc = fb
            fb = func(b)
    raise ValueError("Maximum number of iterations exceeded.")

if __name__ == '__main__':
    # Wrapper function to include the normalization factor
    f_N = lambda x: Fx(x, NORM)

    # Find the maximum of N(x)
    maximum = GoldenSearch(lambda x: -f_N(x), 1, 2) # Minimize -f_N(x)
    N_max = f_N(maximum)                            # Evaluate the maximum value

    # Output the results
    print(f'Local maximum at: x = {maximum:.10}')
    print(f'Maximum value of: N(x) = {N_max:.10}')

    # Plotting the results
    x = np.linspace(0.01, 5, 10000)
    y = f_N(x)

    fig, ax = plt.subplots()
    ax.plot(x, y, color='green', label='N(x)')
    ax.scatter(maximum, N_max, color='r', marker='+', label='Maximum')
    ax.legend(loc='upper right')
    axins = ax.inset_axes([0.30, 0.30, 0.5, 0.5])
    axins.plot(x, y, color='green')
    axins.scatter(maximum, N_max, color='r', marker='+')
    axins.set_xlim(0.15, 0.30)
    axins.set_ylim(260, 270)
    ax.indicate_inset_zoom(axins, edgecolor="black")
    plt.savefig('plots/maximization.png', dpi=300)
    plt.close()
```

minimize.py

In the main part of our script, we call our minimization function for $-N(x)$ which as we previously discussed will give us the location of the maximum value of $N(x)$. We then calculate the actual value of the function at this position and output the results. We also create a plot to visualize the results better.

```
Local maximum at: x = 0.2299829905
Maximum value of: N(x) = 267.8455331
```

output/minimize.txt

We display the results of our script which include the position of the maximum number of galaxies and the value of $N(x)$ at this position. Both are printed with a 10 significant-digit precision. In Figure 1 we can visualize this result even better. As we can see the maximum that we calculate using our script is correctly placed with great precision.

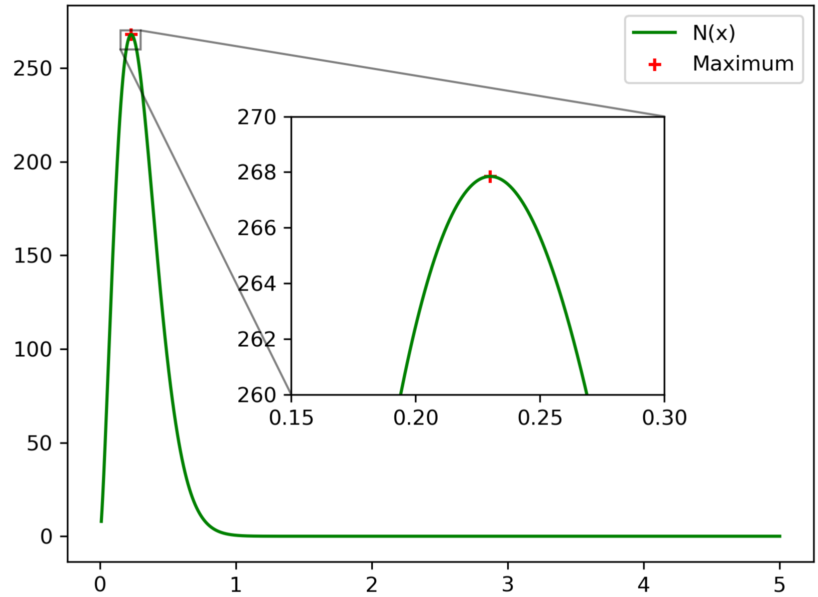

Figure 1: The number of galaxies as a function of x (radius relative to the virial radius). We also plot the position of the maximum value as it was calculated using the "golden section search" algorithm with the help of the "bracketing a minimum" algorithm.

1.2 χ^2 minimization

For our next task, we want to use this model to fit some data. We have 5 different files containing haloes (in a certain halo mass bin) with variable numbers of satellites. The satellite galaxies are discrete objects and we, therefore, have a Poisson distribution. We can however fit the data using a χ^2 minimization fit to find the values of the free parameters a, b, c.

We will perform a χ^2 fit with the correct Poisson variance $\sigma^2 = \mu$. To do that we need to first bin our data. We chose 20 bins evenly distributed in real space. The reason we chose to work in real space and not in a logarithmic one was the fact that while we attempted to use both, our implementation worked much better in real space. This certainly has to do with our functions and the minimization algorithm that we used.

To fit a model in a set of binned data using a χ^2 minimization we use the following equation

$$\chi^2 = \sum_{i=0}^{N-1} \frac{[y_i - \mu(x_i|p)]^2}{\sigma^2} = \sum_{i=0}^{N-1} \frac{[y_i - \mu(x|p)]^2}{\mu(x_i|p)^2} \tag{3}$$

where y_i is the observed binned data, $\mu(x|p)$ is the model mean at the middle of each bin, and σ^2 is the variance of the model which as we already discussed is equal to the model mean.

In our problem we evaluate the model mean as follows:

$$N_i = 4\pi \int_{x_i}^{x_{i+1}} n(x) x^2 dx \tag{4}$$

which is essentially the integral over the length of each bin for our equation $N(x)$ which we previously discussed. It is important to note that as the values of a, b, c change during minimization we need to calibrate A so that the following equation is always true:

$$4\pi \int_{x=0}^{5} n(x)x^2 dx = \langle N_{sat} \rangle \tag{5}$$

To accomplish all that we implement many different functions. First of all, we have a "readfile" function that can give us a list of all the satellite galaxy radii and also the number of haloes in each file. We then define the "f_n" function which is the $N(x)$ function defined in a way that doesn't require an open formula for it to be integrated. We also use a "trapezoid" function to enable us to calculate the integral. We also experimented with using a function that utilizes the Romberg integration algorithm, with no improvement on the final performance, so we decided to use the more simple trapezoid algorithm. All these integration routines were implemented in a previous assignment.

To perform the minimization we implemented the downhill simplex algorithm in our function "simplex_3d". This implementation is mainly for 3-dimensional spaces as it requires four initial points to work with (N+1). These four points can be visualized as a triangular prism in the 3d plane of our parameters. We propose a new point by reflecting the worst point on the surface that is defined by the other 3 points. If our new point has the best estimate overall we expand once more and check if there is still an improvement. Otherwise, we check if our new point is an improvement over our worst point and we then accept our new point. If neither of these two cases is met, we propose a new point by contracting instead of reflecting, which means that we search inwards instead of outwards. If this didn't also give us a better result we will shrink our initial prism. We iterate for a number of max iterations and return our best guess or we can stop early if the fractional range in our function evaluation gets sufficiently small.

We then define the "chisquare" function which is the one we want to minimize. The function takes a list of the three parameters we try to evaluate, the counts of the binned data and the edges of each bin. We also input $\langle N_{sat} \rangle$. We then calculate A for the specific values of a, b, c and loop over all bins to calculate the χ^2 sum as we defined in Equation (3).

We also define one final function to handle the plotting of the binned data with the best-fitted model.

```python
import numpy as np
import matplotlib.pyplot as plt
from open_int import Trapezoid

def ReadFile(filename: str) -> tuple:
    '''
    function to read the data from the file
    '''
    f = open(filename, 'r')
    data = f.readlines()[3:]     # Skip first 3 lines
    nhalo = int(data[0])         # number of halos
    radius = []

    for line in data[1:]:
        if line[:-1]!='#':
            radius.append(float(line.split()[0]))

    radius = np.array(radius, dtype=float)
    f.close()
    return radius, nhalo         # Return the virial radius for all the satellites in the
     file, and the number of halos

def F_n(x: float, norm: float, sat: float, a: float, b: float, c: float) -> float:
    '''
    function to be integrated
    we define it differently so we don't need to use an open formula
    we also keep everything as arguments so we can use it in the chi^2 function
    '''
    return 4*np.pi*norm*sat*((1/b)**(a-3))*np.exp(-(x/b)**c)*x**(a-3+2)

def Simplex3D(func: callable, n1: list = [0., 0., 0.], n2: list = [1., 0., 0.], n3: list
     = [0., 1., 0.],
        n4: list = [0., 0., 1.], maxiter: int = 1000, tol: float = 1e-12) -> list:
    '''
```

```
    utilizes the downhill simplex algorithm
    for function minimization in 3-dimensions
    input:
    func: function to be minized
    n1,n2,n3,n4: 4 initial points (N+1)
    maxiter: maximum number of iterations
    tol: sufficiently small fractional range
    function evaluation for early termination
    output:
    single point that minimizes the function
    '''
    N = 3
    f_vals = np.array([func(n1),func(n2),func(n3),func(n4)],dtype=float)
    simplex = np.array([n1,n2,n3,n4],dtype=float)

    for iterat in range(maxiter):
        for i in range(N):
            i_min = i
            for j in range(i+1,N+1):
                if f_vals[j]<f_vals[i_min]:
                    i_min = j
            if i_min != i:
                f_vals[i] , f_vals[i_min] = np.copy(f_vals[i_min]) , np.copy(f_vals[i])
                simplex[i], simplex[i_min]  = np.copy(simplex[i_min]), np.copy(simplex[i])
        if 2*abs(f_vals[-1]-f_vals[0])/abs(f_vals[-1]+f_vals[0])<tol:
            return simplex[0]
        x = np.zeros(N)
        for i in range(N):
            x[i] = np.sum(simplex[:-1,i])/N
        x_try = 2*x - simplex[-1]
        f_try = func(x_try)
        if f_try<f_vals[0]:
            x_exp = 2*x_try-x
            if func(x_exp)<f_try:
                simplex[-1] = x_exp
                f_vals[-1] = func(x_exp)
            else:
                simplex[-1] = x_try
                f_vals[-1] = f_try
        elif f_try<f_vals[-1]:
            simplex[-1] = x_try
            f_vals[-1] = f_try
        else:
            x_try = (x+simplex[-1])/2
            f_try = func(x_try)
            if f_try<f_vals[-1]:
                simplex[-1] = x_try
                f_vals[-1] = f_try
            else:
                for i in range(1,N):
                    simplex[i] = (simplex[0]+simplex[i])/2

    return simplex[0]

def ChiSquare(values: list, counts: list, bins: list, sat: float) -> float:
    '''
    chi^2 function
    input:
    values: list of free parameters
    counts: number of counts in each bin
    bins: the edges of the each bin
    sat: average number of galaxies per halo
    output:
    the chi^2 evaluation of our data with the model
    '''
    a = values[0]
    b = values[1]
    c = values[2]

```

```
    norm = sat/Trapezoid(lambda x: F_n(x,norm=1,sat=sat,a=a,b=b,c=c),0,5,100)

    sums = 0
    for i in range(len(bins)-1):
        mean = Trapezoid(lambda x: F_n(x,norm=norm,sat=sat,a=a,b=b,c=c),bins[i],bins[i+1],100)/sat
        sums += (counts[i]-mean)**2/(mean)
    return sums

def Plot(counts: list, bins: list, x: np.array, y: np.array, name: str = 'hist', types: str = 'png', dpi: int = 300):
    '''
    plotting function
    '''
    plt.bar(bins[:-1] + np.diff(bins) / 2, counts, np.diff(bins),color='r',label='binned data')
    plt.plot(x,y,color='black',label='fitted model')
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('x')
    plt.ylabel('N')
    plt.legend()
    plt.ylim(10**-5,10**1)
    plt.savefig(f'plots/{name}.{types}',dpi=dpi)
    plt.close()

def PlotAppendix(counts: list, bins: list, x: np.array, y: np.array, name: str = 'hist-app', types: str = 'png', dpi: int = 300):
    '''
    plotting function for appendix
    '''
    plt.bar(bins[:-1] + np.diff(bins) / 2, counts, np.diff(bins),color='r',label='binned data')
    plt.plot(x,y,color='black',label='fitted model')
    plt.xlabel('x')
    plt.ylabel('N')
    plt.legend()
    plt.ylim(0,np.amax(y)+.2)
    plt.savefig(f'plots/{name}.{types}',dpi=dpi)
    plt.close()

if __name__ == '__main__':

    #reading the data
    radius1, nhalo1 = ReadFile('data/satgals_m11.txt')
    radius2, nhalo2 = ReadFile('data/satgals_m12.txt')
    radius3, nhalo3 = ReadFile('data/satgals_m13.txt')
    radius4, nhalo4 = ReadFile('data/satgals_m14.txt')
    radius5, nhalo5 = ReadFile('data/satgals_m15.txt')

    # binning the data
    count1, bins1 = np.histogram(radius1, bins=20, density=True)
    count2, bins2 = np.histogram(radius2, bins=20, density=True)
    count3, bins3 = np.histogram(radius3, bins=20, density=True)
    count4, bins4 = np.histogram(radius4, bins=20, density=True)
    count5, bins5 = np.histogram(radius5, bins=20, density=True)

    # calculating sat for each file
    sat1 = len(radius1)/nhalo1
    sat2 = len(radius2)/nhalo2
    sat3 = len(radius3)/nhalo3
    sat4 = len(radius4)/nhalo4
    sat5 = len(radius5)/nhalo5

    # minimizing chisquare using simplex
    res1=Simplex3D(lambda values: ChiSquare(values,counts=count1,bins=bins1,sat=sat1),n1=(2.4,.25,1.6)
        ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

    res2=Simplex3D(lambda values: ChiSquare(values,counts=count2,bins=bins2,sat=sat2),n1
```

```
=(2.4,.25,1.6)
    ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

res3=Simplex3D(lambda values: ChiSquare(values,counts=count3,bins=bins3,sat=sat3),n1
=(2.4,.25,1.6)
    ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

res4=Simplex3D(lambda values: ChiSquare(values,counts=count4,bins=bins4,sat=sat4),n1
=(2.4,.25,1.6)
    ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

res5=Simplex3D(lambda values: ChiSquare(values,counts=count5,bins=bins5,sat=sat5),n1
=(2.4,.25,1.6)
    ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

np.save('output/chi_results.npy',np.array([res1,res2,res3,res4,res5]))

# getting the model for plotting
x = np.linspace(.0001,5,1000)

norm1 = sat1/Trapezoid(lambda x: F_n(x,sat=sat1,norm=1,a=res1[0],b=res1[1],c=res1
[2]),0,5,1000)
y1 = F_n(x,sat=sat1,norm=norm1,a=res1[0],b=res1[1],c=res1[2])/sat1

norm2 = sat2/Trapezoid(lambda x: F_n(x,sat=sat2,norm=1,a=res2[0],b=res2[1],c=res2
[2]),0,5,1000)
y2 = F_n(x,sat=sat2,norm=norm2,a=res2[0],b=res2[1],c=res2[2])/sat2

norm3 = sat3/Trapezoid(lambda x: F_n(x,sat=sat3,norm=1,a=res3[0],b=res3[1],c=res3
[2]),0,5,1000)
y3 = F_n(x,sat=sat3,norm=norm3,a=res3[0],b=res3[1],c=res3[2])/sat3

norm4 = sat4/Trapezoid(lambda x: F_n(x,sat=sat4,norm=1,a=res4[0],b=res4[1],c=res4
[2]),0,5,1000)
y4 = F_n(x,sat=sat4,norm=norm4,a=res4[0],b=res4[1],c=res4[2])/sat4

norm5 = sat5/Trapezoid(lambda x: F_n(x,sat=sat5,norm=1,a=res5[0],b=res5[1],c=res5
[2]),0,5,1000)
y5 = F_n(x,sat=sat5,norm=norm5,a=res5[0],b=res5[1],c=res5[2])/sat5

# plotting
Plot(count1,bins1,x,y1,'chi-fit1')
Plot(count2,bins2,x,y2,'chi-fit2')
Plot(count3,bins3,x,y3,'chi-fit3')
Plot(count4,bins4,x,y4,'chi-fit4')
Plot(count5,bins5,x,y5,'chi-fit5')

PlotAppendix(count1,bins1,x,y1,'chi-fit-app1')
PlotAppendix(count2,bins2,x,y2,'chi-fit-app2')
PlotAppendix(count3,bins3,x,y3,'chi-fit-app3')
PlotAppendix(count4,bins4,x,y4,'chi-fit-app4')
PlotAppendix(count5,bins5,x,y5,'chi-fit-app5')

# results
print(f'dataset m11')
print(f'N of sat = {sat1}')
print(f'[a,b,c] = {res1}')
print(f'chi^2 = {ChiSquare(values=res1,counts=count1,bins=bins1,sat=sat1)}')
print(f'\ndataset m12')
print(f'N of sat = {sat2}')
print(f'[a,b,c] = {res2}')
print(f'chi^2 = {ChiSquare(values=res2,counts=count2,bins=bins2,sat=sat2)}')
print(f'\ndataset m13')
print(f'N of sat = {sat3}')
print(f'[a,b,c] = {res3}')
print(f'chi^2 = {ChiSquare(values=res3,counts=count3,bins=bins3,sat=sat3)}')
print(f'\ndataset m14')
print(f'N of sat = {sat4}')
print(f'[a,b,c] = {res4}')
print(f'chi^2 = {ChiSquare(values=res4,counts=count4,bins=bins4,sat=sat4)}')
```

```
    print(f'\ndataset m15')
    print(f'N of sat = {sat5}')
    print(f'[a,b,c] = {res5}')
    print(f'chi^2 = {ChiSquare(values=res5, counts=count5, bins=bins5, sat=sat5)}')
```

chisquare.py

In the main part of our code, we first want to read the data from the text files. We then bin the data using the numpy routine with "density=True" to normalize it. We also calculate $\langle N_{sat} \rangle$ for each of our data files. Because we are working with normalized distributions our $\langle N_{sat} \rangle$ is the number of galaxies in each file divided by the number of haloes. After that, we want to minimize our χ^2 function for each data file and get the best estimates of a, b, c. The next part of the script is just for plotting and printing the values we want. Before plotting we need to calculate the correct value of A for the specific data file and the values of the parameters.

```
dataset m11
N of sat = 0.013680508914912019
[a,b,c] = [1.25189591 1.14825645 3.30446672]
chi^2 = 51.14466772367277

dataset m12
N of sat = 0.25088679479075005
[a,b,c] = [1.552253   0.93283261 3.57893627]
chi^2 = 90.39842492272949

dataset m13
N of sat = 4.373510796723753
[a,b,c] = [1.42181743 0.82740094 3.01461248]
chi^2 = 97.88014132647872

dataset m14
N of sat = 29.134529147982065
[a,b,c] = [1.88923573 0.62441869 2.58991663]
chi^2 = 190.61651011411195

dataset m15
N of sat = 329.5
[a,b,c] = [1.91675076 0.73196938 2.0553574 ]
chi^2 = 110.4576038502151
```

output/chisquare.txt

In the output of our script, we can see the values of $\langle N_{sat} \rangle$, a, b, c, χ^2 for each data file. From Figure 2 we can see that our fitting was successful in all five datasets. The fitted model is falling on top of the bins as we would expect. This becomes even more clear when we plot in real space instead of log space. These plots can be found in Figure 4.

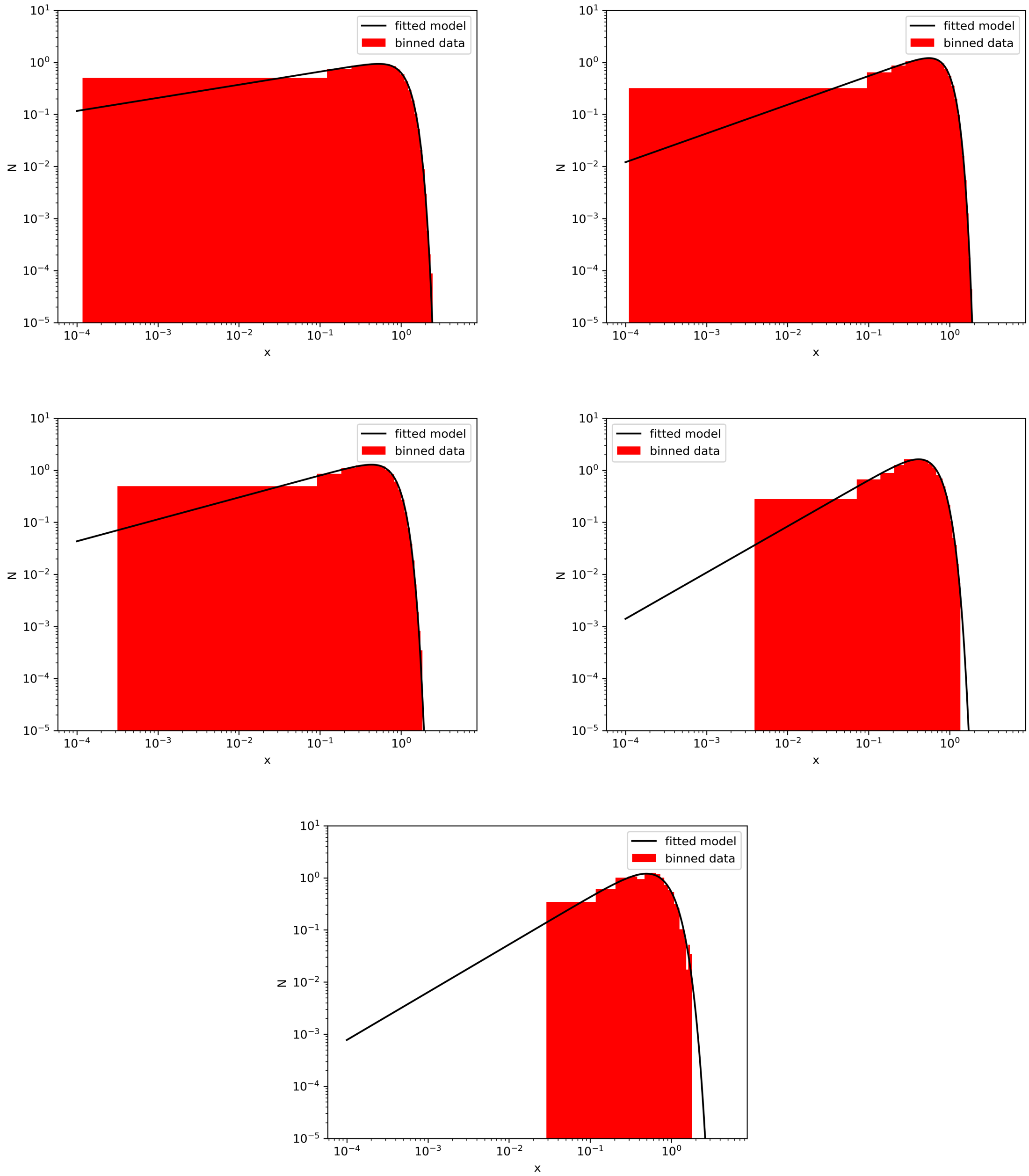

Figure 2: The binned data with the best-fit model for each dataset. The plot is in logarithmic scale for both the x and the y axis. To fit the model we minimized χ^2 function. We can see that the model greatly fits our data in all five cases.

1.3 Poisson log-likelihood

In this task, we want to switch from minimizing the χ^2 function to the more appropriate one for our problem Poisson log-likelihood. The negative log-likelihood that we want to minimize can be expressed by:

$$-\ln(L(p)) = -\sum_{i=0}^{N-1} [y_i \ln(\mu(x_i|p)) - \mu(x_i|p) - \ln(y_i!)] \tag{6}$$

where y_i is the observed binned data, $\mu(x|p)$ is the model mean at the middle of each bin. We notice that the last term is not dependent on our parameters p and can therefore be disregarded during the minimization process.

We implement our function "loglikelihood" which calculates the negative log-likelihood as it was described by inputting the observed binned data and the model. We then use the same minimization process that was described and implemented in Section 1.2 by calling the "simplex_3d" function to minimize the log-likelihood.

```python
import numpy as np
from chisquare import ReadFile, F_n, Trapezoid, Simplex3D, Plot, PlotAppendix

def LogLikelihood(values: list, counts: list, bins: list, sat: float) -> float:
    '''
    log-likelihood function
    input:
    values: list of free parameters
    counts: number of counts in each bin
    bins: the edges of the each bin
    sat: average number of galaxies per halo
    output:
    the log-likelihood evaluation of our data
    with the model
    '''
    a = values[0]
    b = values[1]
    c = values[2]

    norm = sat/Trapezoid(lambda x: F_n(x,norm=1,sat=sat,a=a,b=b,c=c),0,5,1000)

    sums = 0
    for i in range(len(bins)-1):
        mean = Trapezoid(lambda x: F_n(x,norm=norm,sat=sat,a=a,b=b,c=c),bins[i],bins[i+1],100)/sat
        sums -= (counts[i]*np.log(mean)-mean)
    return sums


if __name__ == '__main__':

    #reading the data
    radius1, nhalo1 = ReadFile('data/satgals_m11.txt')
    radius2, nhalo2 = ReadFile('data/satgals_m12.txt')
    radius3, nhalo3 = ReadFile('data/satgals_m13.txt')
    radius4, nhalo4 = ReadFile('data/satgals_m14.txt')
    radius5, nhalo5 = ReadFile('data/satgals_m15.txt')

    # binning the data
    count1, bins1 = np.histogram(radius1, bins=20, density=True)
    count2, bins2 = np.histogram(radius2, bins=20, density=True)
    count3, bins3 = np.histogram(radius3, bins=20, density=True)
    count4, bins4 = np.histogram(radius4, bins=20, density=True)
    count5, bins5 = np.histogram(radius5, bins=20, density=True)

    # calculating sat for each file
    sat1 = len(radius1)/nhalo1
    sat2 = len(radius2)/nhalo2
    sat3 = len(radius3)/nhalo3
    sat4 = len(radius4)/nhalo4
    sat5 = len(radius5)/nhalo5

    res1 = Simplex3D(lambda values: LogLikelihood(values,counts=count1,bins=bins1,sat=sat1),n1=(2.4,.25,1.6)
        ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

    res2 = Simplex3D(lambda values: LogLikelihood(values,counts=count2,bins=bins2,sat=sat2),n1=(2.4,.25,1.6)
        ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

    res3 = Simplex3D(lambda values: LogLikelihood(values,counts=count3,bins=bins3,sat=sat3),n1=(2.4,.25,1.6)
        ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)
```

```
    res4 = Simplex3D(lambda values: LogLikelihood(values,counts=count4,bins=bins4,sat=sat4),n1=(2.4,.25,1.6)
         ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

    res5 = Simplex3D(lambda values: LogLikelihood(values,counts=count5,bins=bins5,sat=sat5),n1=(2.4,.25,1.6)
         ,n2=(2.3,.25,1.6),n3=(2.4,.35,1.6),n4=(2.4,.25,1.5),maxiter=100,tol=1e-6)

    np.save('output/pois_results.npy',np.array([res1,res2,res3,res4,res5]))


    # getting the model for plotting
    x = np.linspace(.0001,5,1000)

    norm1 = sat1/Trapezoid(lambda x: F_n(x,norm=1,sat=sat1,a=res1[0],b=res1[1],c=res1[2]),0,5,1000)
    y1 = F_n(x,norm=norm1,sat=sat1,a=res1[0],b=res1[1],c=res1[2])/sat1

    norm2 = sat2/Trapezoid(lambda x: F_n(x,norm=1,sat=sat2,a=res2[0],b=res2[1],c=res2[2]),0,5,1000)
    y2 = F_n(x,norm=norm2,sat=sat2,a=res2[0],b=res2[1],c=res2[2])/sat2

    norm3 = sat3/Trapezoid(lambda x: F_n(x,norm=1,sat=sat3,a=res3[0],b=res3[1],c=res3[2]),0,5,1000)
    y3 = F_n(x,norm=norm3,sat=sat3,a=res3[0],b=res3[1],c=res3[2])/sat3

    norm4 = sat4/Trapezoid(lambda x: F_n(x,norm=1,sat=sat4,a=res4[0],b=res4[1],c=res4[2]),0,5,1000)
    y4 = F_n(x,norm=norm4,sat=sat4,a=res4[0],b=res4[1],c=res4[2])/sat4

    norm5 = sat5/Trapezoid(lambda x: F_n(x,norm=1,sat=sat5,a=res5[0],b=res5[1],c=res5[2]),0,5,1000)
    y5 = F_n(x,norm=norm5,sat=sat5,a=res5[0],b=res5[1],c=res5[2])/sat5

    # plotting
    Plot(count1,bins1,x,y1,'pois-fit1')
    Plot(count2,bins2,x,y2,'pois-fit2')
    Plot(count3,bins3,x,y3,'pois-fit3')
    Plot(count4,bins4,x,y4,'pois-fit4')
    Plot(count5,bins5,x,y5,'pois-fit5')

    PlotAppendix(count1,bins1,x,y1,'pois-fit-app1')
    PlotAppendix(count2,bins2,x,y2,'pois-fit-app2')
    PlotAppendix(count3,bins3,x,y3,'pois-fit-app3')
    PlotAppendix(count4,bins4,x,y4,'pois-fit-app4')
    PlotAppendix(count5,bins5,x,y5,'pois-fit-app5')

    # results
    print(f'dataset m11')
    print(f'[a,b,c] = {res1}')
    print(f'-ln(L) = {LogLikelihood(values=res1,counts=count1,bins=bins1,sat=sat1)}')
    print(f'\ndataset m12')
    print(f'[a,b,c] = {res2}')
    print(f'-ln(L) = {LogLikelihood(values=res2,counts=count2,bins=bins2,sat=sat2)}')
    print(f'\ndataset m13')
    print(f'[a,b,c] = {res3}')
    print(f'-ln(L) = {LogLikelihood(values=res3,counts=count3,bins=bins3,sat=sat3)}')
    print(f'\ndataset m14')
    print(f'[a,b,c] = {res4}')
    print(f'-ln(L) = {LogLikelihood(values=res4,counts=count4,bins=bins4,sat=sat4)}')
    print(f'\ndataset m15')
    print(f'[a,b,c] = {res5}')
    print(f'-ln(L) = {LogLikelihood(values=res5,counts=count5,bins=bins5,sat=sat5)}')
```

likelihood.py

The code outputs the values of a, b, c for each dataset and also the negative log-likelihood that was minimized. We also create plots to visualize the performance of our fitting. As we can see in Figure 3 the fitting is good as our model greatly matches the binned data. This becomes even more clear when we plot in real space as is shown in Figure 5

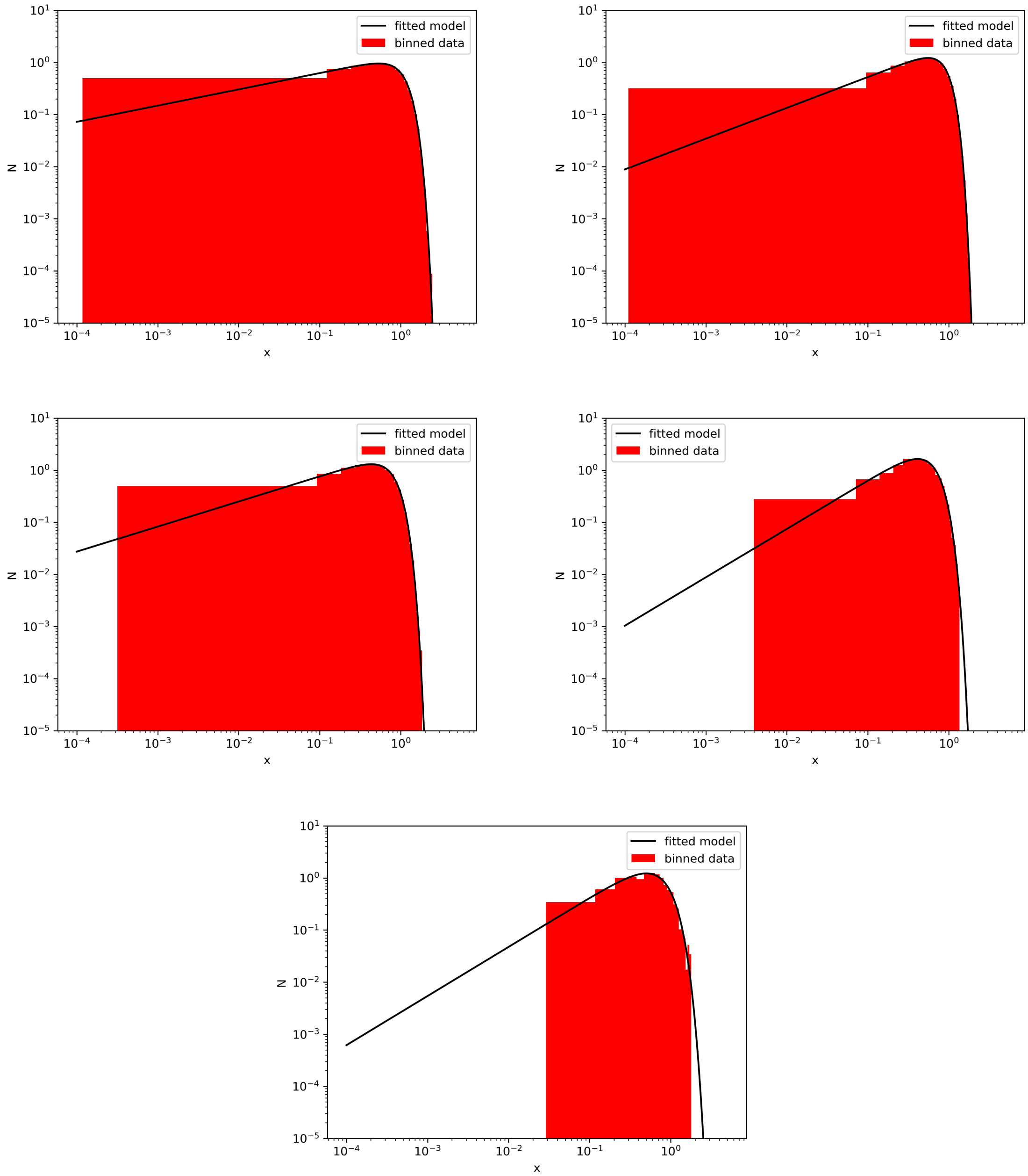

Figure 3: The binned data with the best-fit model for each dataset. The plot is in logarithmic scale for both the x and the y axis. To fit the model we minimized Poisson log-likelihood. We can see that the model greatly fits our data in all five cases.

```
dataset m11
[a,b,c] = [1.31213871 1.11505364 3.13817147]
−ln(L) = 21.33742003102692

dataset m12
[a,b,c] = [1.59115263 0.92056622 3.49200163]
−ln(L) = 27.514618311032272

dataset m13
```

```
[a,b,c] = [1.48177745 0.80386392 2.87972997]
−ln(L) = 28.274319516330866

dataset m14
[a,b,c] = [1.92976627 0.61176745 2.53381833]
−ln(L) = 39.73620879224263

dataset m15
[a,b,c] = [1.94332333 0.73509004 2.11947913]
−ln(L) = 31.385736555336113
```

output/likelihood.txt

1.4 Statistical tests

In this section, we want to test our fitting by using two different statistical tests. The first one is the G-test which computes the G-value using the formula:

$$G = 2\sum_i O_i \ln(\frac{O_i}{E_i}) \tag{7}$$

where O_i, E_i are the observed and expected number of counts in each bin. After calculating this value we compute the probability by calculating:

$$P(G,k) = \frac{\gamma(\frac{k}{2},\frac{G}{2})}{\Gamma(\frac{k}{2})} \tag{8}$$

where γ is the incomplete Γ function and k are the degrees of freedom. For our problem, the degrees of freedom can be calculated by subtracting the number of parameters (3) from the number of bins that were used for each dataset. Although during our implementation we calculate this value for each dataset to make the code more generalized, the value of k for all cases is 17 as we have 20 bins for all the datasets. When we have the probability we calculate the Q-value which is equivalent to the more known p-value by using the equation:

$$Q = 1 - P(G,k) \tag{9}$$

The second test we want to implement is the Kolmogorov-Smirnov test more known as the K-S test. We now want to calculate the cdf of both our observations and our fitted model. We then calculate the maximum difference between the two of them which is denoted as D and used to calculate z using the formula:

$$z = (\sqrt{N} + 0.12 + \frac{0.11}{\sqrt{N}})D \tag{10}$$

where N is the number of bins that were used to bin our observations. We then use z to calculate the K-S probability. To do this in the numerically optimal way we use one of the two formulas depending on the value of z

$$P_{KS}(z) = \frac{\sqrt{2\pi}}{z}[(e^{-\pi^2/(8z^2)}) + (e^{-\pi^2/(8z^2)})^9 + (e^{-\pi^2/(8z^2)})^2 5] \quad if \;\; z < 1.18 \tag{11}$$

$$P_{KS}(z) = 1 - 2[(e^{-2z^2}) - (e^{-2z^2})^4 + (e^{-2z^2})^9] \quad if \;\; z \geq 1.18 \tag{12}$$

We then calculate the Q-value as we did in the G-test by using Equation (9)

In our script, we implement the "romberg" function that was already used in our previous assignment to get better results when integrating. We then define our "G_test" function that first calculates the G-value, then the degrees of freedom for our problem (number of bins - 3). Then computes the probability by calling the incomplete γ function from the "scipy" library. The "gammainc" routine already normalizes by diving with $\Gamma(a) = \Gamma(k/2)$, where a is the first term in the "gammainc" input. We, therefore, do not have to make this division in our code.

The function "ks_test" implements the K-S test as it was described before. It calculates the cumulative counts for our data and model and then divides it by the maximum value (last value) to get the cdf. The function then finds the value of the maximum distance between the two (also returns the position for some plots) and then calculates the z value and finally the K-S probability.

```python
import numpy as np
import matplotlib.pyplot as plt
from chisquare import ReadFile, F_n
from open_int import Romberg
from scipy.special import gammainc

def PopulateBins(counts: np.ndarray, bins: np.ndarray, sat: int, res: np.ndarray, m: int
     = 4) -> np.ndarray:
    '''
    populate the bins with the model
    '''
    norm = sat/Romberg(lambda x: F_n(x,norm=1,sat=sat,a=res[0],b=res[1],c=res[2])
    ,0,5,100,m=m)
    y = np.zeros_like(counts)
    for i in range(len(bins)-1):
        y[i]=(Romberg(lambda x: F_n(x,norm=norm,sat=sat,a=res[0],b=res[1],c=res[2])
            ,bins[i],bins[i+1],100,m=3))
    return y

def GTest(counts: np.ndarray, y: np.ndarray) -> tuple:
    '''
    input:
    counts: number of counts in each bin
    y: fitted model evaluated at the center
    of each bin
    output:
    G: the G value
    P: the probability of the G-test
    '''
    sums = 0
    for i in range(len(y)):
        if y[i] !=0 :  # avoiding division by zero
            sums += counts[i]*np.log(counts[i]/y[i])
    G = 2*sums
    k = len(counts)-3
    P = gammainc(k/2,G/2)

    return G,P

def KSTest(counts: np.ndarray, y: np.ndarray) -> tuple:
    '''
    input:
    counts: number of counts in each bin
    y: fitted model evaluated at the center
    of each bin
    output:
    cumul_counts: the cdf of the data
    cumul_y: the cdf of the model
    P_ks: the Probability of K-S test
    pos: position of max difference
    '''
    if len(counts)!=len(y):
        raise ValueError("Length of counts must be equal to length of y")

    cumul_counts=np.zeros_like(counts)
    cumul_counts[0]=counts[0]
    for i in range(1,len(counts)):
        cumul_counts[i]=cumul_counts[i-1]+counts[i]
    cumul_counts = cumul_counts/cumul_counts[-1]  #normalize

    cumul_y=np.zeros_like(y)
    cumul_y[0]=y[0]
    for i in range(1,len(y)):
        cumul_y[i]=cumul_y[i-1]+y[i]
    cumul_y = cumul_y/cumul_y[-1]

    ks=np.amax(np.abs(cumul_counts-cumul_y))
    pos=np.argmax(np.abs(cumul_counts-cumul_y)) #normalize

```

```
    N = len(counts)
    ks = (np.sqrt(N)+.12+.11/np.sqrt(N))*ks

    if ks < 1.18:
        exp = np.exp(-np.pi**2/(8*ks**2))
        P_ks = ((np.sqrt(2*np.pi))/ks)*(exp+exp**9+exp**25)
    else:
        exp = np.exp(-2*ks**2)
        P_ks = 1 - 2*(exp-exp**4+exp**9)
    return cumul_counts, cumul_y,P_ks,pos

def KSPlot(bins: np.ndarray, cumul_counts: np.ndarray, cumul_y: np.ndarray, x: np.
    ndarray,
        pos: int, name: str = 'test', types: str = 'png', dpi: int = 300):
    '''
    plotting function
    '''
    plt.bar(bins[:-1] + np.diff(bins) / 2, cumul_counts, np.diff(bins),color='blue',
    label='data cdf')
    plt.plot(x,cumul_y,color='green',label='model cdf')
    plt.scatter(x,cumul_y,color='green')
    plt.vlines(x[pos],cumul_counts[pos],cumul_y[pos],color='red',label='max difference')
    plt.legend()
    plt.savefig(f'plots/{name}.{types}',dpi=dpi)
    plt.close()

if __name__ == '__main__':

    #reading the data
    radius1, nhalo1 = ReadFile('data/satgals_m11.txt')
    radius2, nhalo2 = ReadFile('data/satgals_m12.txt')
    radius3, nhalo3 = ReadFile('data/satgals_m13.txt')
    radius4, nhalo4 = ReadFile('data/satgals_m14.txt')
    radius5, nhalo5 = ReadFile('data/satgals_m15.txt')

    # binning the data
    count1, bins1 = np.histogram(radius1, bins=20)
    count2, bins2 = np.histogram(radius2, bins=20)
    count3, bins3 = np.histogram(radius3, bins=20)
    count4, bins4 = np.histogram(radius4, bins=20)
    count5, bins5 = np.histogram(radius5, bins=20)

    # calculating sat for each file
    # this time sat = len(radius)
    sat1 = len(radius1)
    sat2 = len(radius2)
    sat3 = len(radius3)
    sat4 = len(radius4)
    sat5 = len(radius5)

    # reading the results of parameters after fitting
    chi_res = np.load('output/chi_results.npy')
    pois_res = np.load('output/pois_results.npy')

    # getting the models
    x1 = bins1[:-1] + np.diff(bins1) / 2
    x2 = bins2[:-1] + np.diff(bins2) / 2
    x3 = bins3[:-1] + np.diff(bins3) / 2
    x4 = bins4[:-1] + np.diff(bins4) / 2
    x5 = bins5[:-1] + np.diff(bins5) / 2

    y1_chi = PopulateBins(count1,bins1,sat1,chi_res[0],m=4)
    y2_chi = PopulateBins(count2,bins2,sat2,chi_res[1],m=3)
    y3_chi = PopulateBins(count3,bins3,sat3,chi_res[2],m=3)
    y4_chi = PopulateBins(count4,bins4,sat4,chi_res[3],m=4)
    y5_chi = PopulateBins(count5,bins5,sat5,chi_res[4],m=4)

    y1_pois = PopulateBins(count1,bins1,sat1,pois_res[0],m=5)
    y2_pois = PopulateBins(count2,bins2,sat2,pois_res[1],m=4)
    y3_pois = PopulateBins(count3,bins3,sat3,pois_res[2],m=4)
```

```
    y4_pois = PopulateBins(count4,bins4,sat4,pois_res[3],m=4)
    y5_pois = PopulateBins(count5,bins5,sat5,pois_res[4],m=4)

    # G-test
    G1_chi, PG1_chi = GTest(count1,y1_chi)
    G2_chi, PG2_chi = GTest(count2,y2_chi)
    G3_chi, PG3_chi = GTest(count3,y3_chi)
    G4_chi, PG4_chi = GTest(count4,y4_chi)
    G5_chi, PG5_chi = GTest(count5,y5_chi)

    G1_pois, PG1_pois = GTest(count1,y1_pois)
    G2_pois, PG2_pois = GTest(count2,y2_pois)
    G3_pois, PG3_pois = GTest(count3,y3_pois)
    G4_pois, PG4_pois = GTest(count4,y4_pois)
    G5_pois, PG5_pois = GTest(count5,y5_pois)

    # k-s test
    cumul_counts1_chi, cumul_y1_chi,ks1_chi,pos1_chi = KSTest(count1,y1_chi)
    cumul_counts2_chi, cumul_y2_chi,ks2_chi,pos2_chi = KSTest(count2,y2_chi)
    cumul_counts3_chi, cumul_y3_chi,ks3_chi,pos3_chi = KSTest(count3,y3_chi)
    cumul_counts4_chi, cumul_y4_chi,ks4_chi,pos4_chi = KSTest(count4,y4_chi)
    cumul_counts5_chi, cumul_y5_chi,ks5_chi,pos5_chi = KSTest(count5,y5_chi)

    cumul_counts1_pois, cumul_y1_pois,ks1_pois,pos1_pois = KSTest(count1,y1_pois)
    cumul_counts2_pois, cumul_y2_pois,ks2_pois,pos2_pois = KSTest(count2,y2_pois)
    cumul_counts3_pois, cumul_y3_pois,ks3_pois,pos3_pois = KSTest(count3,y3_pois)
    cumul_counts4_pois, cumul_y4_pois,ks4_pois,pos4_pois = KSTest(count4,y4_pois)
    cumul_counts5_pois, cumul_y5_pois,ks5_pois,pos5_pois = KSTest(count5,y5_pois)

    # plots for k-s test
    KSPlot(bins1,cumul_counts1_chi,cumul_y1_chi,x1,pos1_chi,name='ks_chi_1')
    KSPlot(bins2,cumul_counts2_chi,cumul_y2_chi,x2,pos2_chi,name='ks_chi_2')
    KSPlot(bins3,cumul_counts3_chi,cumul_y3_chi,x3,pos3_chi,name='ks_chi_3')
    KSPlot(bins4,cumul_counts4_chi,cumul_y4_chi,x4,pos4_chi,name='ks_chi_4')
    KSPlot(bins5,cumul_counts5_chi,cumul_y5_chi,x5,pos5_chi,name='ks_chi_5')

    KSPlot(bins1,cumul_counts1_pois,cumul_y1_pois,x1,pos1_pois,name='ks_pois_1')
    KSPlot(bins2,cumul_counts2_pois,cumul_y2_pois,x2,pos2_pois,name='ks_pois_2')
    KSPlot(bins3,cumul_counts3_pois,cumul_y3_pois,x3,pos3_pois,name='ks_pois_3')
    KSPlot(bins4,cumul_counts4_pois,cumul_y4_pois,x4,pos4_pois,name='ks_pois_4')
    KSPlot(bins5,cumul_counts5_pois,cumul_y5_pois,x5,pos5_pois,name='ks_pois_5')

    # printing the results
    print(f'dataset m11')
    print(f'chi^2 G = {G1_chi}')
    print(f'chi^2 G test Q = {1-PG1_chi}')
    print(f'poisson G = {G1_pois}')
    print(f'poisson G test Q = {1-PG1_pois}')
    print(f'chi^2 k-s test Q = {1-ks1_chi}')
    print(f'poisson k-s Q = {1-ks1_pois}')
    print(f'\ndataset m12')
    print(f'chi^2 G = {G2_chi}')
    print(f'chi^2 G test Q = {1-PG2_chi}')
    print(f'poisson G = {G2_pois}')
    print(f'poisson G test Q = {1-PG2_pois}')
    print(f'chi^2 k-s test Q = {1-ks2_chi}')
    print(f'poisson k-s test Q = {1-ks2_pois}')
    print(f'\ndataset m13')
    print(f'chi^2 G = {G3_chi}')
    print(f'chi^2 G test Q = {1-PG3_chi}')
    print(f'poisson G = {G3_pois}')
    print(f'poisson G test Q = {1-PG3_pois}')
    print(f'chi^2 k-s test Q = {1-ks3_chi}')
    print(f'poisson k-s test Q = {1-ks3_pois}')
    print(f'\ndataset m14')
    print(f'chi^2 G = {G4_chi}')
    print(f'chi^2 G test Q = {1-PG4_chi}')
    print(f'poisson G = {G4_pois}')
    print(f'poisson G test Q = {1-PG4_pois}')
    print(f'chi^2 k-s test Q = {1-ks4_chi}')
```

```
    print(f'poisson k-s test Q = {1-ks4_pois}')
    print(f'\ndataset m15')
    print(f'chi^2 G = {G5_chi}')
    print(f'chi^2 G test Q = {1-PG5_chi}')
    print(f'poisson G = {G5_pois}')
    print(f'poisson G test Q = {1-PG5_pois}')
    print(f'chi^2 k-s test Q = {1-ks5_chi}')
    print(f'poisson k-s test Q = {1-ks5_pois}')
```

stat_test.py

In the main part of our script, we compute the test values. This time we will work with the actual number of counts instead of the pdf which was used when we were fitting our models. We, therefore, bin our data with no normalization and define our average number of galaxies for each dataset to be equal to the actual number of galaxies in each dataset. We then get the x position of the bin centers for each data set. After that, we calculate the model in this position by integrating over each bin. This job is done for all the datasets and for both the χ^2 and Poisson minimization. Finally, we call the test functions and output the results. We also create some plots for better visualization.

```
dataset m11
chi^2 G = 146.87860059372406
chi^2 G test Q = 0.0
poisson G = 2.5164469878842217
poisson G test Q = 0.9999806945960857
chi^2 k-s test Q = 1.0
poisson k-s Q = 1.0

dataset m12
chi^2 G = 42.12958916141422
chi^2 G test Q = 0.0006423825630412772
poisson G = 9.232189705313631
poisson G test Q = 0.9326945350680935
chi^2 k-s test Q = 1.0
poisson k-s test Q = 1.0

dataset m13
chi^2 G = 29.722170087399313
chi^2 G test Q = 0.028419725947152807
poisson G = 18.26744780379846
poisson G test Q = 0.372159400963064
chi^2 k-s test Q = 1.0
poisson k-s test Q = 1.0

dataset m14
chi^2 G = 44.377213942015125
chi^2 G test Q = 0.00030126544168485037
poisson G = 45.74657896215117
poisson G test Q = 0.0001883593934713934
chi^2 k-s test Q = 1.0
poisson k-s test Q = 1.0

dataset m15
chi^2 G = 35.53129661682164
chi^2 G test Q = 0.005294198451321508
poisson G = 24.925083339804875
poisson G test Q = 0.09639908224762805
chi^2 k-s test Q = 1.0
poisson k-s test Q = 1.0
```

output/stat_test.txt

The output of our script is very interesting as the Q-values of the G-test suggest that our fitting is not that great for almost all of the cases. We know that this is not the case and the Q-values from the K-S test confirm that we indeed have a great fit for all our models. The problem with the values of the G-test probably has to do with errors in the integration as experimentation using our script has shown that is indeed very sensitive to different calculations of the normalizing parameter A. This is the main reason we decided to use the “romberg” function to integrate instead of the more simple “trapezoid” function.

It is however still evident that the Poisson log-likelihood minimization worked better in all the cases except for m14 where the G-values were very similar and slightly better for the χ^2 minimization. We can therefore conclude that the minimization of the negative Poisson log-likelihood worked much better and fitted our observations with more detail.

We can also see how great the match between the observed and model cdf is in Figures 6 and 7. The maximum difference between the two for each case is shown with a red vertical line that is very easy to miss because the match is almost perfect which makes the line very small.

References

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. (2007). *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge University Press.

A Extra Plots

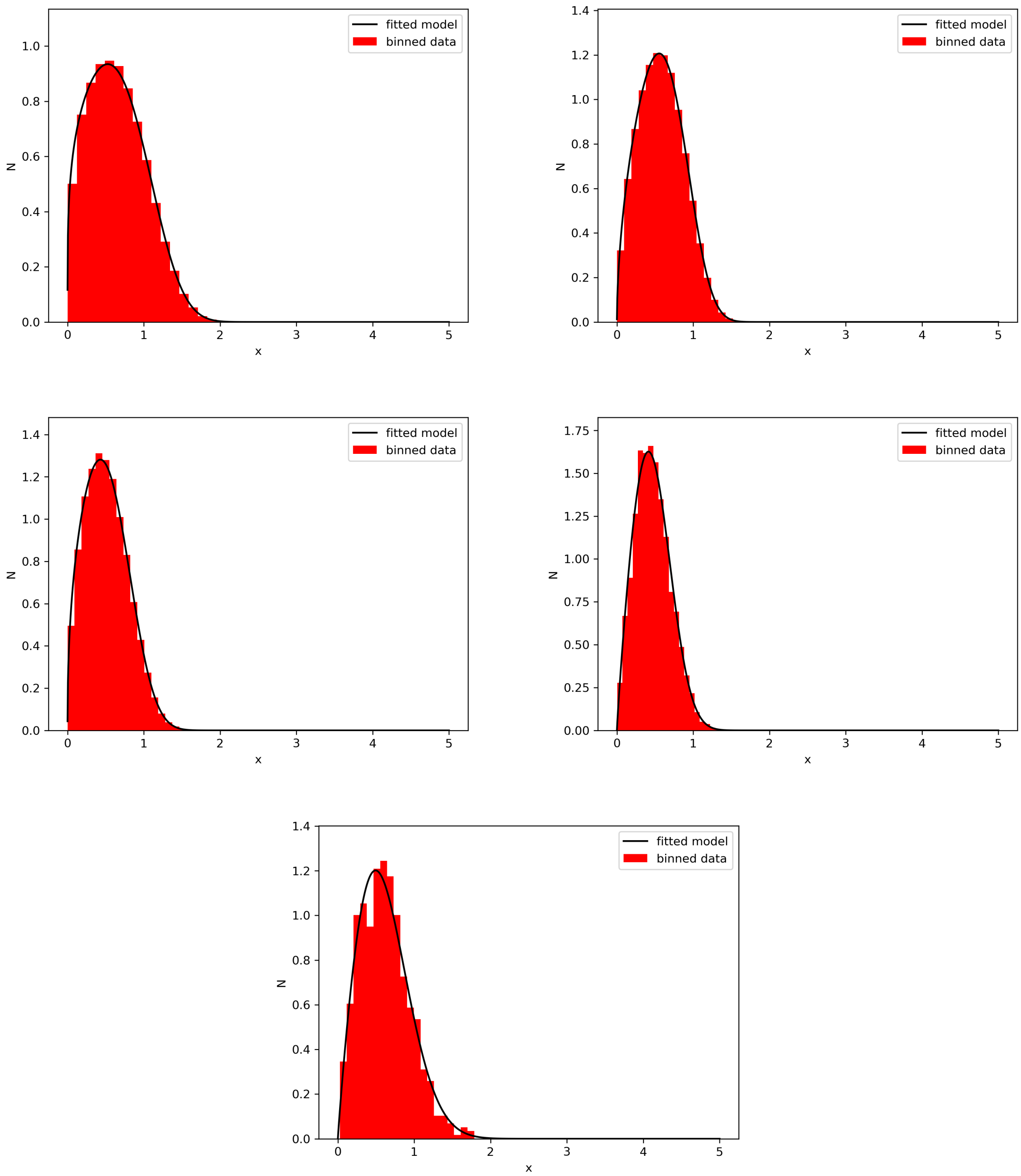

Figure 4: The binned data with the best-fit model for each dataset in real space. To fit the model we minimized χ^2 function. We can see that the model greatly fits our data in all five cases.

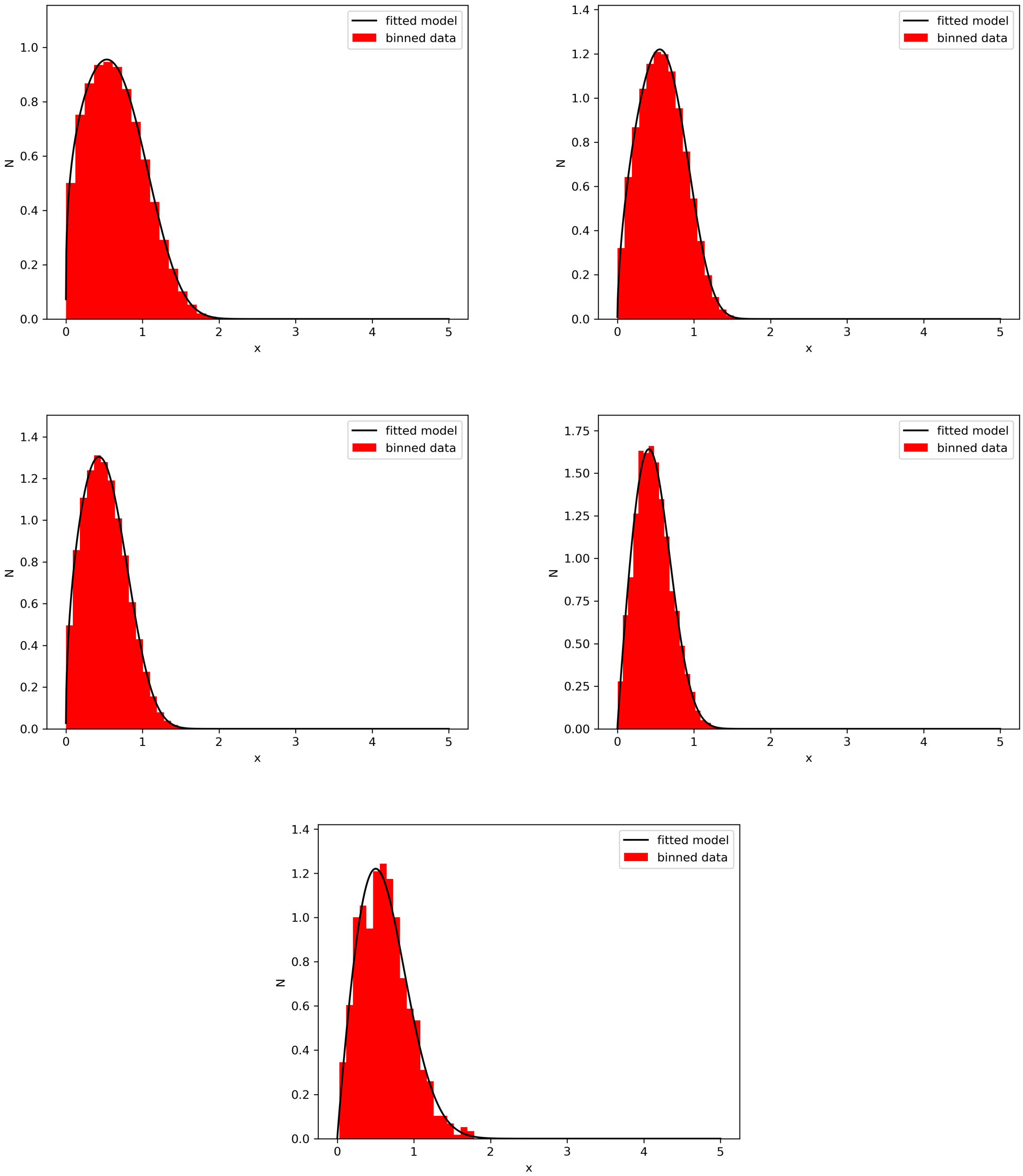

Figure 5: The binned data with the best-fit model for each dataset in real space. To fit the model we minimized Poisson log-likelihood. We can see that the model greatly fits our data in all five cases.

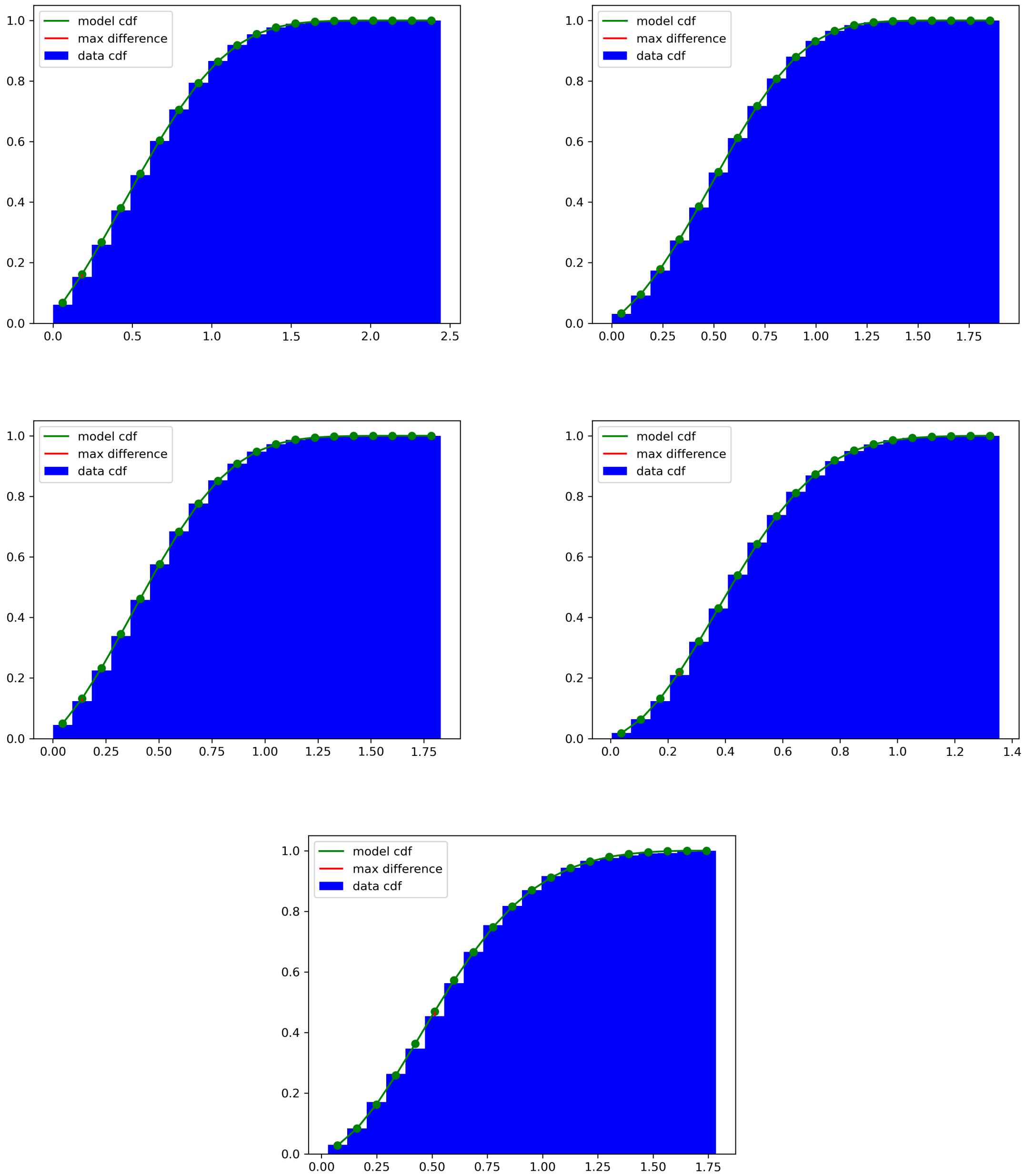

Figure 6: The cumulative distribution for both our binned data and the best-fit model using the χ^2 minimization algorithm. We can see that there is a great match between them in all five cases as their maximum absolute difference is very small. This leads to a very high Q-value in the K-S test.

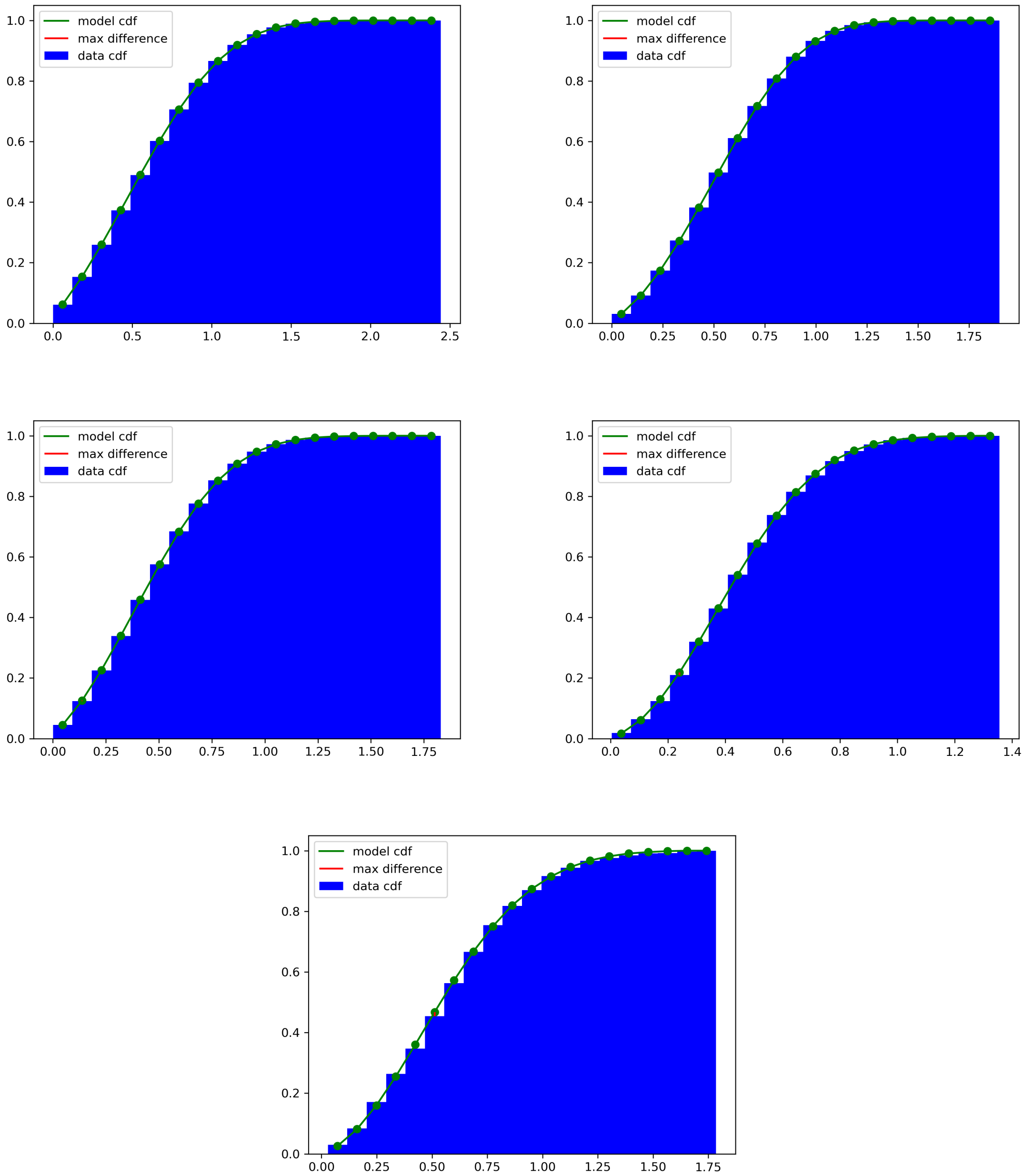

Figure 7: The cumulative distribution for both our binned data and the best-fit model. The models were fit by minimizing the Poisson loglikelihood. We can see that there is a great match between them in all five cases as their maximum absolute difference is very small. This leads to a very high Q-value in the K-S test.