

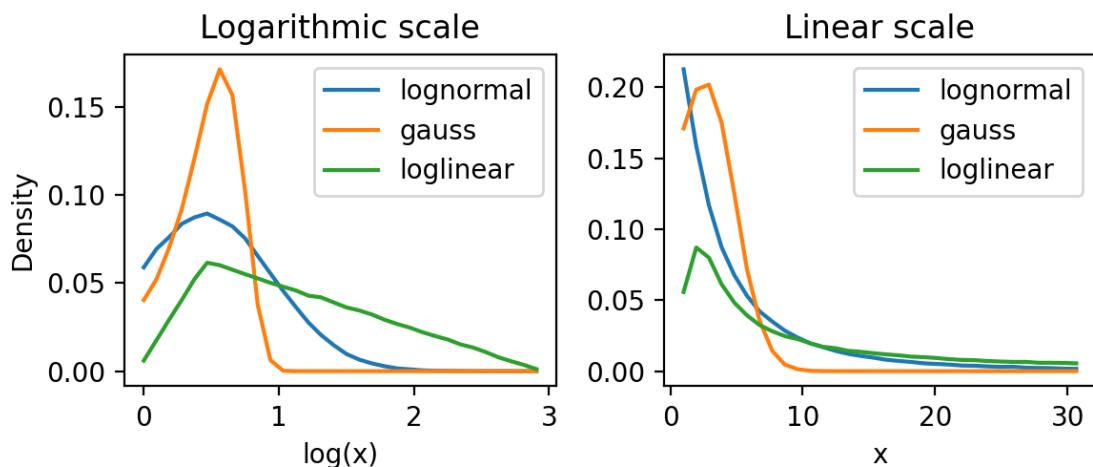
supermodel

December 29, 2021

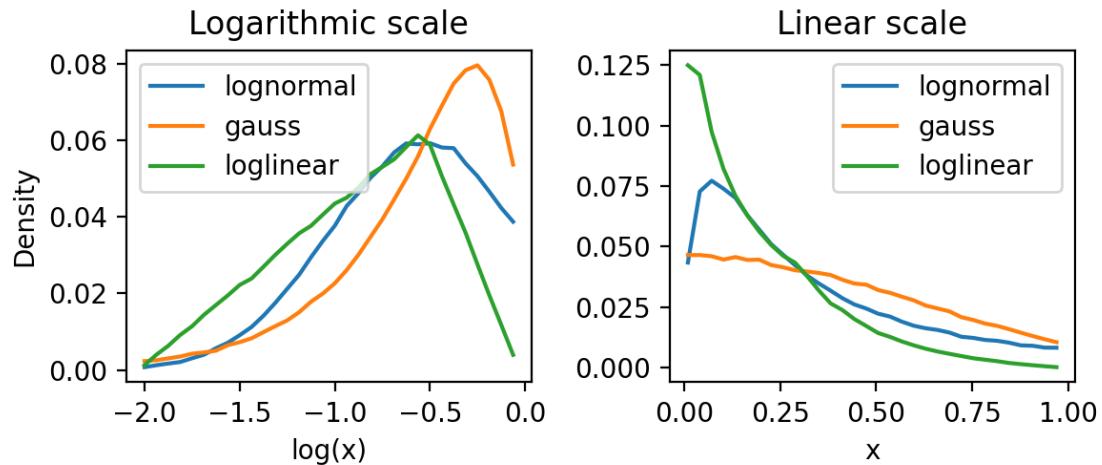
```
[1]: from pca import cluster
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

# primeri vseh treh porazdelitev v primeru povprečja večjega in manjšega od 1,
# ob tem so najprej predstavljene porazdelitve za N v primeru večje možnosti
# za širitev na druge planete
```

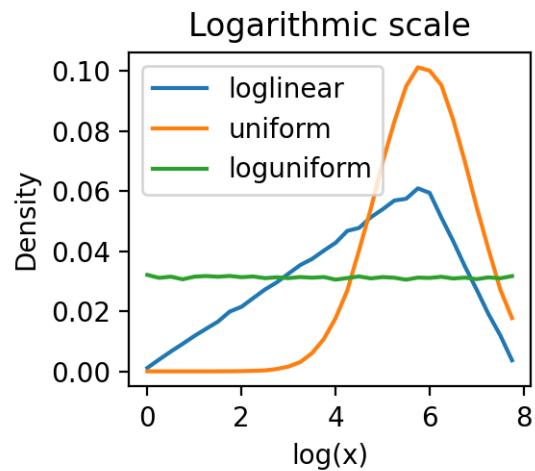
Distributions of x from 1 to 1000 with mean (peak) at 3.16



Distributions of x from 0.01 to 1 with mean (peak) at 0.32



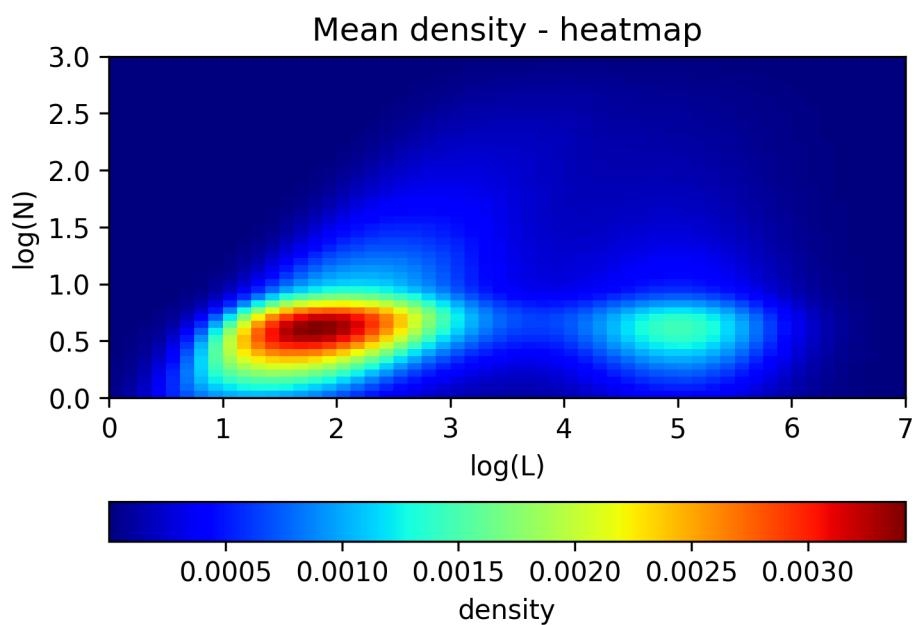
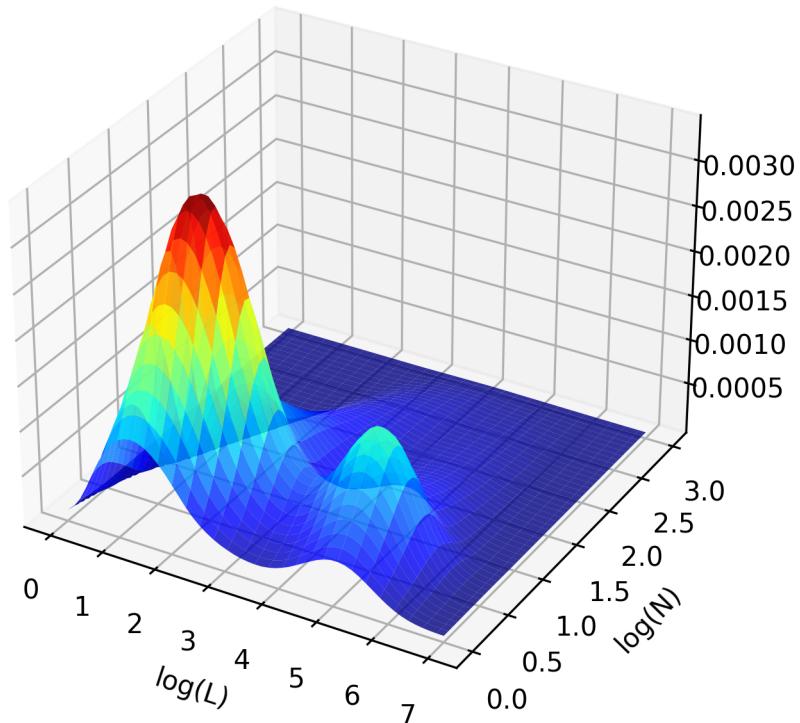
Distributions of x from 1 to 1e8 with mean (peak) at 1e6

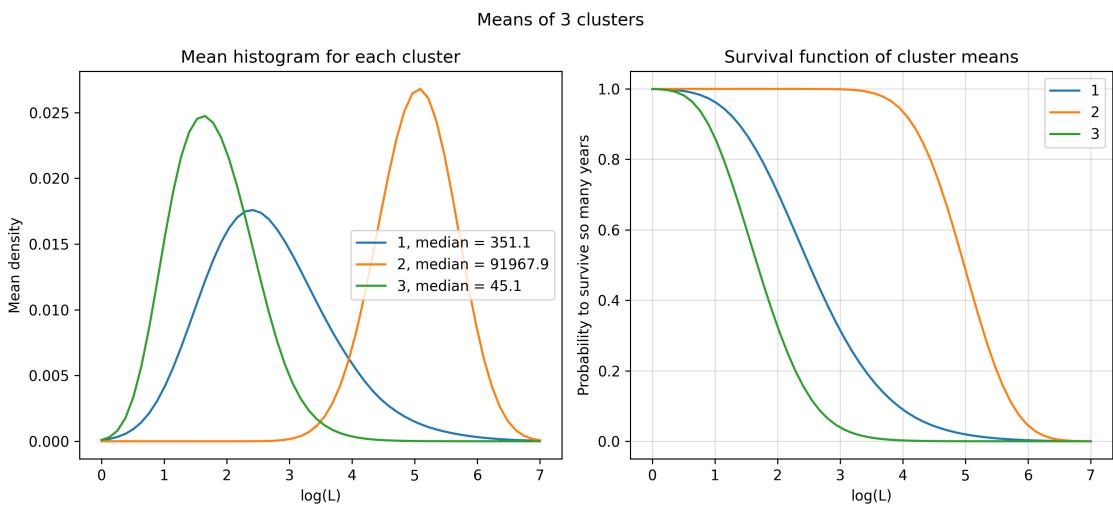
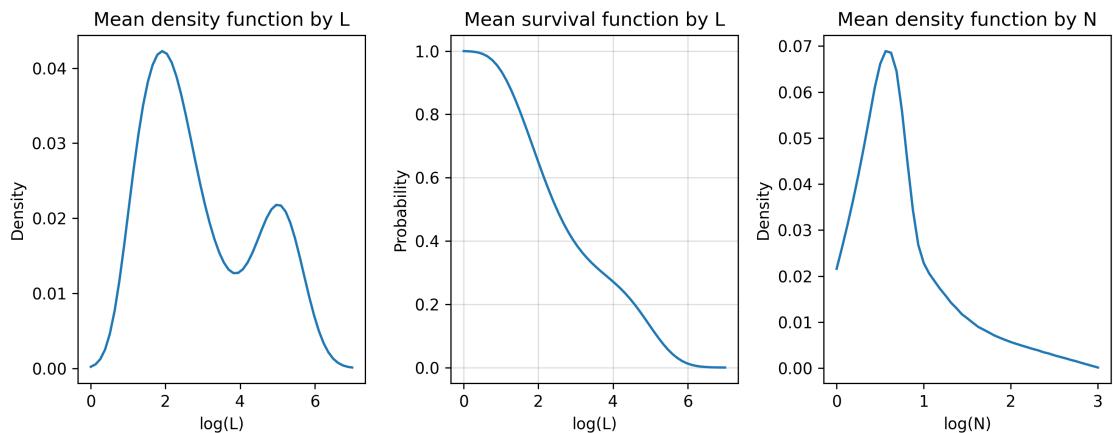


```
[2]: cluster(model=0, ks=[3, 5], supermodel=1) # supermodel  
cluster(model=0, ks=[5], supermodel=2) # supermodel 2
```

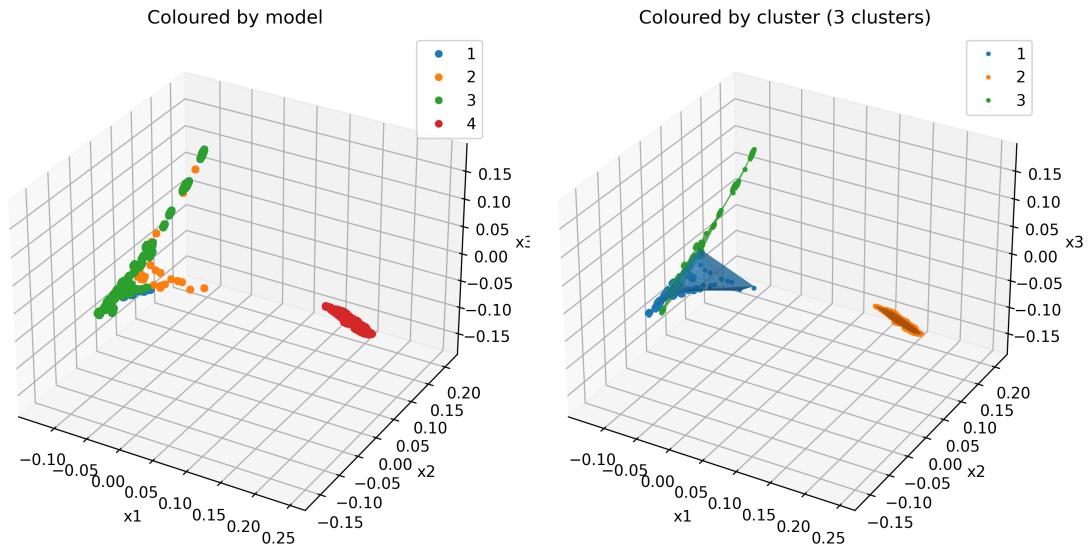
Supermodel

Mean density

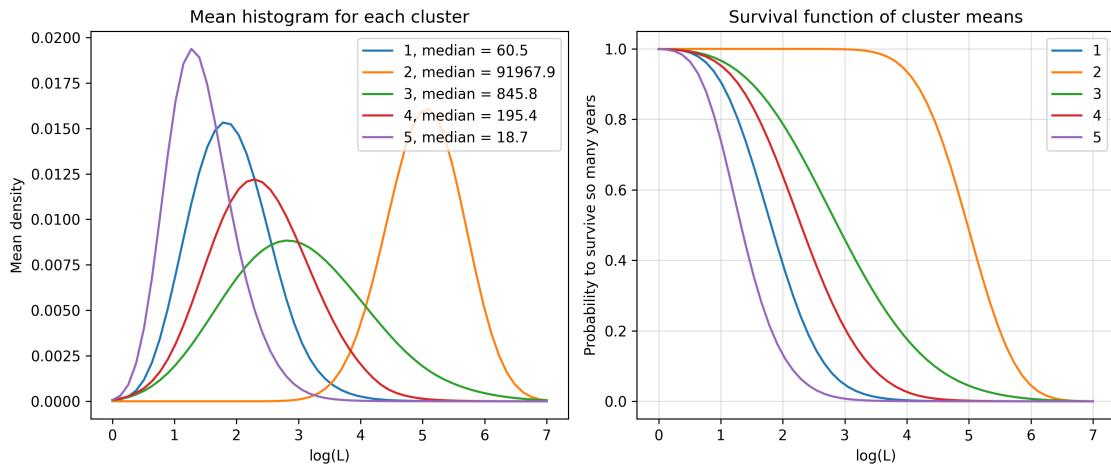




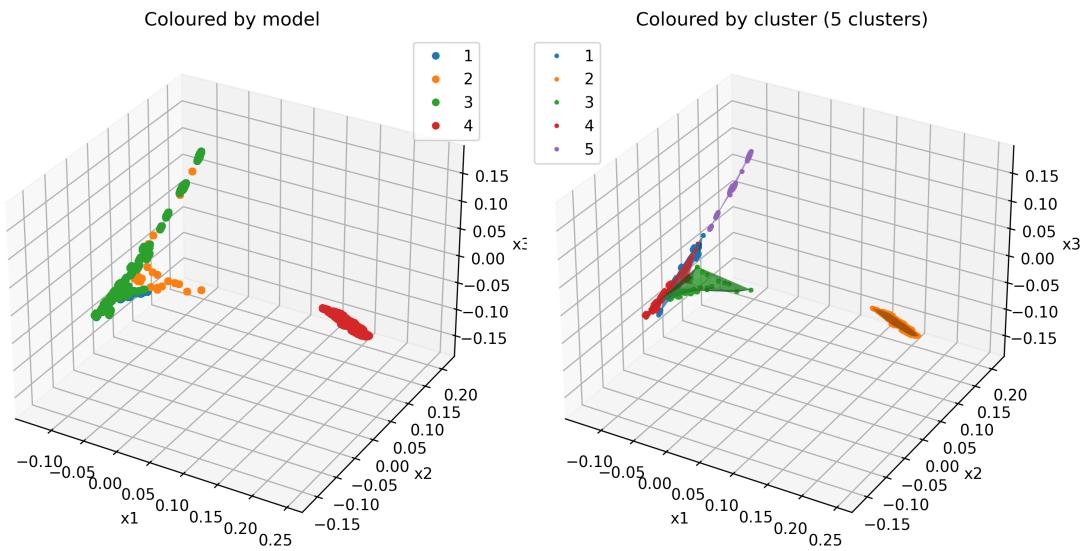
Supermodel after Principal component analysis (PCA)



Means of 5 clusters

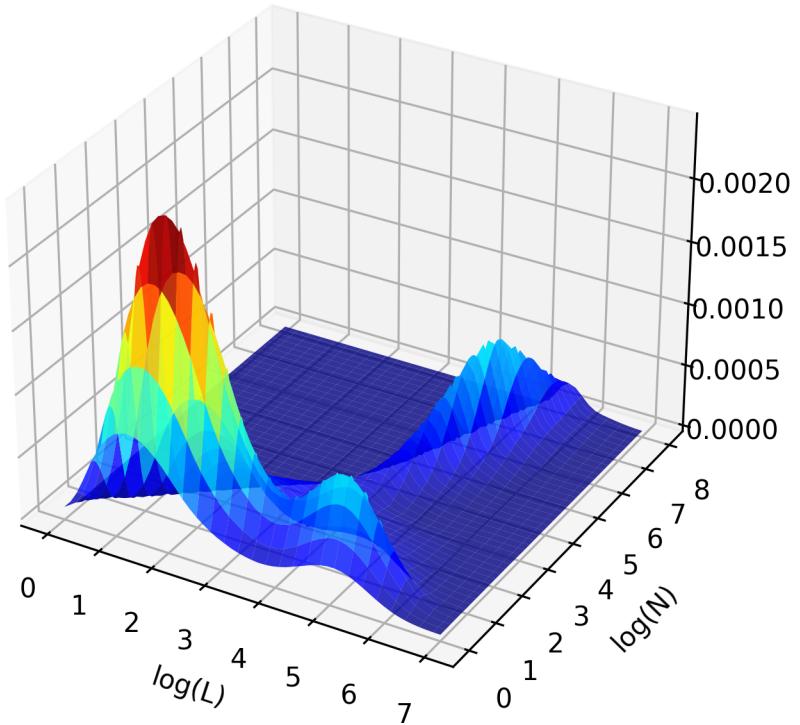


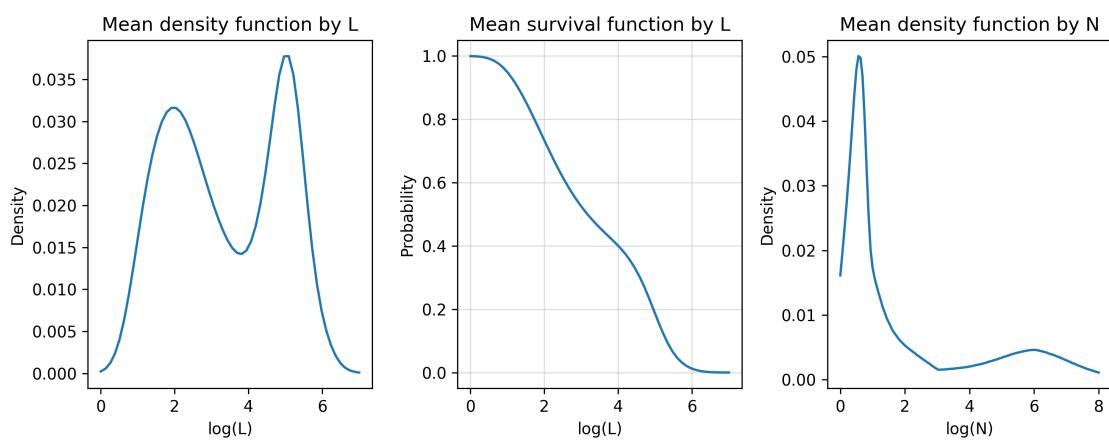
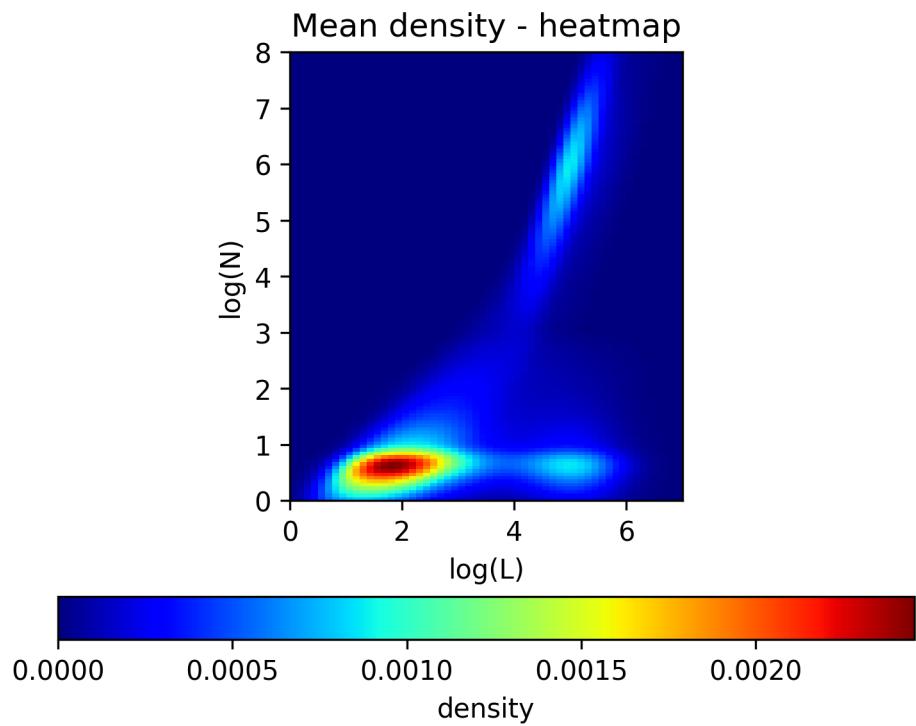
Supermodel after Principal component analysis (PCA)

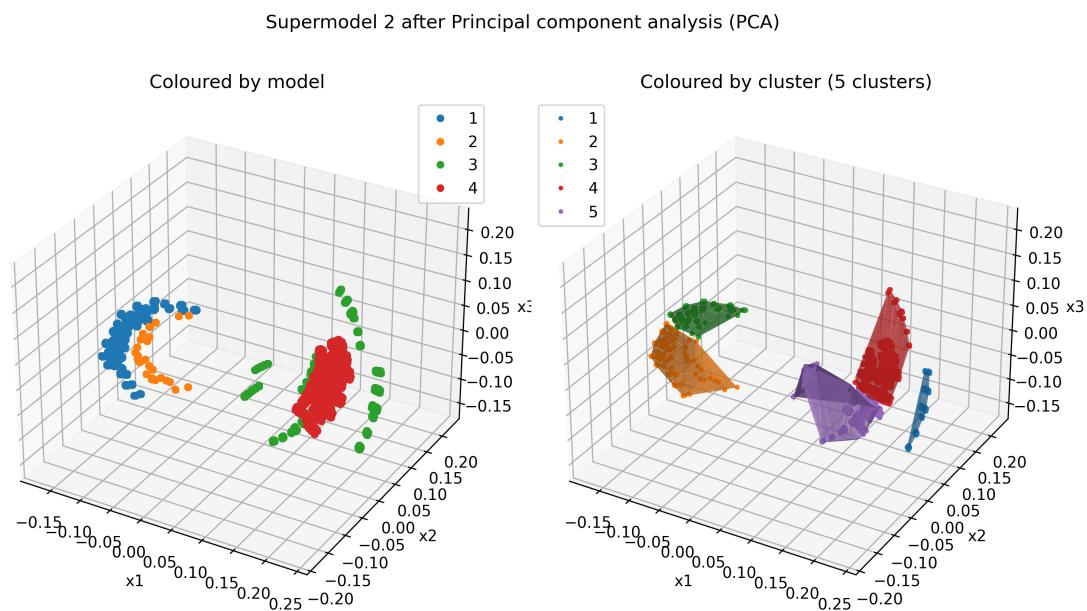
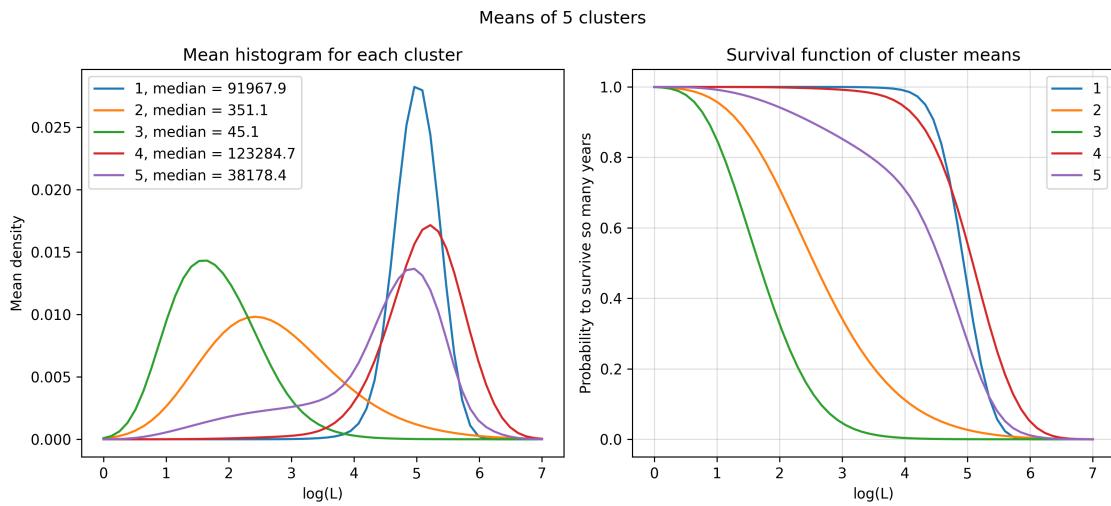


Supermodel 2

Mean density



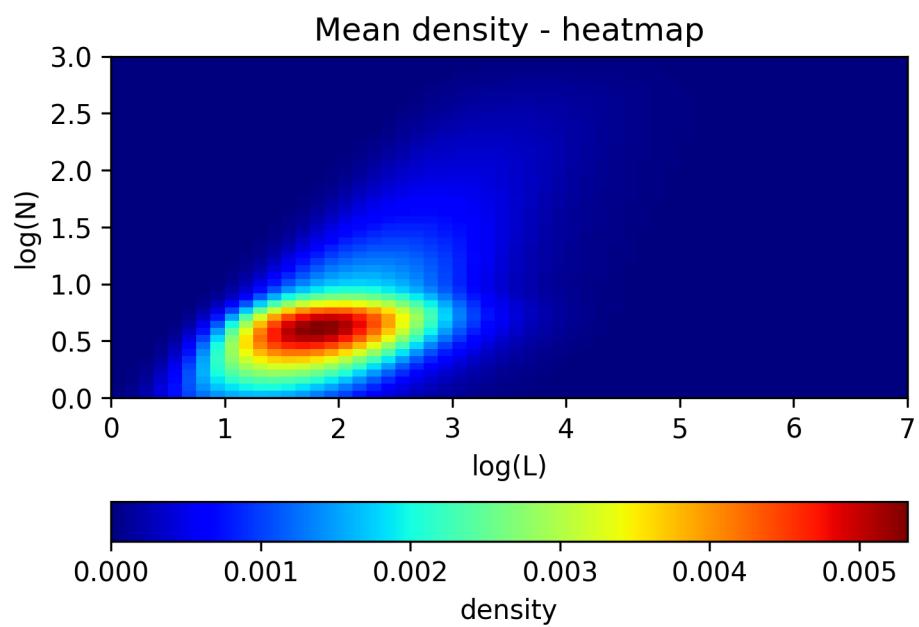
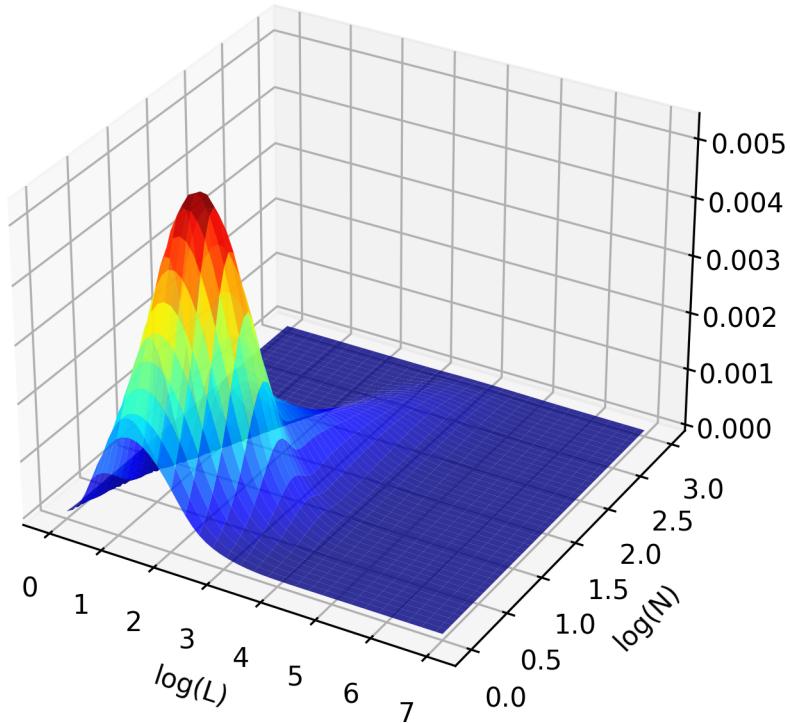


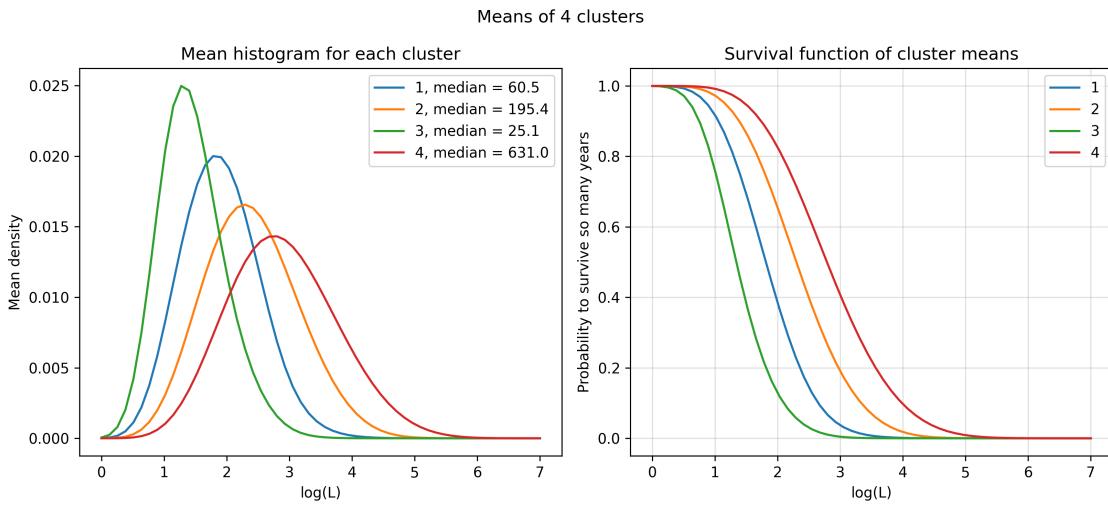
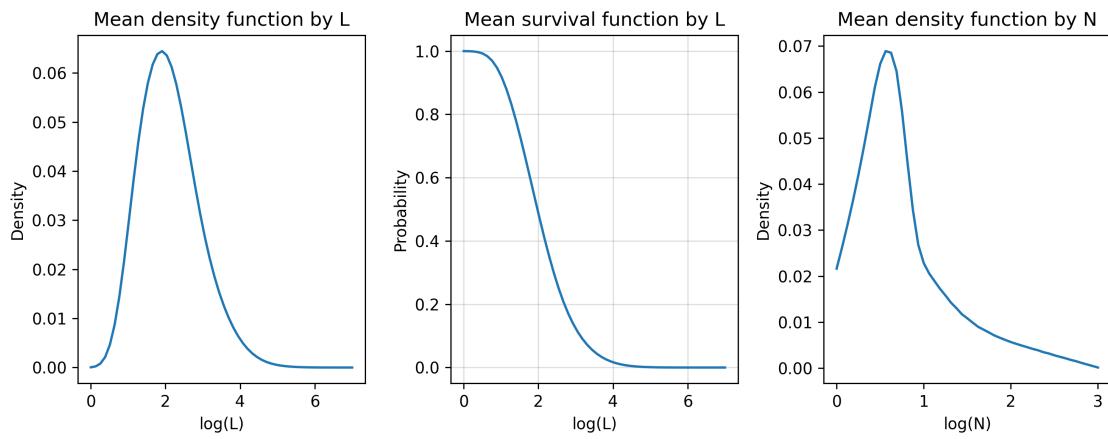


```
[3] : cluster(model=1, ks=[4]) # model 1
```

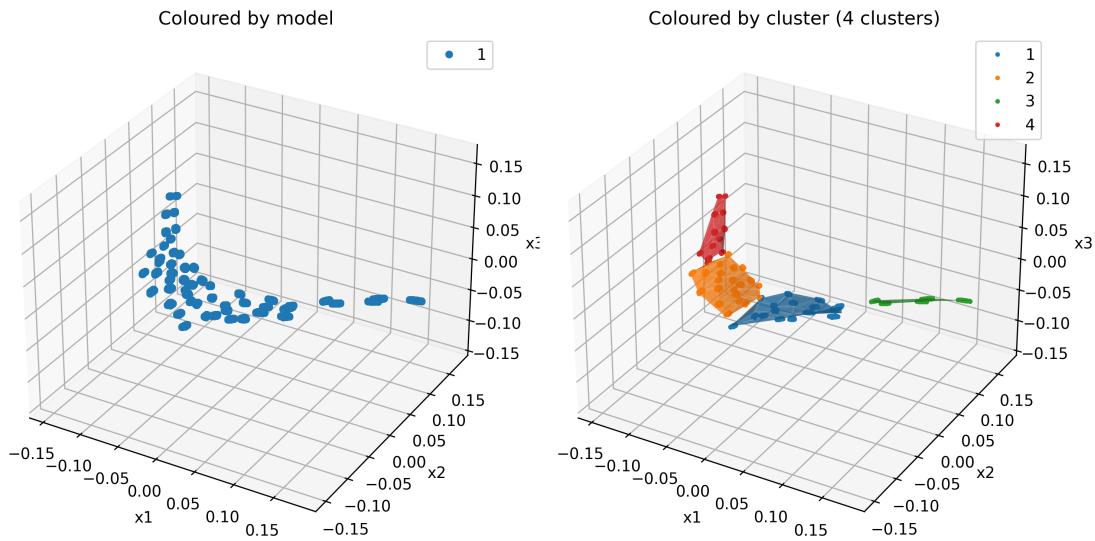
Model I

Mean density

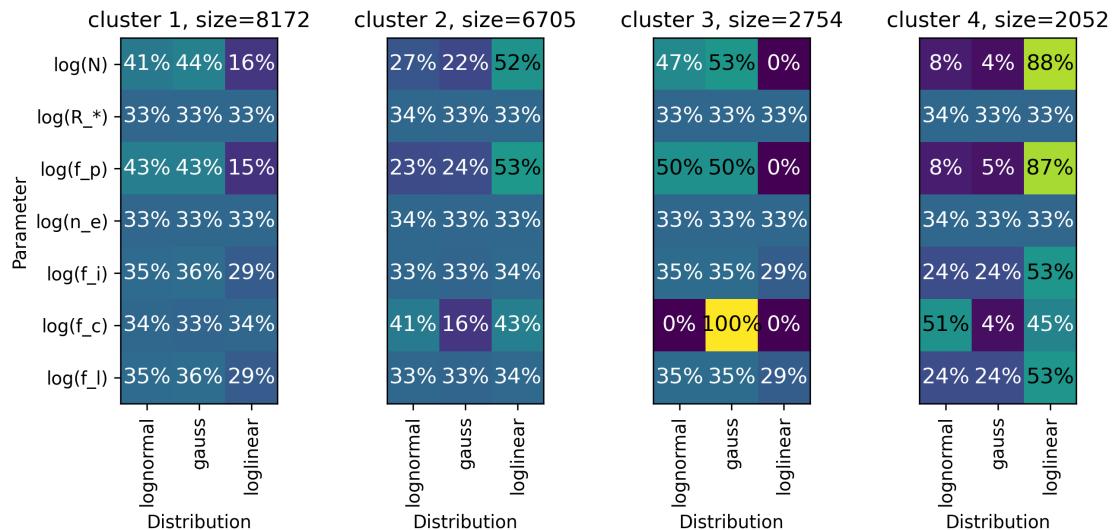




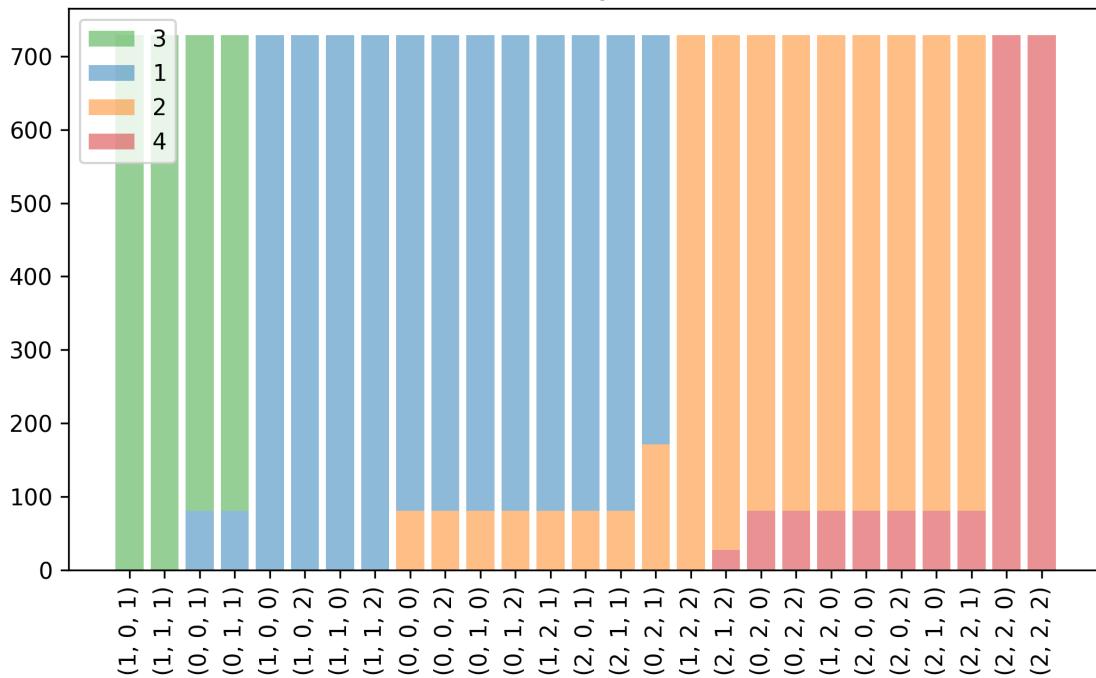
Model I after Principal component analysis (PCA)



Percent of submodels in model 1 distributed with particular distribution on particular parameter in each of 4 clusters



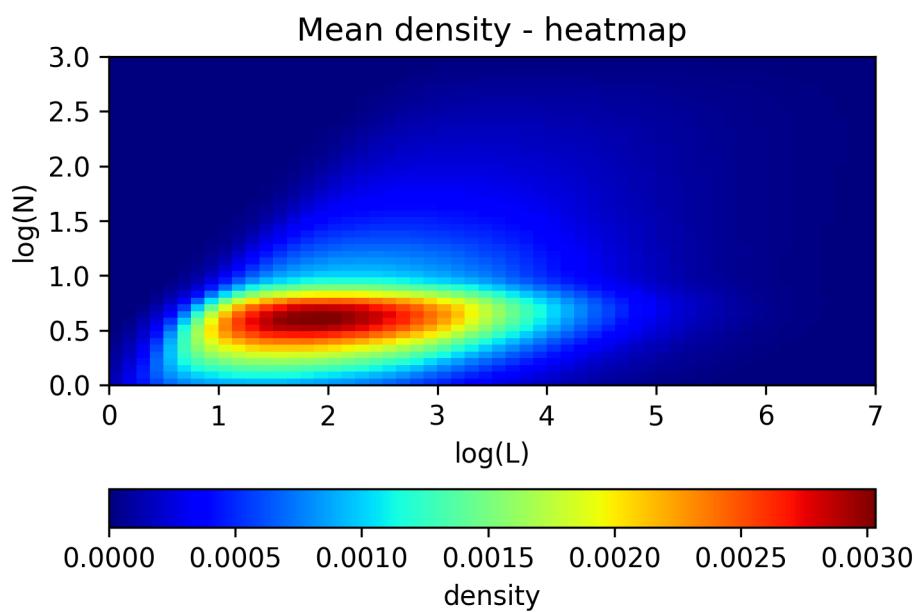
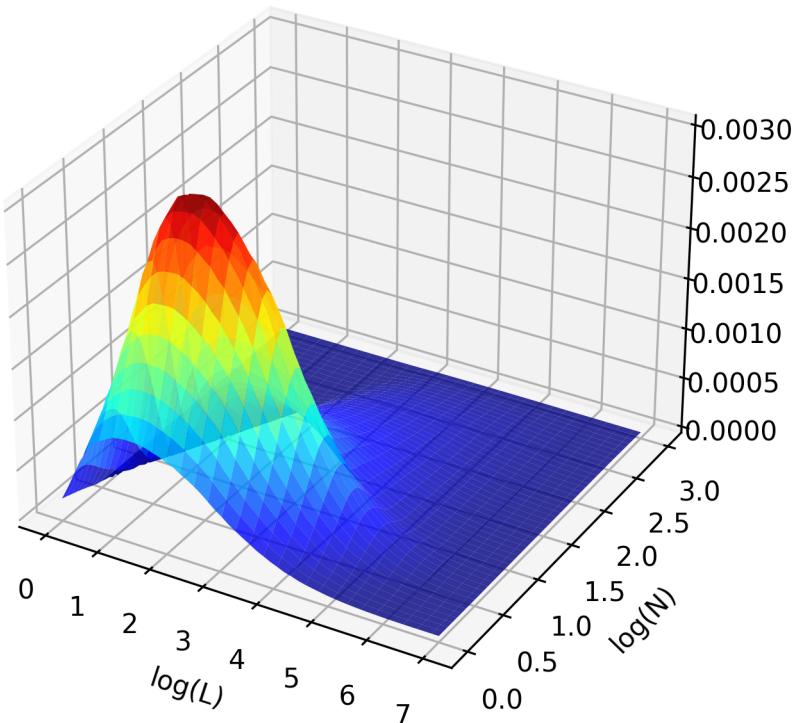
Percentage of distributions
 $(\log(N), \log(f_p), \log(f_c))$ by (lognormal, gauss, loglinear)
 coloured by cluster

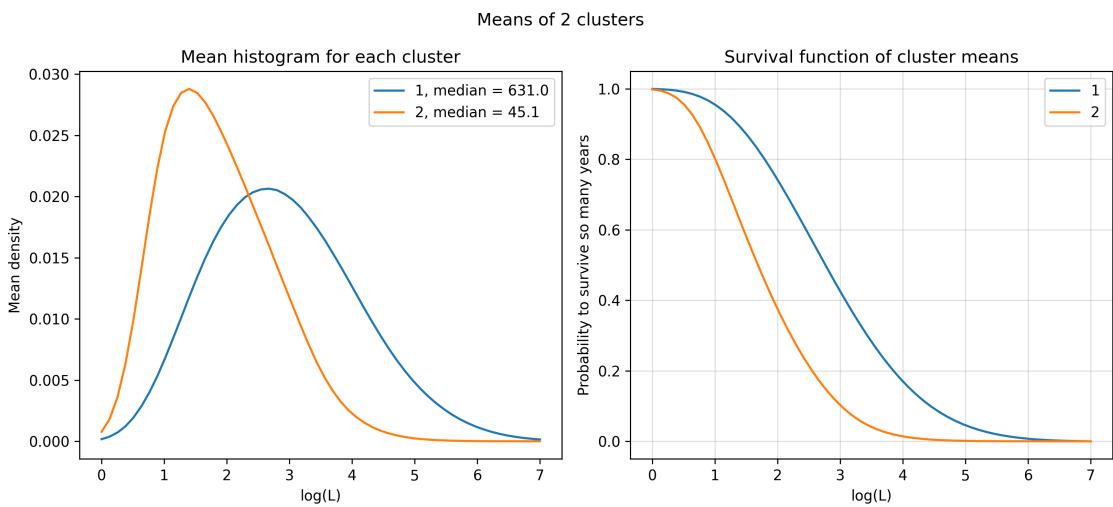
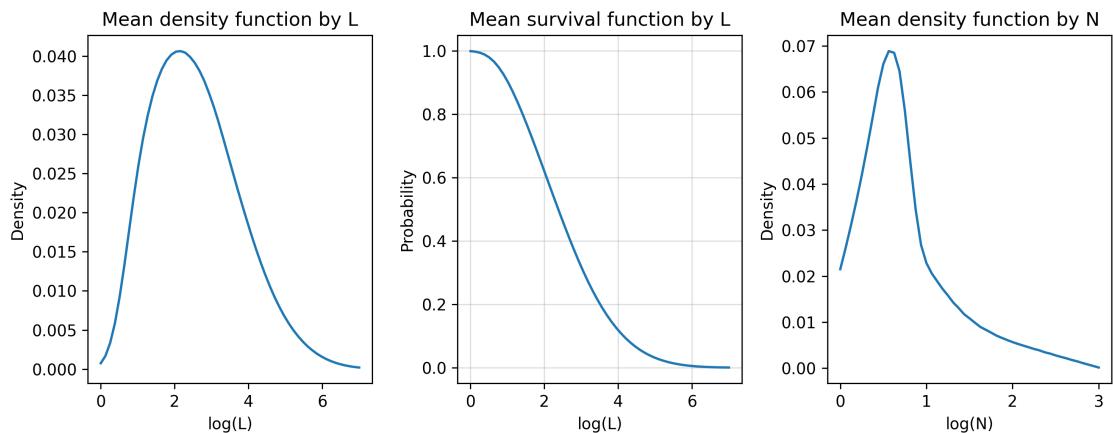


```
[4]: cluster(model=2, ks=[2]) # model 2
```

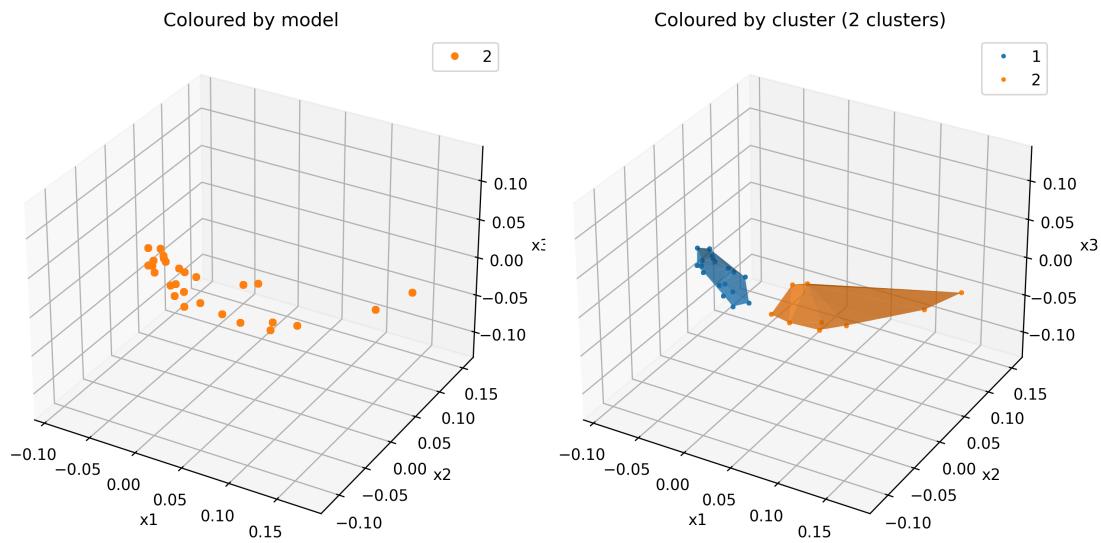
Model II

Mean density





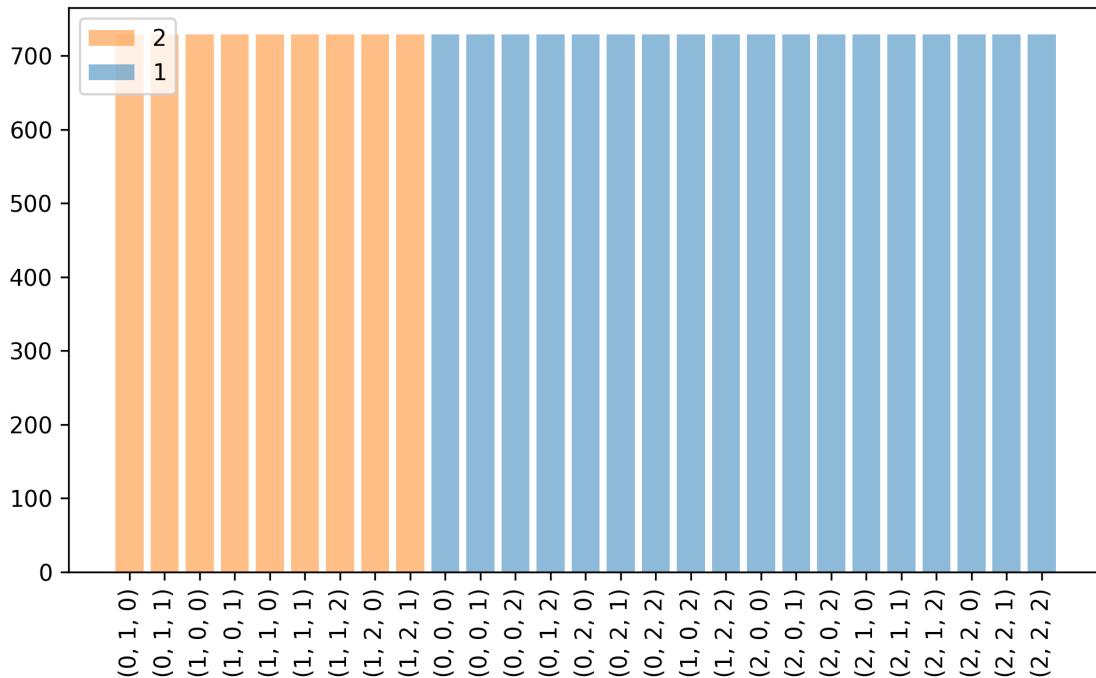
Model II after Principal component analysis (PCA)



Percent of submodels in model 2 distributed with particular distribution on particular parameter in each of 2 clusters



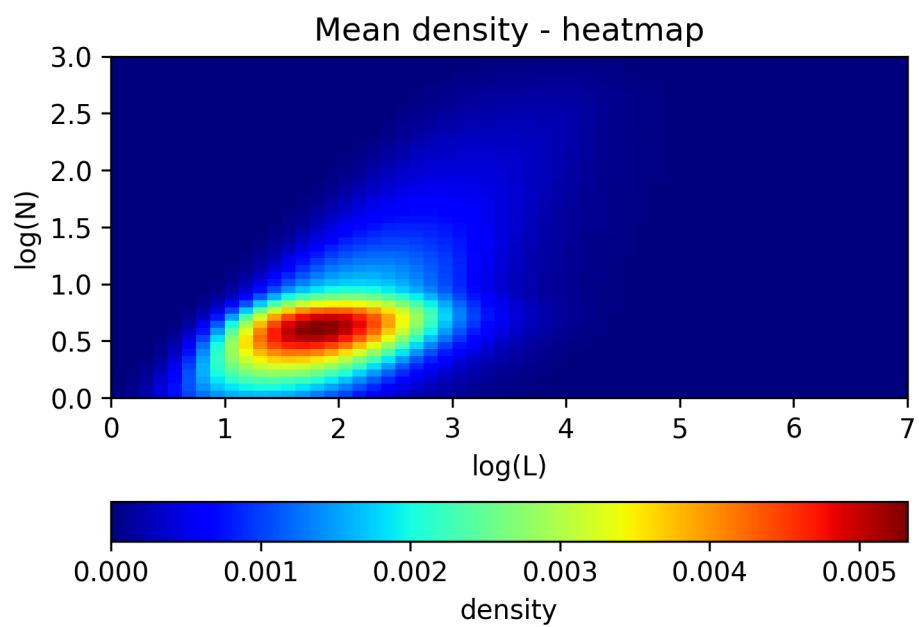
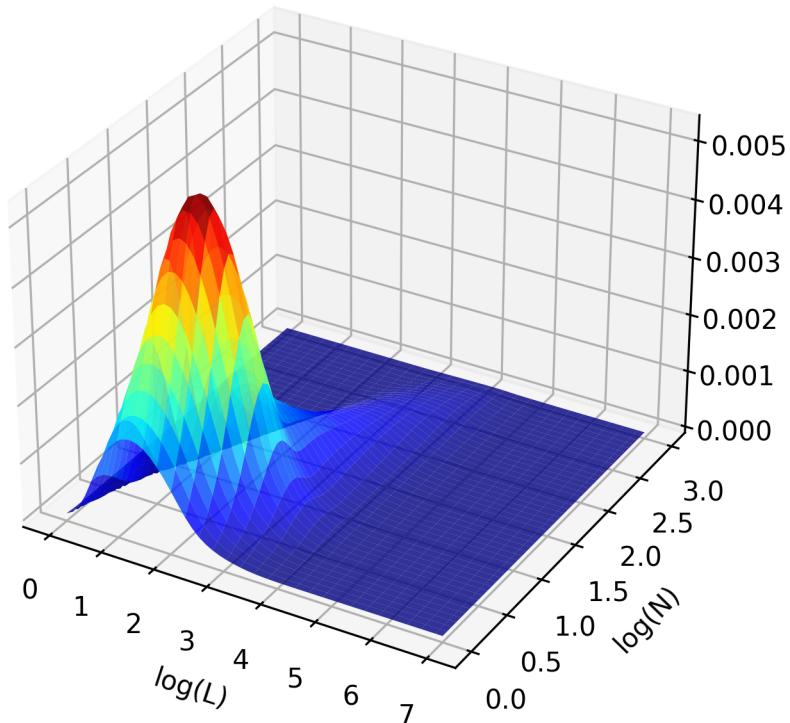
Percentage of distributions
 $(\log(f_b), \log(f_a), \log(N))$ by (lognormal, gauss, loglinear)
 coloured by cluster

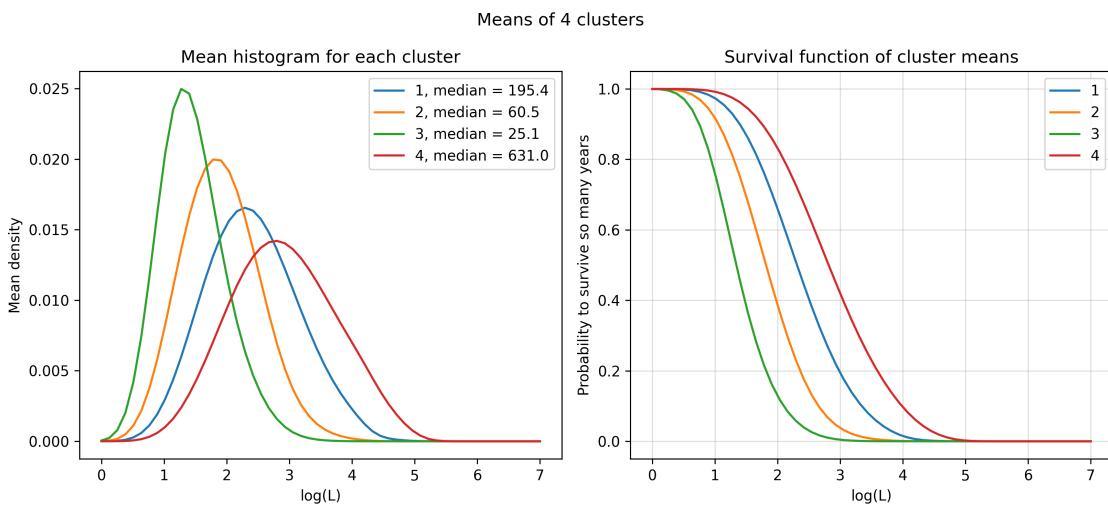
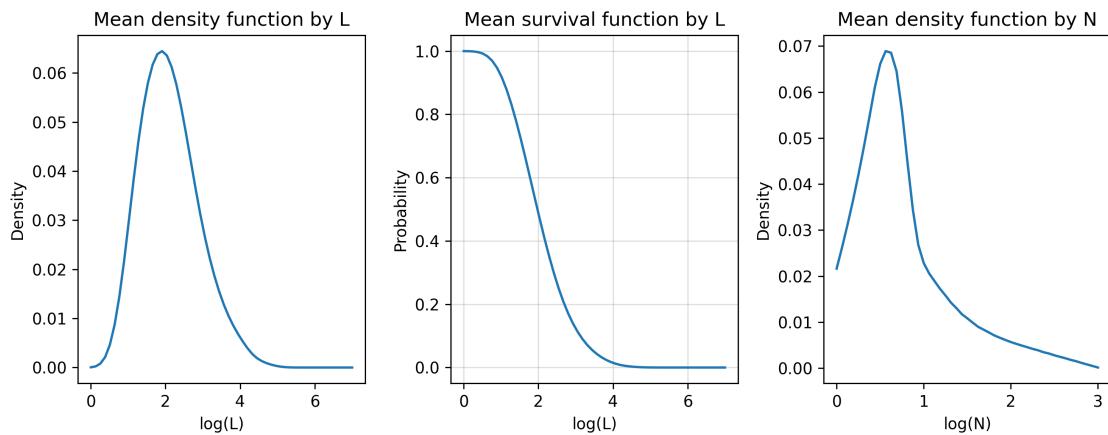


```
[5]: for s in [1, 2]:
    cluster(model=3, ks=[4], supermodel=s) # model 3
```

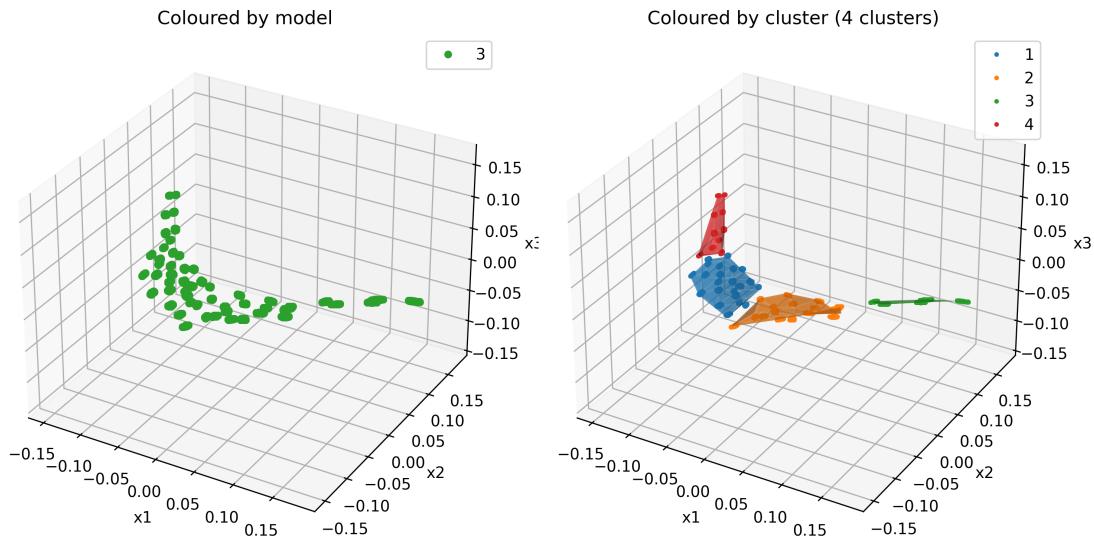
Model III

Mean density

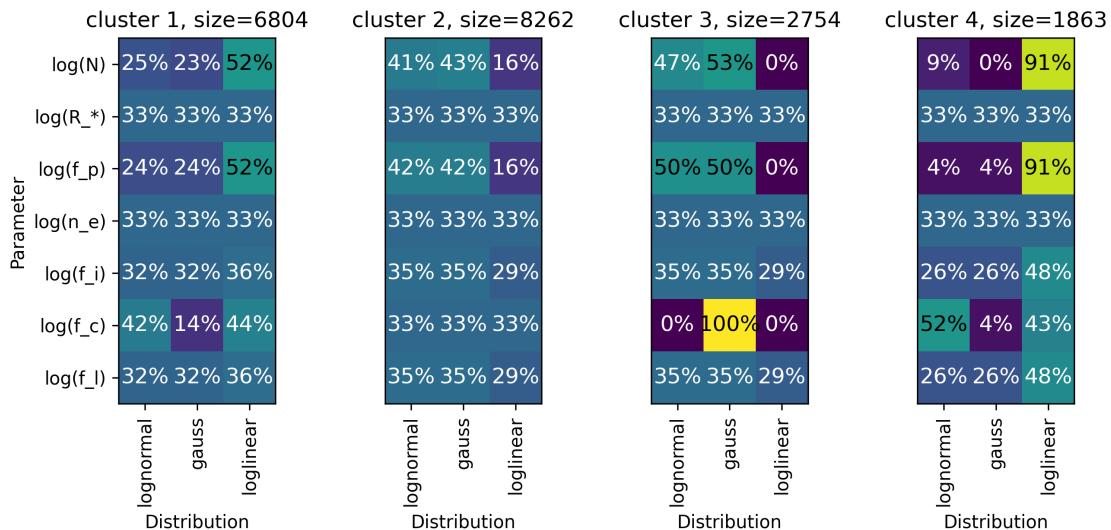




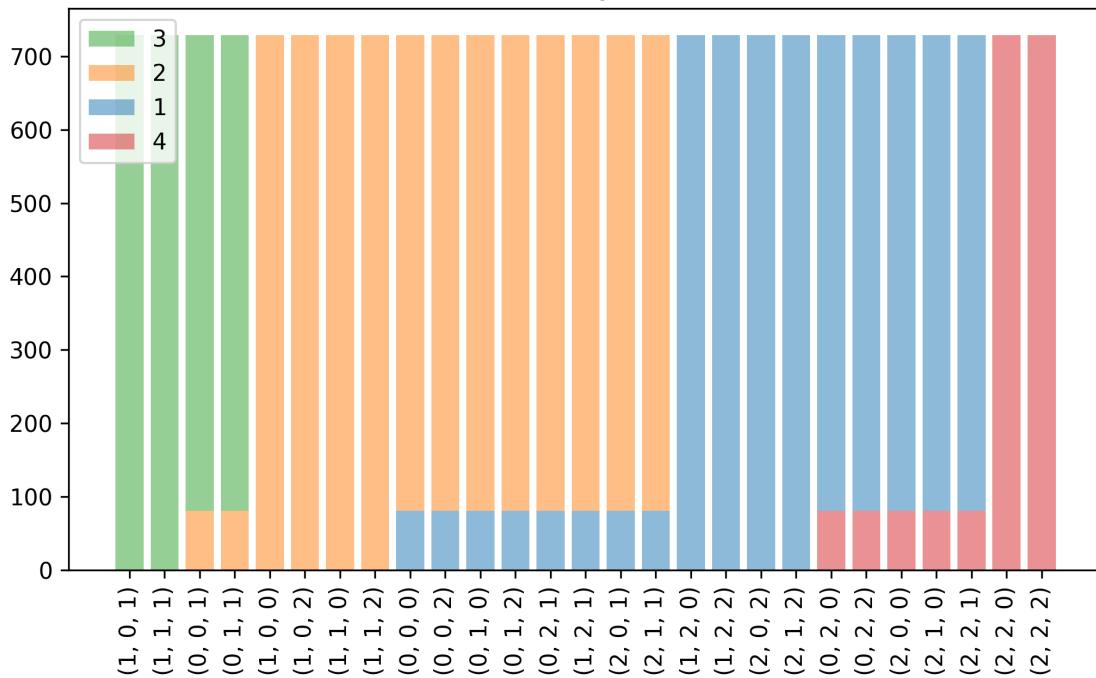
Model III after Principal component analysis (PCA)



Percent of submodels in model 3 distributed with particular distribution on particular parameter in each of 4 clusters

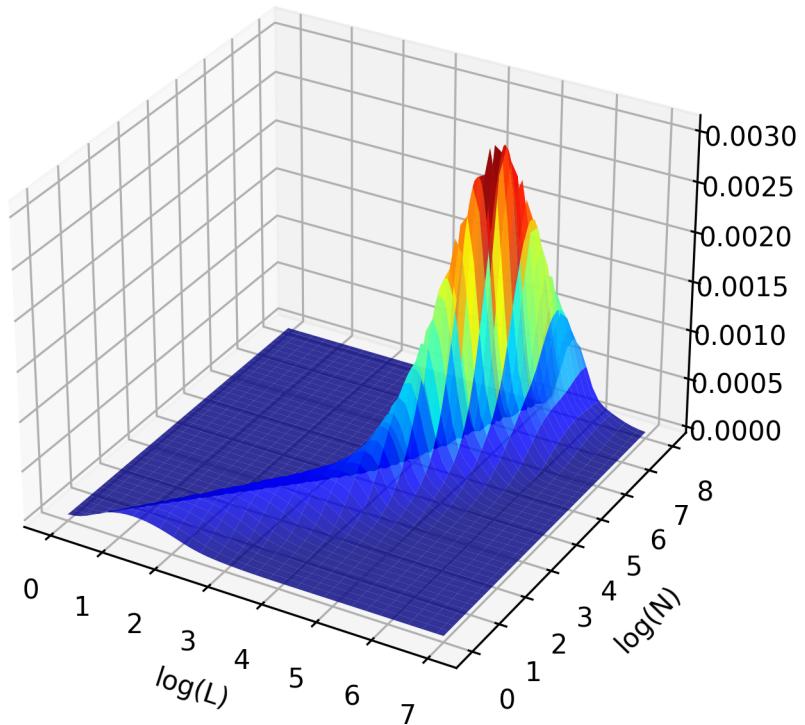


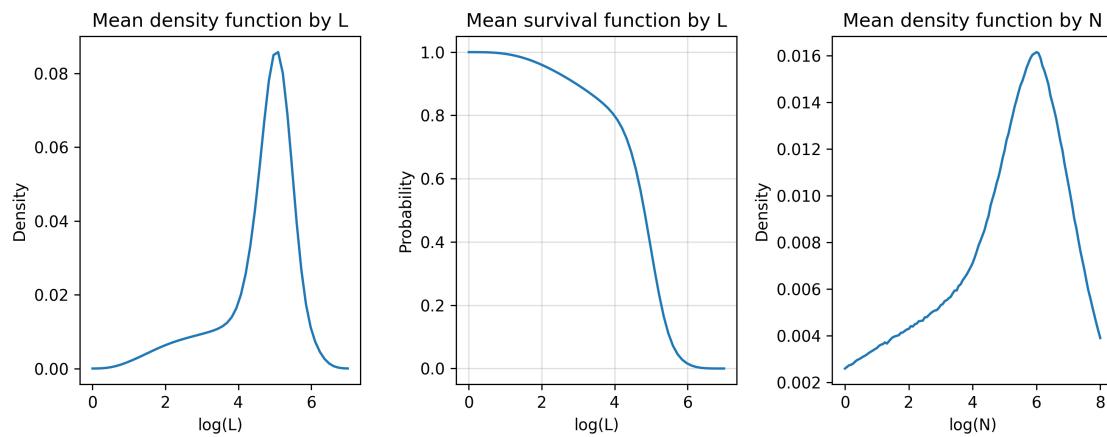
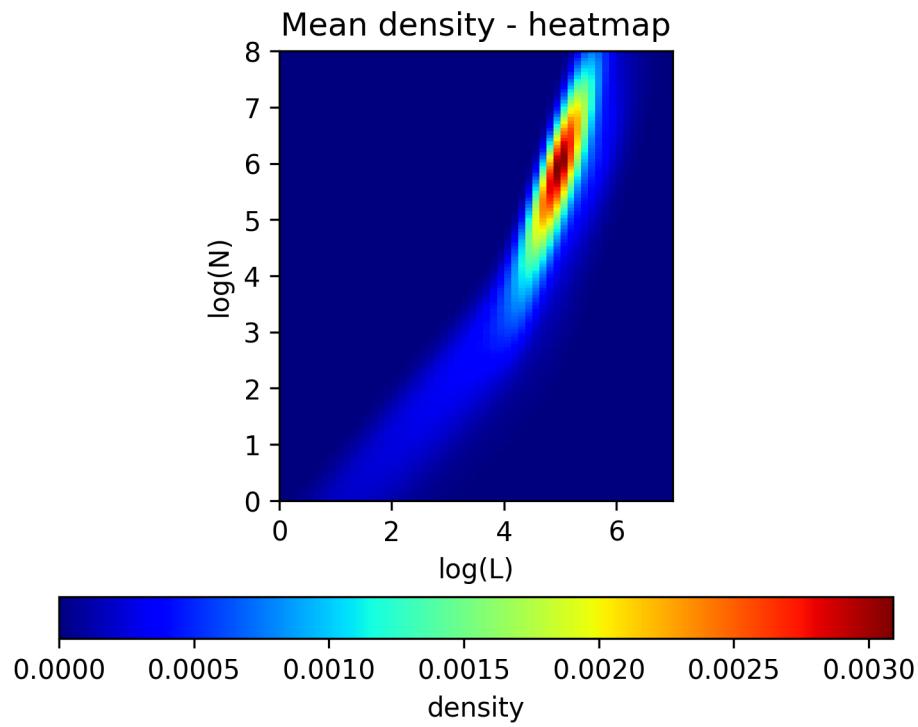
Percentage of distributions
 $(\log(N), \log(f_p), \log(f_c))$ by (lognormal, gauss, loglinear)
 coloured by cluster

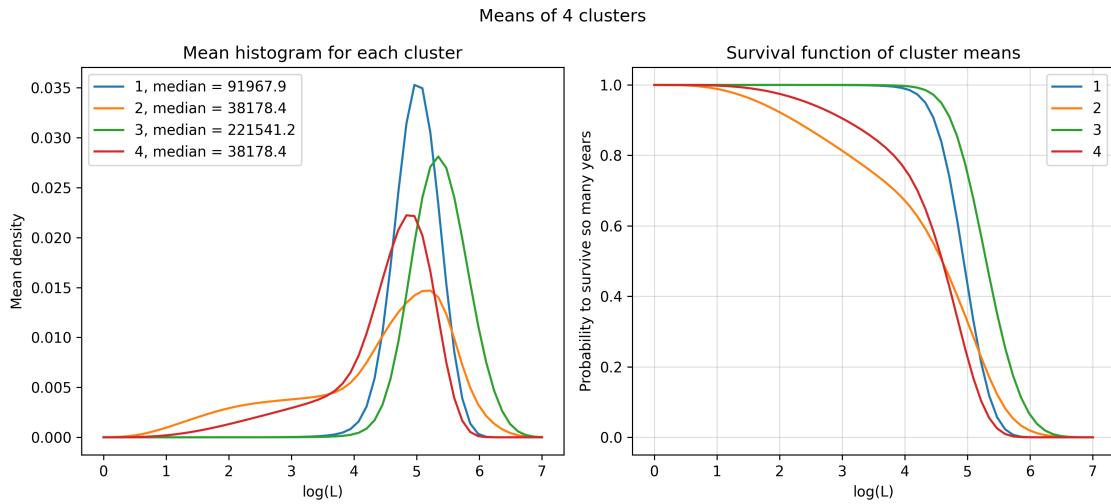


Model III

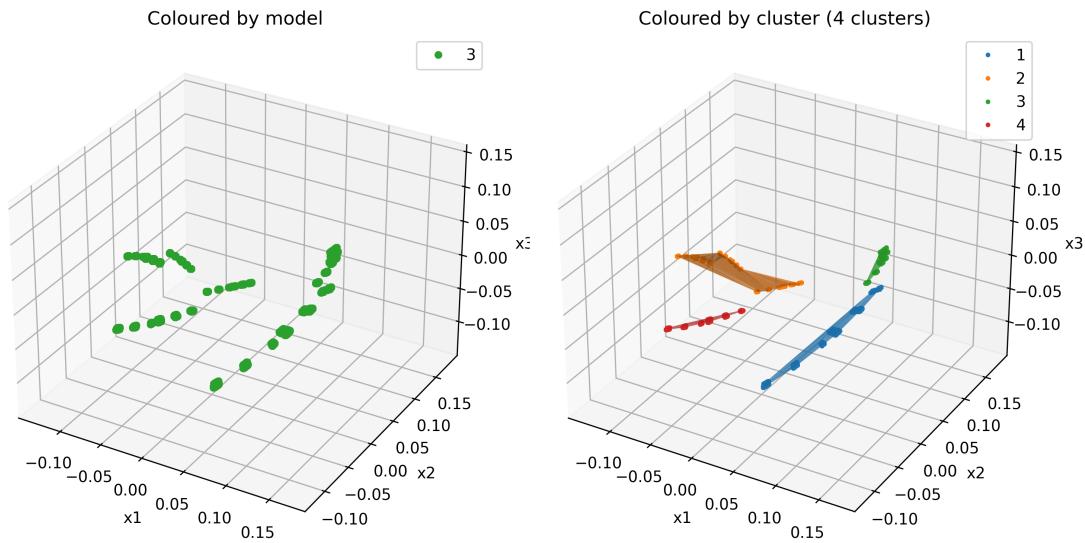
Mean density







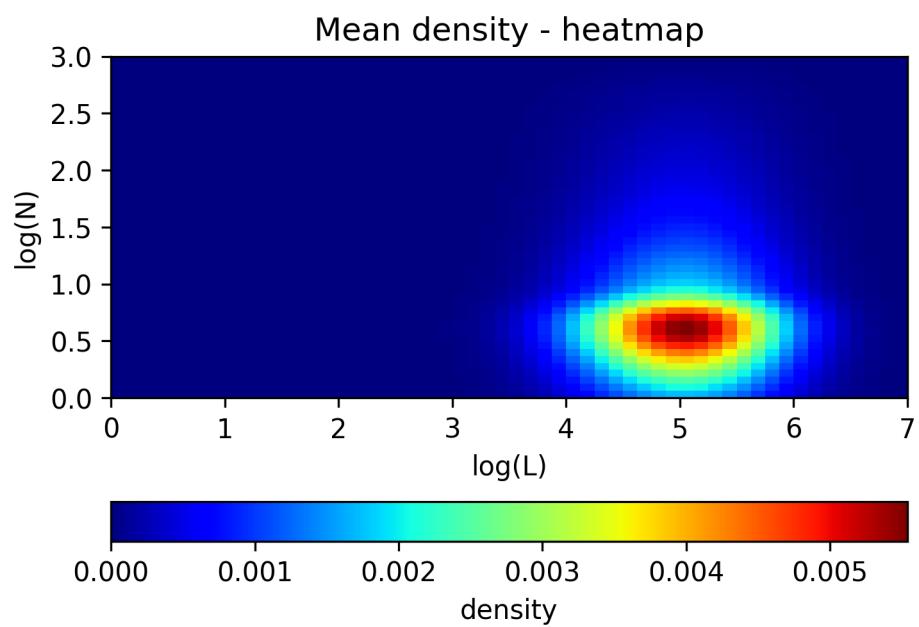
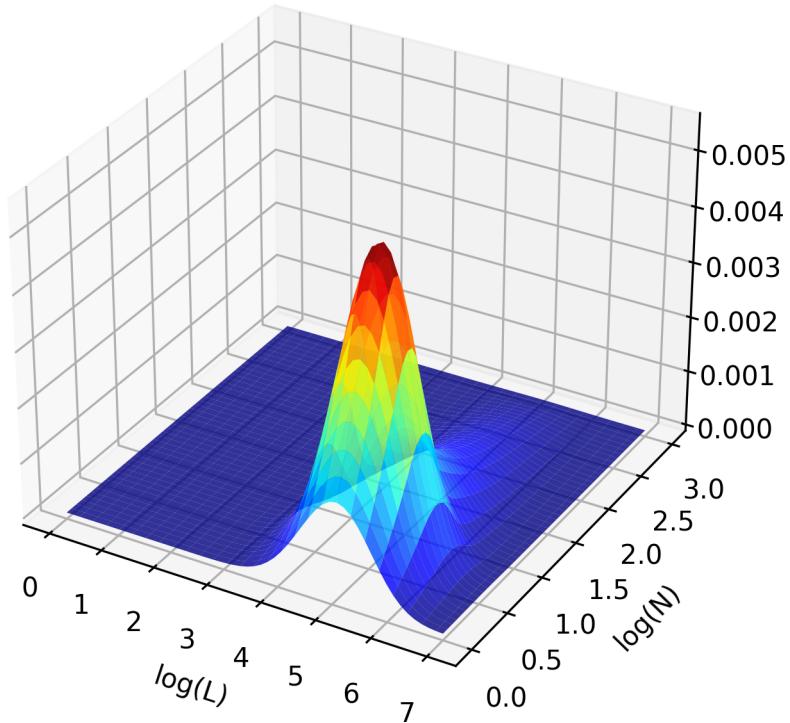
Model III after Principal component analysis (PCA)

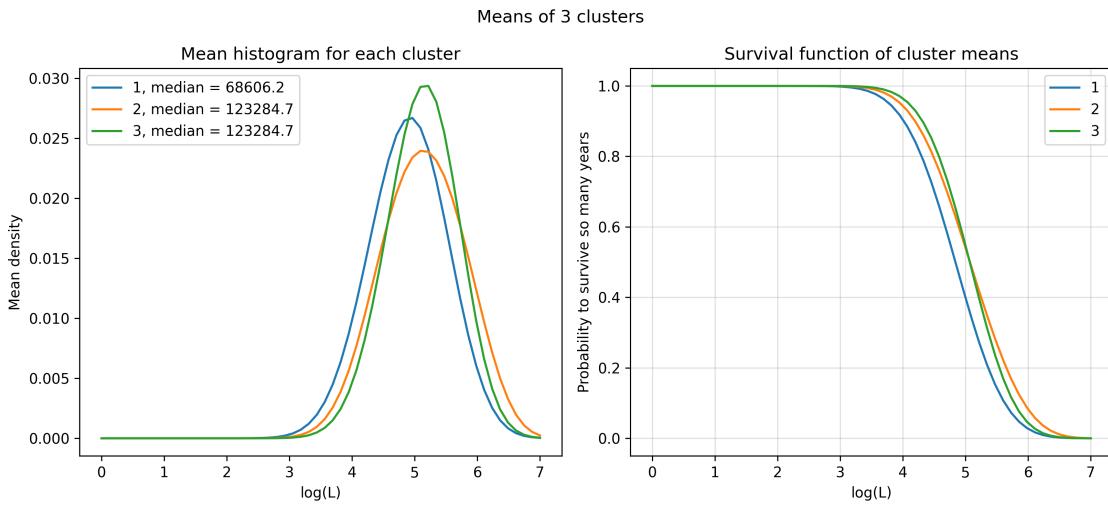
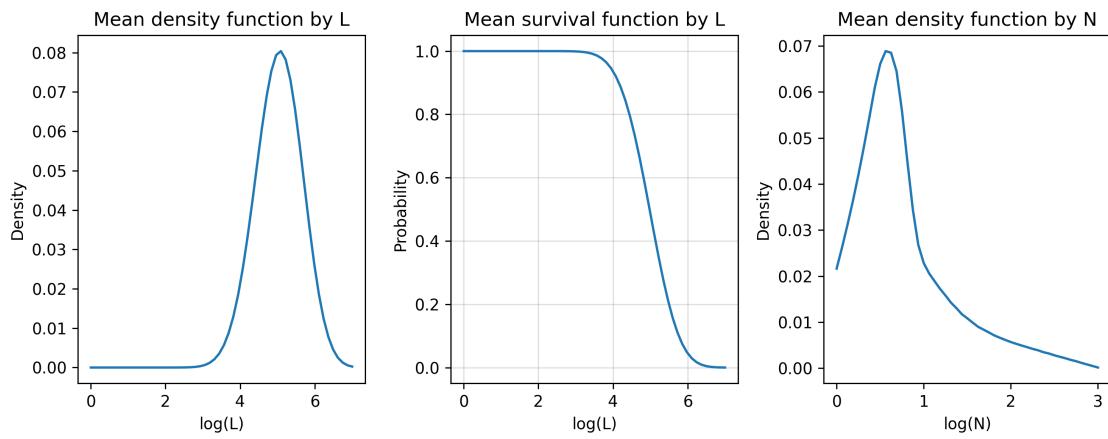


```
[6]: cluster(model=4, ks=[3]) # model 4
```

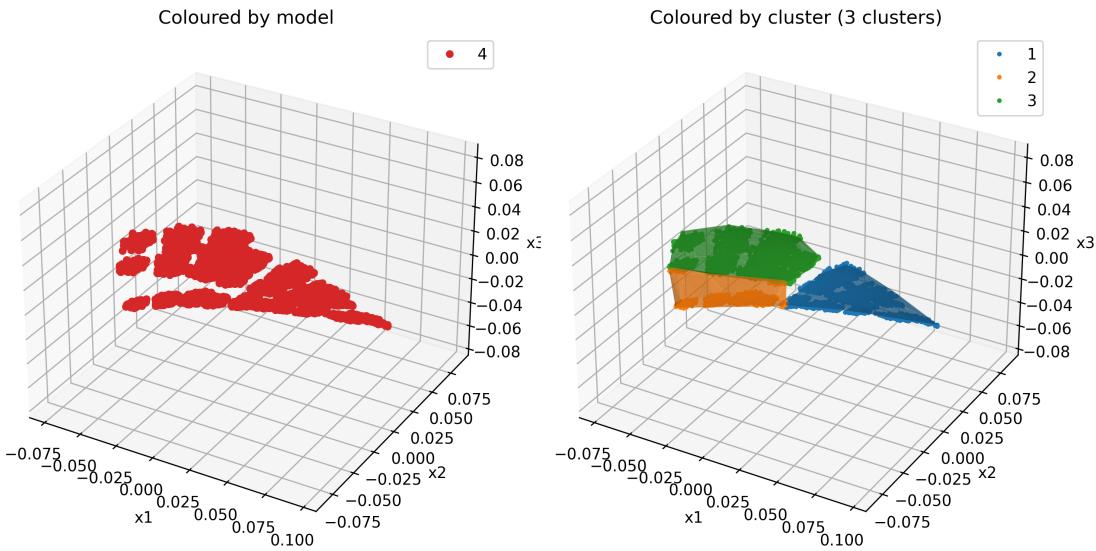
Model IV

Mean density

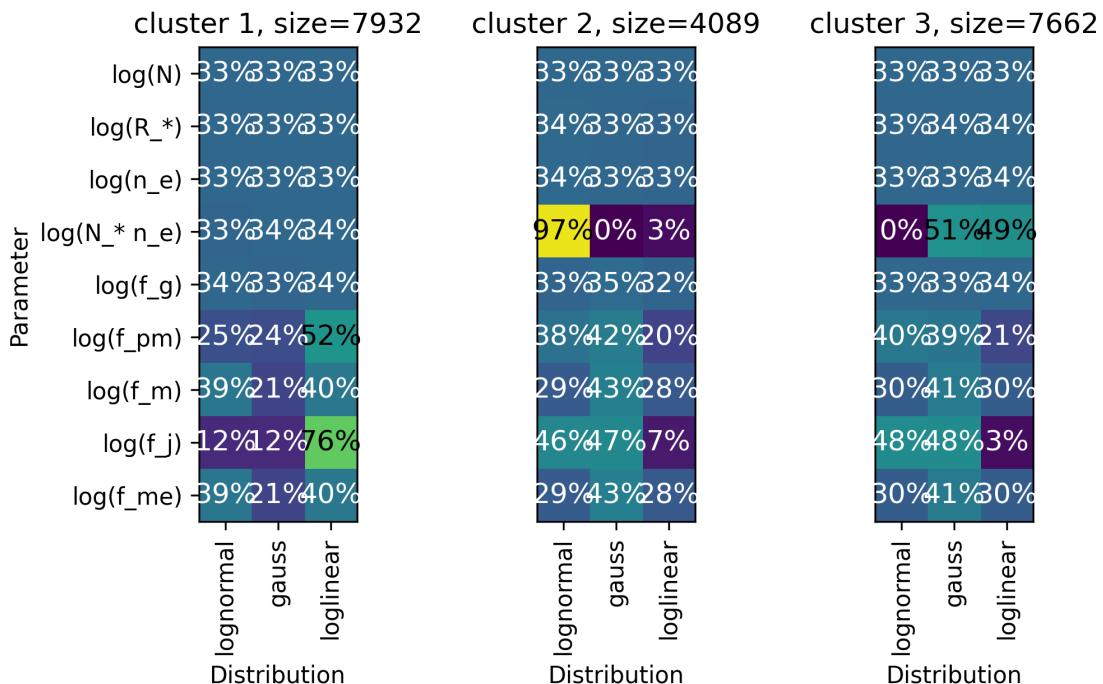


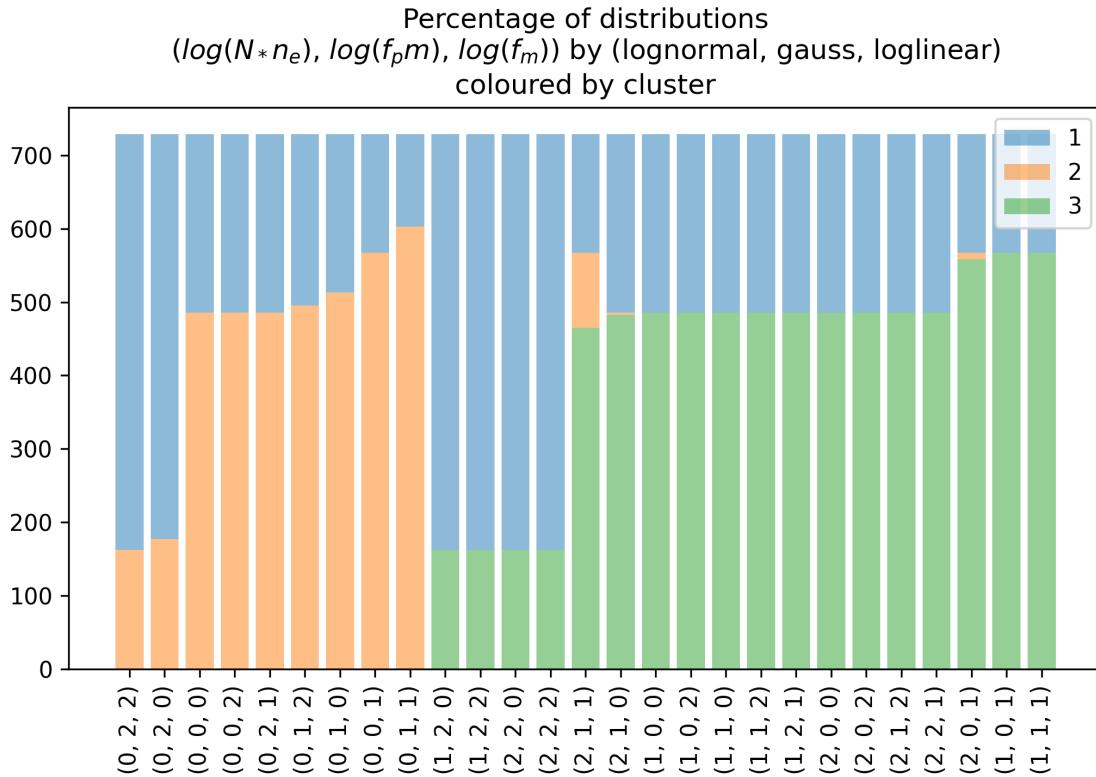


Model IV after Principal component analysis (PCA)



Percent of submodels in model 4 distributed with particular distribution on particular parameter in each of 3 clusters





```
[7]: def polyDeltaL(logL1):
    minP, meanP, maxP = -2, np.log10(1.8), np.log10(5)
    L1 = 10 ** logL1
    a1 = 10 ** (9 - np.array([minP, meanP, maxP]) - np.log10(np.pi * 4))
    a1 = [np.roots([1, 3 * L1, 3 * L1 ** 2 + a, L1 ** 3]) for a in a1]
    return [- min(a, key=lambda x: np.abs(x.imag)).real for a in a1]

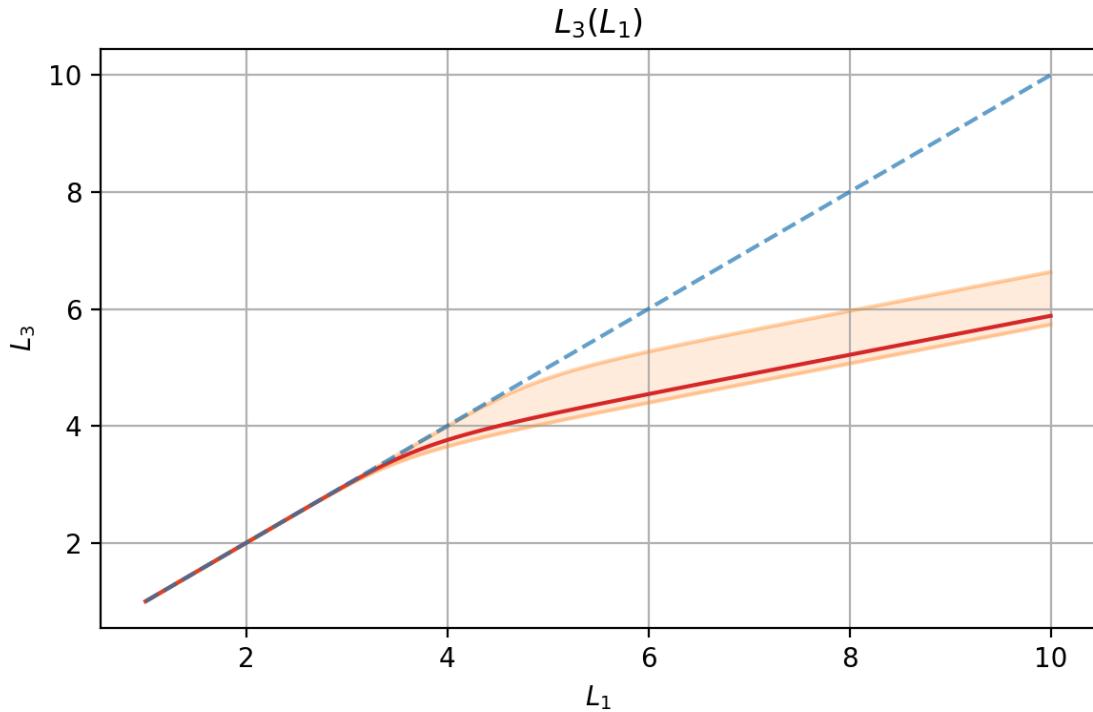
xi = np.linspace(1, 10, 100)
xi3 = np.array([xi, xi, xi]).T
h = np.array([polyDeltaL(x) for x in xi])
ca = [("tab:orange", 0.3), ("tab:red", 1), ("tab:orange", 0.3)]

for ylab, ydat in [("L_3", np.log10(np.abs(10 ** xi3 - h)))]:
    plt.figure(figsize=(6, 4), dpi=200, tight_layout=True)
    plt.fill_between(xi, ydat[:, 0], ydat[:, 2],
                     color=ca[0][0], alpha=ca[0][1] / 2)
    for i in range(3):
        plt.plot(xi, ydat[:, i], color=ca[i][0], alpha=ca[i][1])
    if "L_1" not in ylab:
        plt.plot(xi, xi, "--", alpha=0.7)
    plt.title(f"${ylab}({L_1})$")
```

```

plt.xlabel("$L_1$")
plt.ylabel(f"${ylab}$")
plt.grid()
plt.savefig(f"slice/model3-model1.png")
plt.show()

```

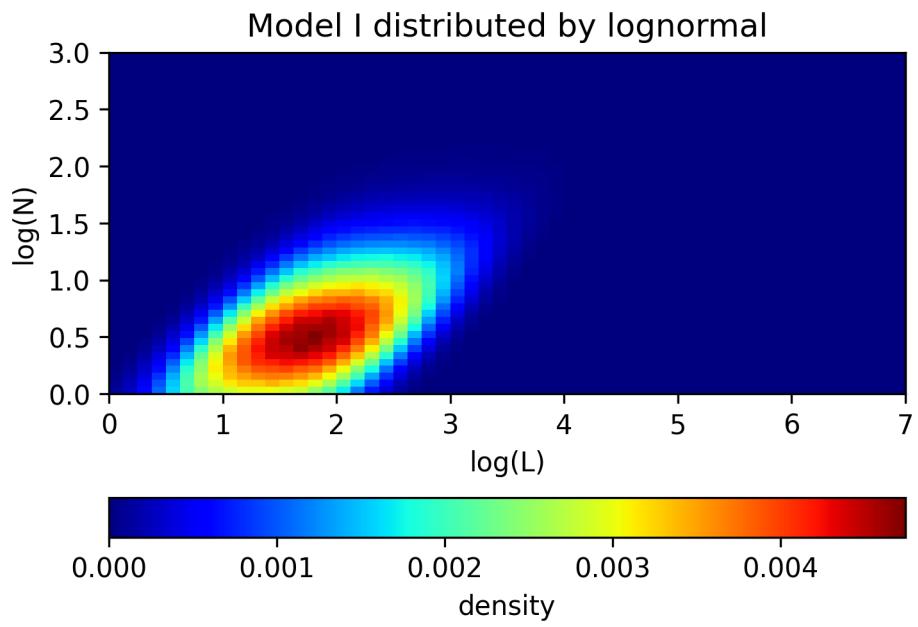
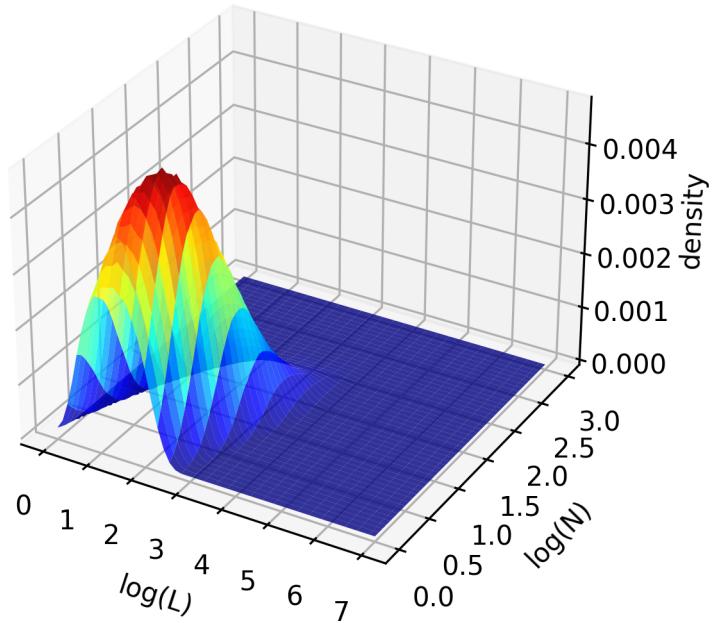


```

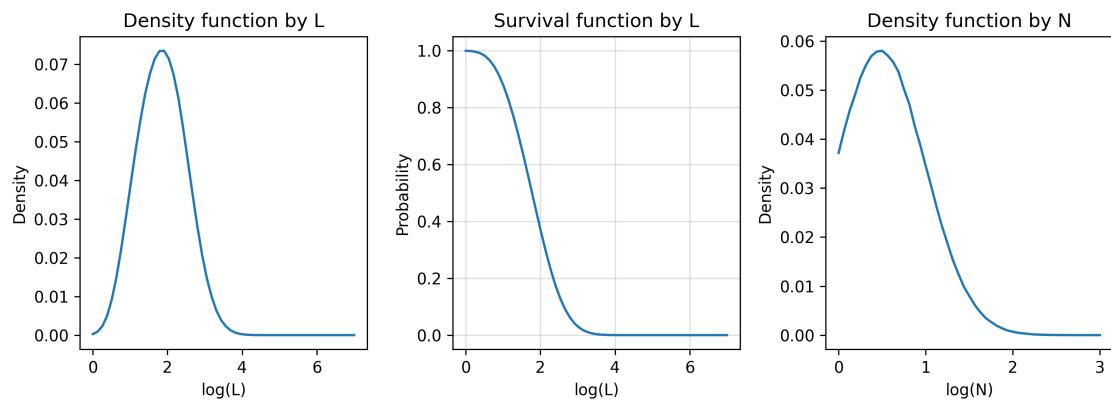
[8]: from plot3D_L import *
draw_histograms3D(model=1, distribution=tuple(0 for _ in range(6)), ↴
    ↪supermodel=1)
draw_histograms3D(model=2, distribution=tuple(0 for _ in range(2)), ↴
    ↪supermodel=1)
draw_histograms3D(model=3, distribution=tuple(1 for _ in range(6)), ↴
    ↪supermodel=1)
draw_histograms3D(model=4, distribution=tuple(0 for _ in range(8)), ↴
    ↪supermodel=1)
draw_histograms_N3D()
plt.show()

```

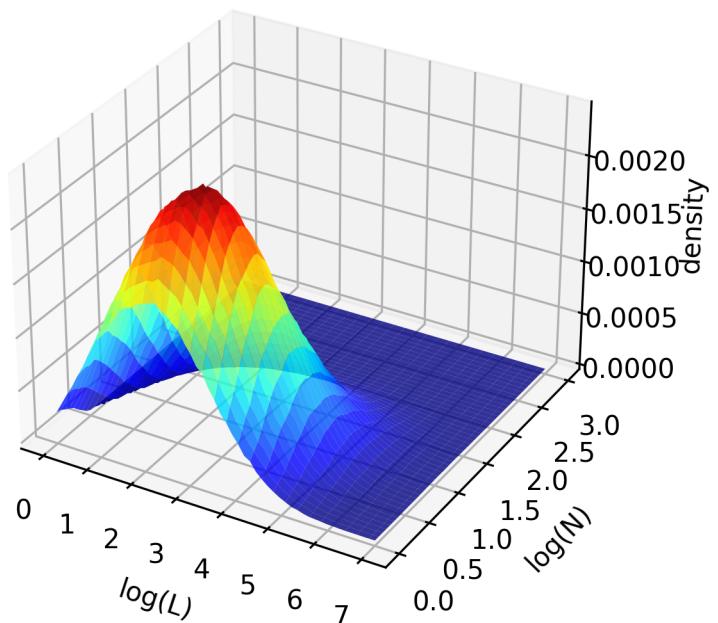
Model I distributed by lognormal

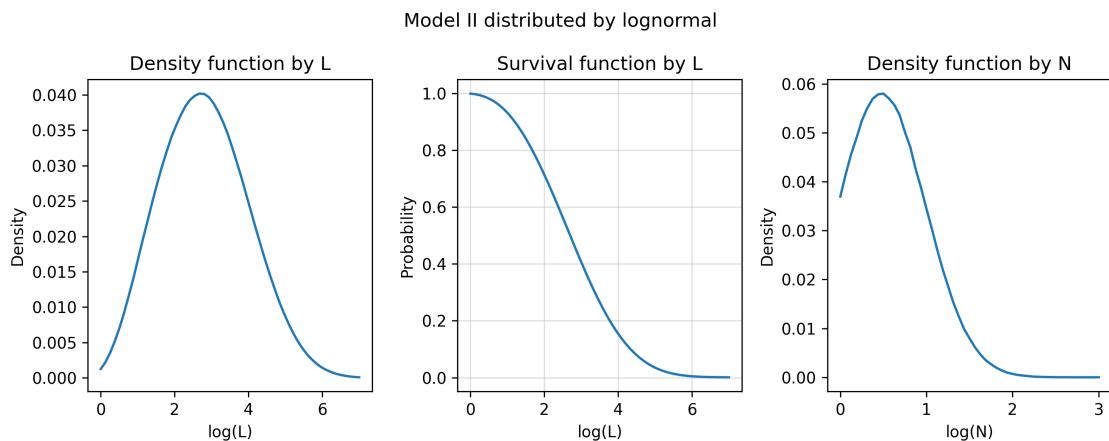
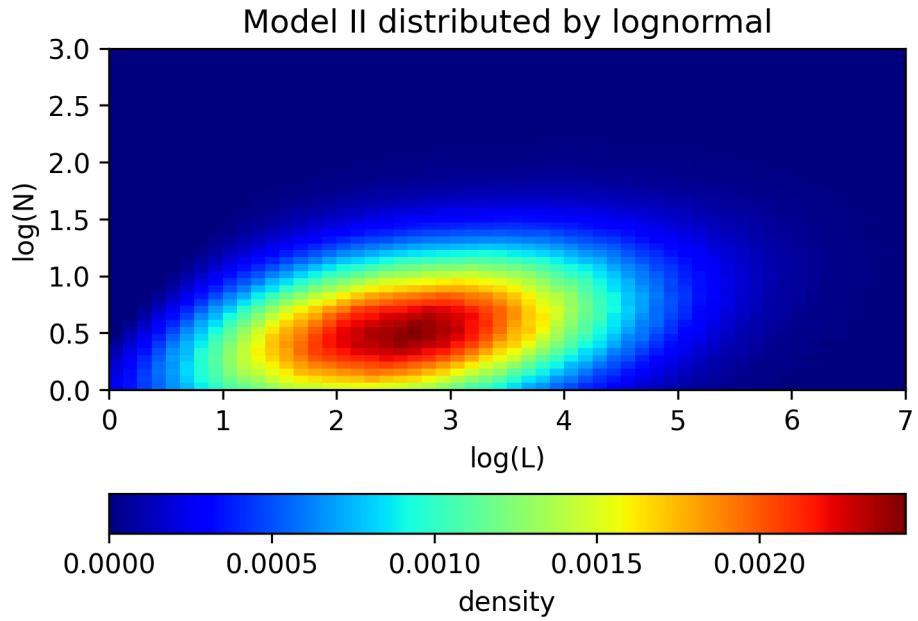


Model I distributed by lognormal

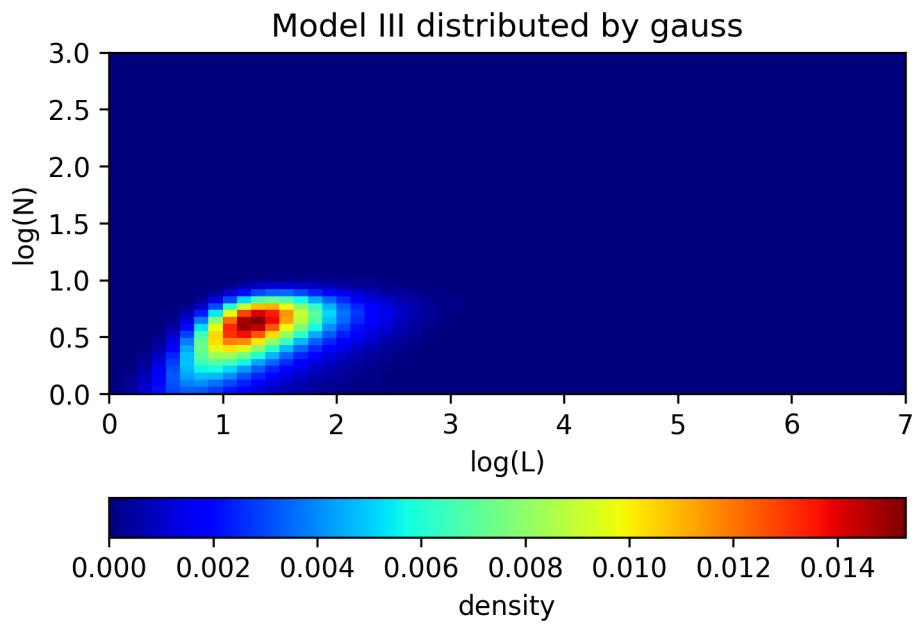
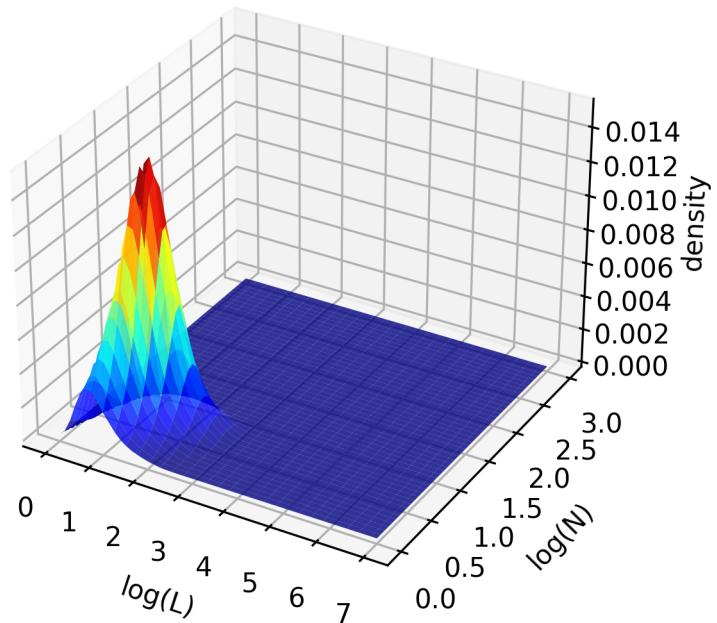


Model II distributed by lognormal

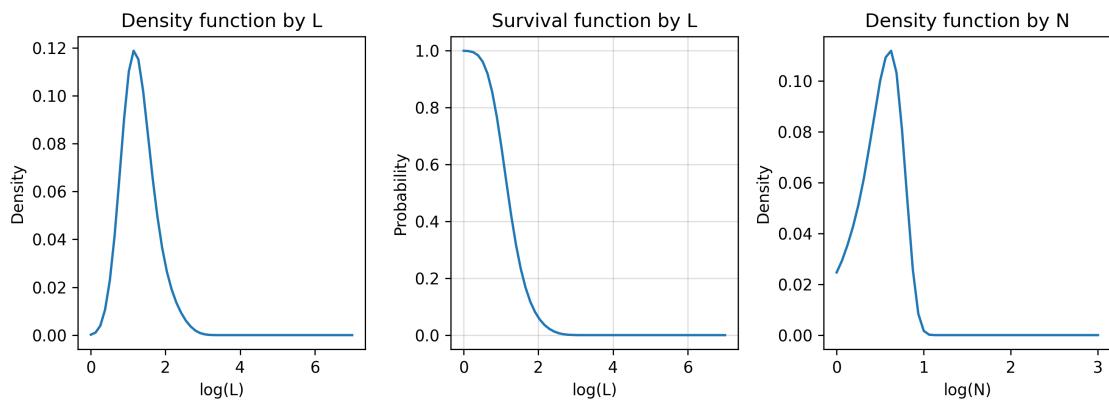




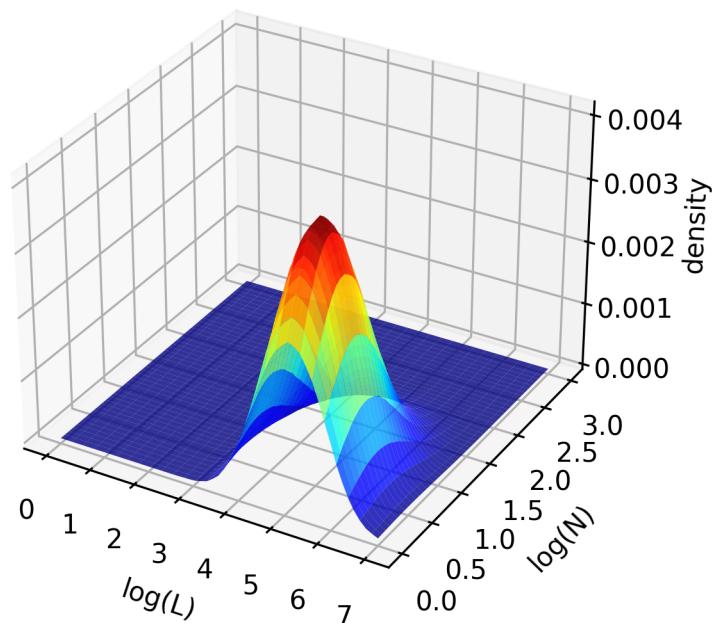
Model III distributed by gauss

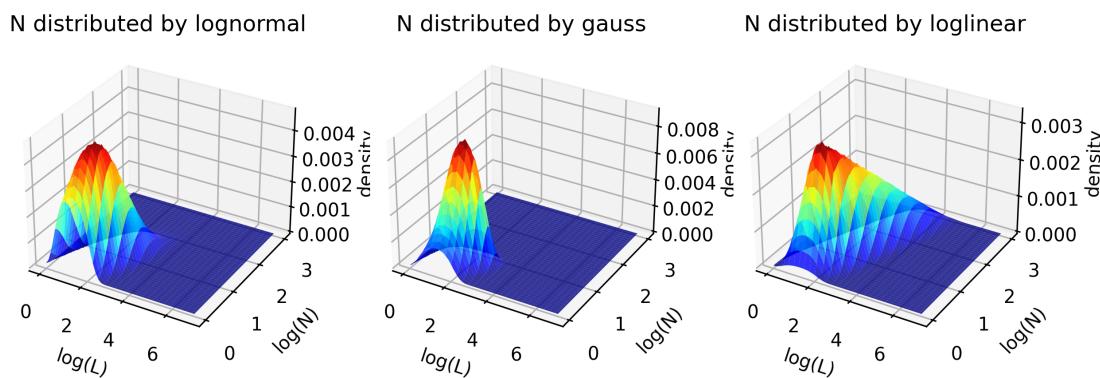
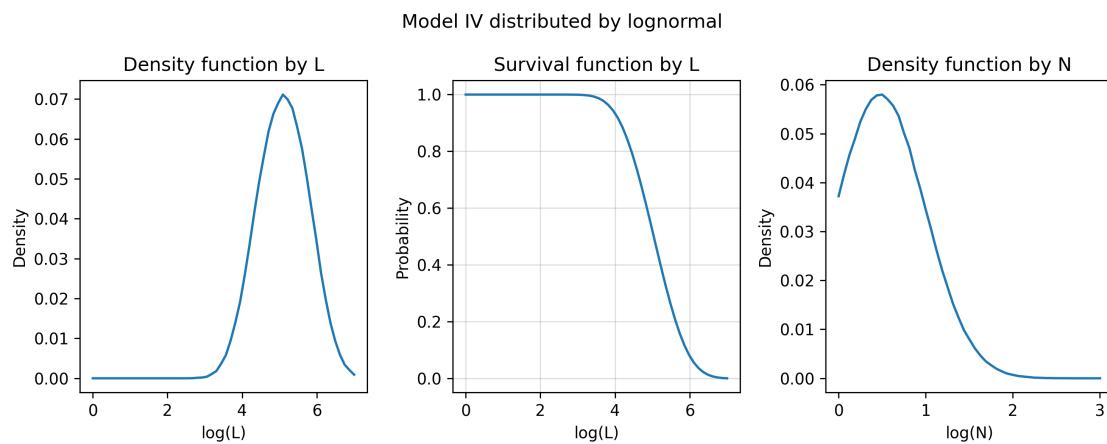
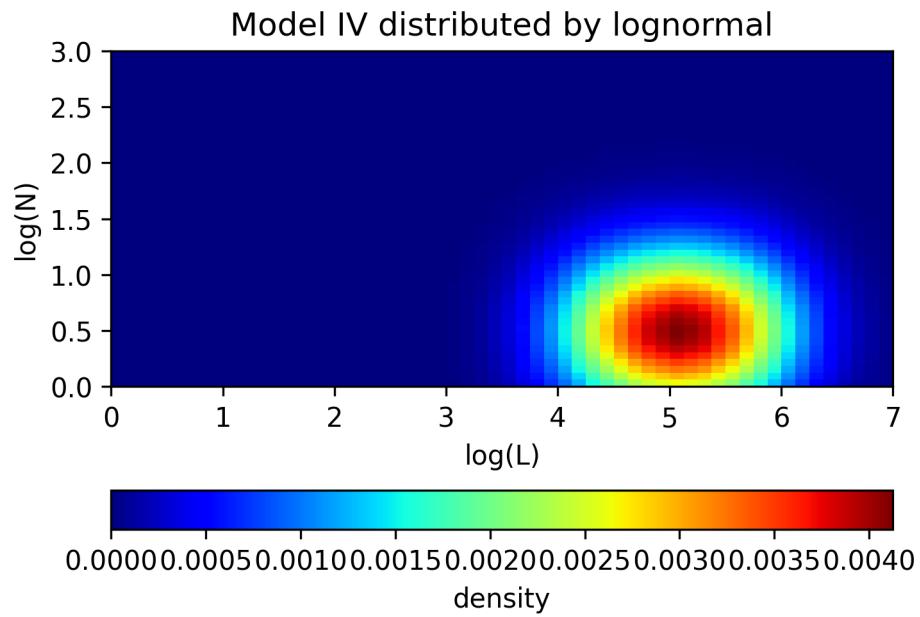


Model III distributed by gauss

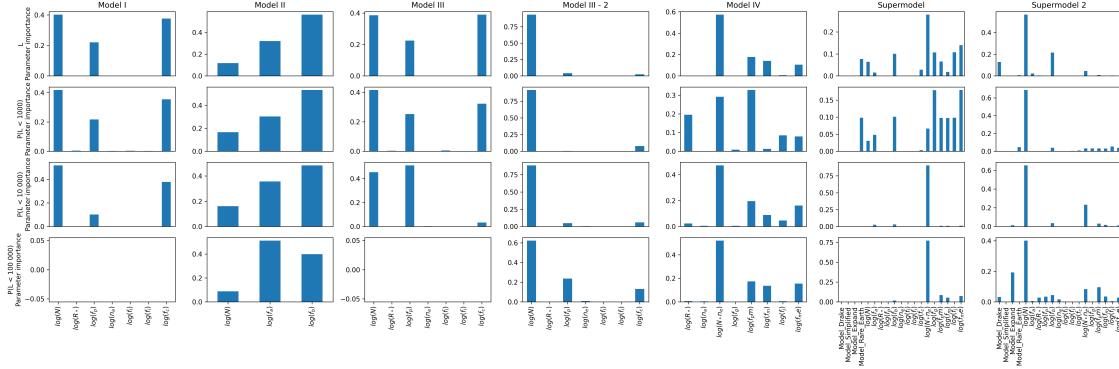


Model IV distributed by lognormal





```
[9]: from tabela_napake_p import modeli_napake
a = modeli_napake()
```



```
[10]: models = []
labels = []
results = {}
with open("ML_tabele.txt", "r") as f:
    lines = f.readlines()
    for i in range(len(lines) // 13 + 1):
        part_lines = lines[i * 13:(i+1) * 13 - 1]
        model = part_lines[-2][-1].replace("_", " ")
        label = part_lines[-1][-1]
        if model not in models:
            models.append(model)
        if label not in labels:
            labels.append(label)
        col_names = part_lines[0][-1].split(" ")[1:]
        row_names = [l[-1].split(" ")[0][:-13]
                    for l in part_lines[1:-2] if "error" not in l]
        part_lines = [[round(float(n), 4) for n in l[-1].split(" ")[1:]]
                      for l in part_lines[1:-2] if "error" not in l]
        results[(model, label)] = pd.DataFrame(part_lines,
                                                index=row_names,
                                                ↪columns=col_names)
```

```
[11]: """
pd.set_option('display.width', 82)
pd.set_option('display.max_columns', None)
for model in models:
    print("\n" + "="*60 + "\n")
    print(" "*5 + "-"*20 + " "*3 + model + " "*3 + "-"*20 + "\n")
```

```

a = pd.concat({k[1]: l for k, l in results.items() if model in k and "<" in
↪k[1]}, 1)
a.columns = a.columns.swaplevel()
a.name = label
a = a[sorted(a.columns, key=lambda x: (x[0], x[1].replace(" 0", "0")))]
print(a)
a.style.background_gradient(cmap="Blues")
# from IPython.display import display
# display(a.style.background_gradient(cmap="Oranges"))
"""
print("")

```

[12]:

```

"""
pd.set_option('display.width', 72)
pd.set_option('display.max_columns', None)

for label in labels:
    print("\n" + "="*60 + "\n")
    print(" "*5 + "-"*20 + " "*3 + label + " "*3 + "-"*20 + "\n")
    a = pd.concat({k[0].replace("Model ", ""): l for k, l in results.items() if
↪label in k}, 1)
    a.columns = a.columns.swaplevel()
    a.name = label
    a = a[sorted(a.columns)]
    if "100 " in label:
        pd.set_option('display.width', 62)
    print(a)
    a.style.background_gradient(cmap="Blues")
    # from IPython.display import display
    # display(a.style.background_gradient(cmap="Oranges"))
"""
print("")

```