Sequential sampling

The purpose of this tutorial is to provide an example of how our new sequential sampling algorithm can be used to generate random samples from the niche-neutral model.

1. Detailed example

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
```

1.1. Define parameter values

We begin by defining some parameter values for our example. Let the fundamental biodiversity per niche $\theta_k = 1.3$, migration parameter $m = 4 \times 10^{-3}$, and define four islands with areas given below.

In [2]:

```
theta_k = 1.3
m = 4e-3
areahV = [0.1, 1, 10, 50] # areas in sq-km
H = len(areahV)
```

As in the main text, density $\rho = 1700$ birds per sq-km, and therefore the number of individuals on each island is

In [3]:

```
rho = 1700
JhV = [ int(rho*area) for area in areahV ]
JhV
```

Out[3]:

```
[170, 1700, 17000, 85000]
```

Assume that the migration rate is the same on every island

In [4]:

```
mhV = [m]*H
```

To make things interesting, let us assume that the number niches varies between islands so that the large islands have more niches.

In [5]:

```
KhV = [8, 8, 9, 10]
```

Assume that every niche has the same fundamental biodiversity number

In [10]:

```
thetakV = [theta_k]*max(KhV)
```

Now create J, a matrix of the number of individuals in niche k on island h

In [9]:

```
J = [ [ 0 for h in range(H)] for k in range(max(KhV)) ]

for k in range(max(KhV)):

    for h in range(H):

        J[k][h] = JhV[h] // KhV[h] if k < KhV[h] else 0

J
```

Out[9]:

```
[[21, 212, 1888, 8500],
 [21, 212, 1888, 8500],
 [21, 212, 1888, 8500],
 [21, 212, 1888, 8500],
 [21, 212, 1888, 8500],
 [21, 212, 1888, 8500],
 [21, 212, 1888, 8500],
 [21, 212, 1888, 8500],
 [0, 0, 1888, 8500],
 [0, 0, 0, 8500]]
```

Each row in J corresponds to one of the 10 niches, and each entry within the row corresponds to one of the four islands.

1.2. The algorithm

The code below follows the algorithm given in the online appendix.

In [13]:

```
ancestors = list() # ancestors urns
community = list() # community urns

no_ancestors = [ 0 for k in range(max(KhV)) ] # keep track of the number of ance
stors drawn

for k in range(max(KhV)): # for each niche

    ancestors.append([]) # each niche has its own ancestors urn
    community.append([]) # community urns are also split between niches

    for h in range(H): # for each island

        community[k].append([ 0 for a_k in range(len(ancestors[k])) ]) # and eac
h island has its own community urn

        I = mhV[h] * (J[k][h]-1) / (1-mhV[h])

        for j in range(J[k][h]): # for each individual

            alpha1 = np.random.rand()

            if alpha1 <= I / (I+j):

                # drawn an immigrant

                alpha2 = np.random.rand()

                if alpha2 <= thetakV[k] / (thetakV[k] + no_ancestors[k]):

                    # immigrant individual is a new species
                    ancestors[k].append(1)
                    community[k][h].append(1)

                else:

                    # species we've seen before
                    prob_i = [ ai / no_ancestors[k] for ai in ancestors[k] ]
                    i_star = np.random.choice( range(len(prob_i)), 1, p = prob_i
)[0]
                    ancestors[k][i_star] += 1
                    community[k][h][i_star] += 1

                no_ancestors[k] += 1 # increment ancestors counter

            else:

                # drawn a local individual
                prob_i = [ ni / j for ni in community[k][h] ]
                i_star = np.random.choice( range(len(prob_i)), 1, p = prob_i )[0
]
                community[k][h][i_star] += 1
```

The final state of the ancestors urns (one urn for each niche):

In [19]:

```
ancestors
```

Out[19]:

```
[[46, 74, 50, 26, 4, 7, 17, 1, 1],
 [76, 81, 38, 4, 6, 21, 3, 4, 1],
 [68, 53, 35, 61, 8, 2, 1],
 [93, 19, 73, 54, 5, 1, 1],
 [80, 119, 1, 8, 8, 4, 2, 15, 1],
 [173, 41, 2, 12, 1, 1, 1, 2, 1],
 [19, 105, 67, 38, 2, 2, 1, 2, 1],
 [46, 2, 73, 96, 10, 14, 1, 5, 1, 2, 4, 2],
 [85, 150, 9, 3, 1, 2, 1],
 [125, 21, 8, 7, 5, 1, 3, 1]]
```

The final state of the community urns:

In [14]:

```
community
```

Out[14]:

```
[[[21],
  [63, 142, 2, 4, 1],
  [353, 967, 404, 0, 0, 81, 83],
  [2041, 2471, 1892, 1070, 44, 316, 654, 11, 1]],
 [[21],
  [1, 205, 6],
  [698, 857, 151, 113, 69],
  [2411, 2845, 825, 422, 154, 866, 952, 16, 9]],
 [[19, 2],
  [0, 28, 183, 1],
  [677, 638, 43, 529, 1],
  [2400, 2135, 2273, 1473, 199, 19, 1]],
 [[21], [9, 203], [1114, 68, 669, 37], [3013, 717, 2737, 1916, 115,
1, 1]],
 [[21],
  [27, 185],
  [498, 1046, 20, 85, 187, 49, 3],
  [3355, 3683, 0, 157, 212, 55, 8, 1000, 30]],
 [[21],
  [210, 2],
  [1648, 142, 19, 76, 3],
  [5767, 1824, 302, 464, 0, 90, 31, 21, 1]],
 [[21],
  [0, 208, 4],
  [6, 365, 859, 656, 1, 1],
  [771, 3530, 2642, 937, 200, 18, 154, 247, 1]],
 [[16, 5],
  [0, 0, 195, 17],
  [523, 0, 335, 959, 42, 29],
  [682, 3, 3488, 2870, 316, 468, 163, 150, 256, 19, 78, 7]],
 [[], [], [1319, 557, 9, 3], [1879, 5620, 188, 795, 1, 14, 3]],
 [[], [], [], [5688, 1615, 316, 307, 171, 235, 156, 12]]]
```

Each row of `community` corressponds to a niche, and each entry corresponds to an island. For example, these are the community urns for the four islands for the first niche:

In [20]:

```
community[0]
```

Out[20]:

```
[[21],
 [63, 142, 2, 4, 1],
 [353, 967, 404, 0, 0, 81, 83],
 [2041, 2471, 1892, 1070, 44, 316, 654, 11, 1]]
```

Each entry indicates the number of individuals on that island of that species. For example:

In [22]:

```
community[0][2]
```

Out[22]:

```
[353, 967, 404, 0, 0, 81, 83]
```

... says that, in the first niche on the third island, there are 353 individuals of species 1, 967 of species 2, 404 of species 3, 0 of species 4 and 5, etc.

Get the species richness on each island:

In [23]:

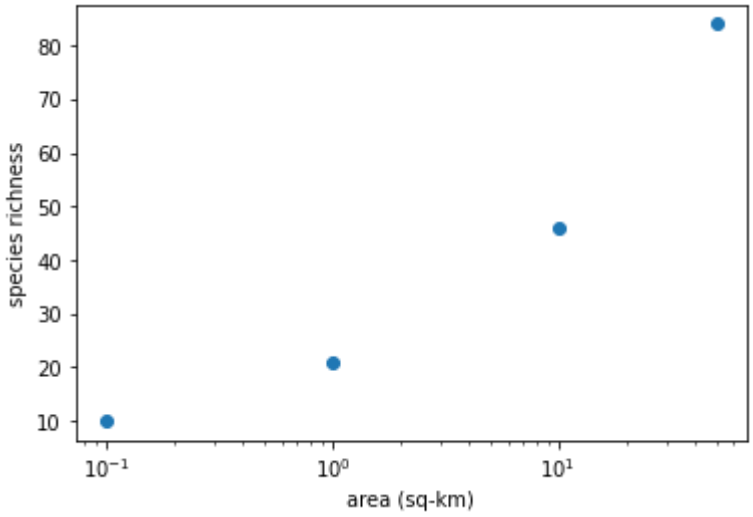
```
richnessV = [ sum(sum( 1 if i > 0 else 0 for i in ls ) for ls in v ) for v in zip(*community) ]
richnessV
```

Out[23]:

```
[10, 21, 46, 84]
```

In [18]:

```
plt.scatter(areahV, richnessV)
plt.xlabel('area (sq-km)')
plt.ylabel('species richness')
plt.xscale('log')
plt.show()
```



2. Use function from code repository

The sequential sampling algorithm above can also be found in `/functions/my_functions.py` : draw `sample_species_generator_general()`

In [24]:

```
import sys
sys.path.insert(0,'../functions')

from my_functions import draw_sample_species_generator_general
```

In [26]:

```
# draw_sample_species_generator_general? # to see documentation
```

In [27]:

```
ancestors2, community2 = draw_sample_species_generator_general(thetakV, mhV, J)
```

In [29]:

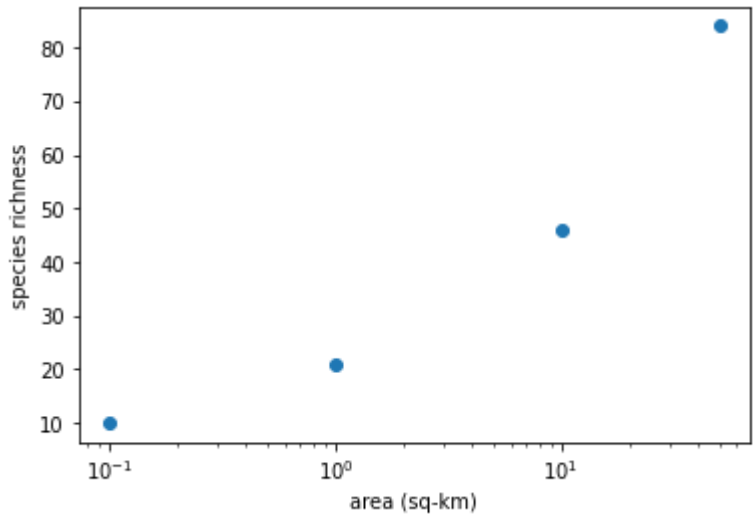
```
richness2V = [ sum(sum( 1 if i > 0 else 0 for i in ls ) for ls in v ) for v in z
ip(*community2) ]
richness2V
```

Out[29]:

```
[8, 24, 48, 75]
```

In [30]:

```
plt.scatter(areahV, richnessV)
plt.xlabel('area (sq-km)')
plt.ylabel('species richness')
plt.xscale('log')
plt.show()
```



In [ ]:

```

```