To My Dearest Tong

Xun

0.

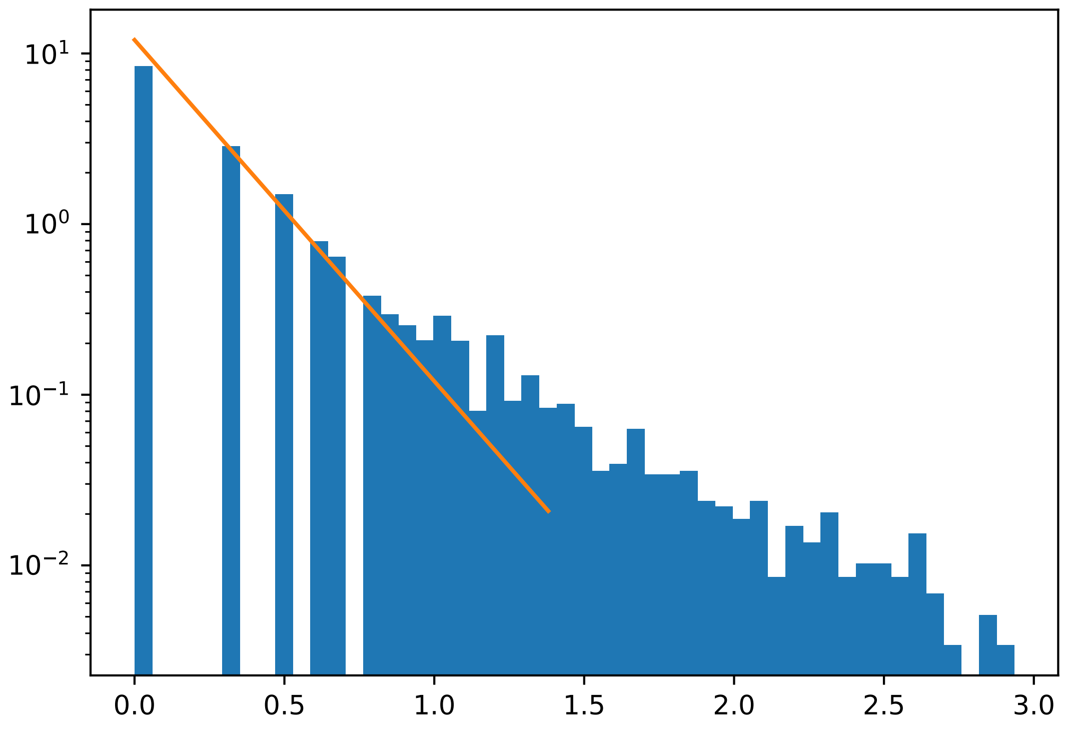
Personally, it is more important to generate dense data points in a small range of values, instead of sparse data points (10^3 is this case) in a large range of values (10^5 in this case), because it is easier to obtain trend of change intuitively when using denser points So, I switch them in codes below, namely, 10^5 trails and 10^3 replications. Plus, for three parts of codes below, the first part needs about $10sec.$, the second needs $90sec.$ and the third needs $150sec.$ on my laptop. It is time-consuming and occupies a lot of CPU and RAM resources. So be aware of your battery life when not charging.

1.

From the problem, the experiment should be like this: Starting with one cell, let it and its offsprings replicate. However, for every generation, not all cells but one chosen cell can replicate, where there will a chance for mutation to happen. When mutation happens, from one wild type, we will get one wild type and one mutant. In other cases, we get two same copies of the parental cell. In addition, when we have done $N = 10^3$ replications, stop the experiment. Plus, repeat the experiment for $M = 10^3$ times to get 10^3 data points.

The full-annotated code is as follows,

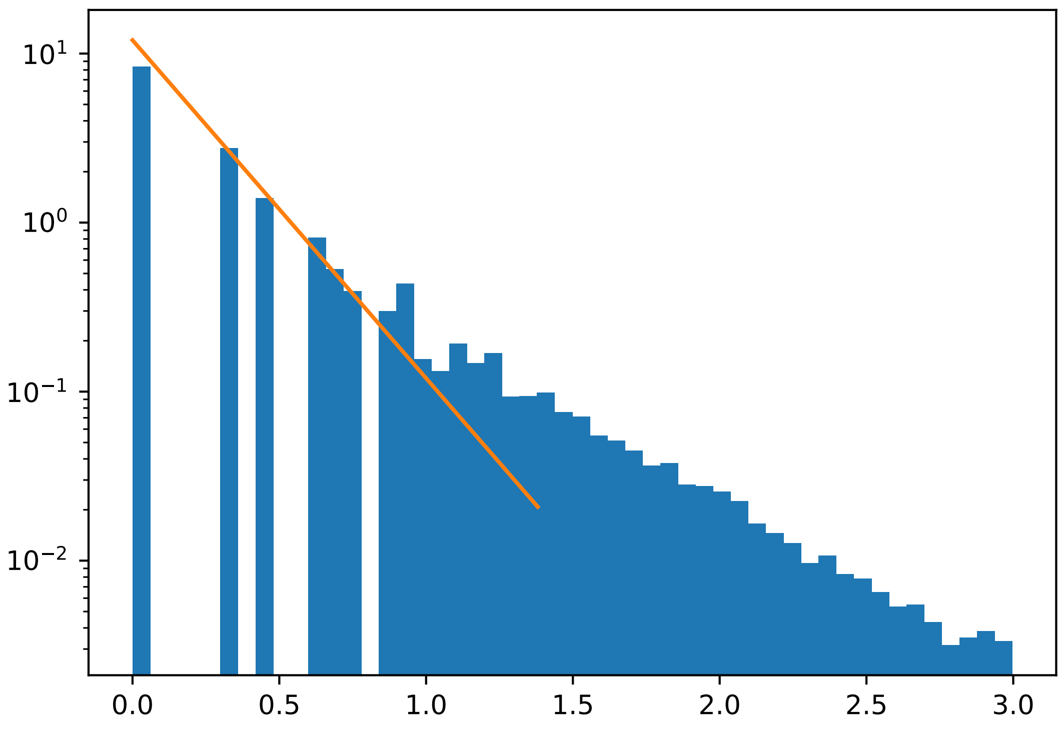
```
import random
import matplotlib.pyplot as plt
from math import log10
wild = 1 # the ancestor cell, and the number of wild type
mutant = 0 # the number of mutation
data = [] # to store 1000 data points
N = 10 ** 3 # number of maximum replications
M = 10 ** 4 # number of trials
# this code is not efficienct enough, so use smaller number
for trial in range(M): # repeat trials
    rep_mut = random.randint(0, N - 1)
    # randomly select a certain replication for mutation happens
    for rep in range(N): # replication for-loop
        who_rep = random.randint(1, wild + mutant)
        # choose one randomly to replicate
        if who_rep <= wild and rep != rep_mut:
            # wild replicates without mutation
            wild += 1
        else:
            # wild replicates with mutation
            # or mutant replicates
            mutant += 1
    data.append(log10(mutant)) # store the number of mutants
    wild = 1
    mutant = 0
    # reset the numbers
# set matplotlib.pyplot fig canvas
p1 = plt.hist(data, bins = 50, density = True, log = True)
# use matplotlib.pyplot to draw histogram
# with data, and 100 rectangles in the graph,
# and normalized it to 1 to show the frequency
# and convert Y-axis to log form
X = list(x for x in range(1, 25))
Y = [12 / x ** 2 for x in X]
# I do not know what you derived in class, so the log-log line
# of power law distribution is not correct. Do it youself !!
p2 = plt.plot([log10(x) for x in X], Y)
plt.show()
```



More efficient code:

```
import random
import matplotlib.pyplot as plt
from math import log10
wild = 1 # the ancestor cell, and the number of wild type
data = [] # to store 1000 data points
N = 10 ** 3 # number of maximum replications
M = 10 ** 5 # number of trials
for trial in range(M): # repeat trials
    rep_mut = random.randint(0, N - 1)
    # randomly select a certain replication for mutation happens
    wild += rep_mut
    # there will be no mutant until mutation happens,
    # so for replication 0 ~ (rep_mut - 1),
    # only need to add one more wild type every time
    # and at rep_mut-th replication, there are rep_mut + 1 wilds
    # and 1 mutant
    for rep in range(rep_mut + 1, N):
        wild += 1 if random.randint(1, rep + 1) <= wild else 0
```

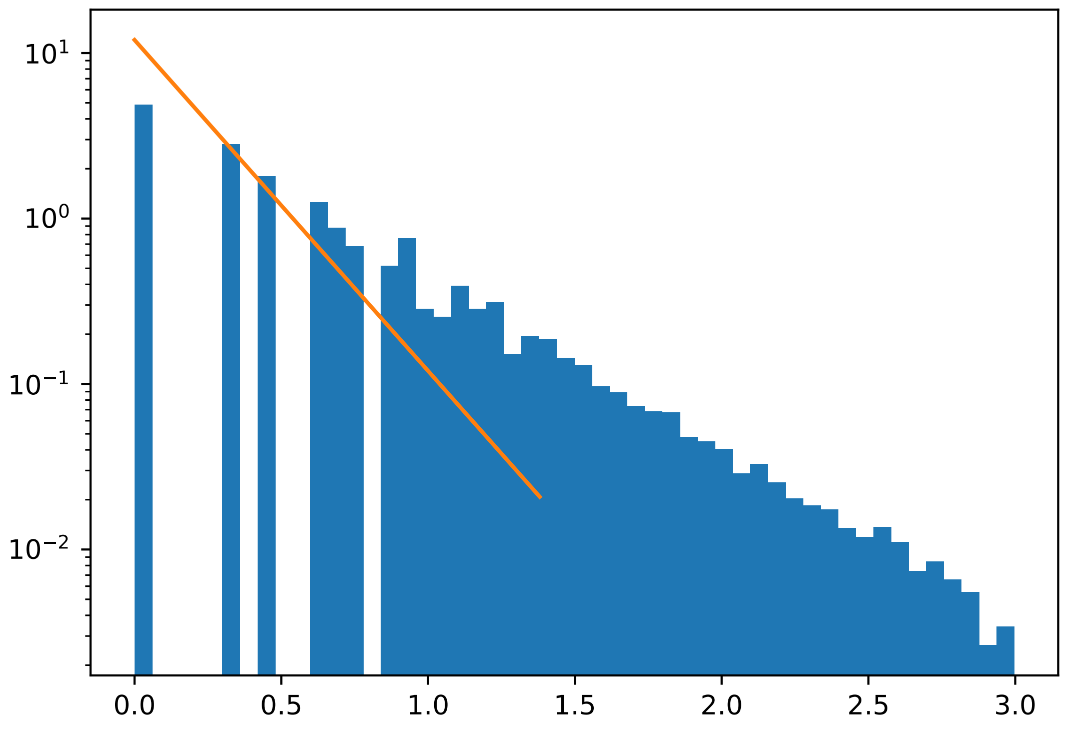
```
          # at rep-th replication, there are rep + 1 individuals
          # if the random number is smaller than the population of
          # wilds, then plus one to wilds
     data.append(log10(N + 1 - wild))
     # because the total population is N + 1 at last, so we only need
     # to calculate the number of wild individuals
     wild = 1
     # reset
plt.cla() # reset figure canvas
p1 = plt.hist(data, bins = 50, density = True, log = True)
X = list(x for x in range(1, 25))
Y = [12 / x ** 2 for x in X]
p2 = plt.plot([log10(x) for x in X], Y)
plt.show()
```



2.

In codes below, I generate a list called `rep_muts` to store when the replications will happen. That is, for each N replications, generate a random number between. If the number is smaller than threshold 10^{-3}, we add it into the list, which means this i-th replication will give out a mutant. When doing the replication for-loop, only if the replication is not in `rep_muts` (means that it is withou mutation), and the replication individual is wildtype, we add a new wildtype to it.

```
import random
import matplotlib.pyplot as plt
from math import log10
wild, data = 1, []
N, M = 10 ** 3, 10 ** 5
for trial in range(M):
    rep_muts = [i for i in range(N) if random.random() < 10 ** (-3)]
    for rep in range(N):
        wild += 1 if rep not in rep_muts \
            and random.randint(1, rep + 1) <= wild else 0
    if wild != N + 1:
        data.append(log10(N + 1 - wild))
    wild = 1
plt.cla()
p1 = plt.hist(data, bins = 50, density = True, log = True)
X = list(x for x in range(1, 25))
Y = [12 / x ** 2 for x in X]
p2 = plt.plot([log10(x) for x in X], Y)
plt.show()
```



10^1
10^0
10^{-1}
10^{-2}
0.0
0.5
1.0
1.5
2.0
2.5
3.0