```python
#This is the numeric simulation used in Section 6.1 of the paper
#"Evolution Heritable Risk, and Skewness Loving"
#(2020, Yuval Heller & Arthur Robson).
#The code was written by Renana Heller in Python 3.7.
#The updated code can be found at:
#https://sites.google.com/site/yuval26/numeric-simulation-heller-robson.py
#The updated paper can be found at
#https://sites.google.com/site/yuval26/local-risk.pdf

from random import choices
from collections import Counter
import matplotlib.pyplot as plt
import math
import csv
import time
import winsound

#Time string that is used in the exported file names.
timestr = time.strftime("%Y%m%d-%H%M%S")

for rowindex in range(1,11):
    for colindex in range(1,16):
        #Initial population size
        _numOfPeople = 3000

        #Maximal number of years (=iterations) in each simulation run.
        _numIterations = 20000

        #Number of different dynasties
        #(also called, islands or sub-populations in the simulations)
        _numOfIslands = 300

        #Distribution of heritable birth rate component
        #(denoted by X and q_x in the paper)
        islandPossibleProbs = [0, 0.02]
        islandProbWeights = [0.5, 0.5]

        #Probability of an offspring being born in the same dynasty
        #as the parent (rather than in a random dynasty)
        inheritProb=1

        #Distribution of idiosyncratic birth rate component
        #(denoted by Y and q_y in the paper; in the 150 simulation runs
        #presented in the paper there is no idiosincratic risk)
        ownProbs = [0, 0.02]
        ownWeights = [1, 0]

        #distribution of aggregate birth rate component
        #(denoted by Z and q_z in the paper; in the 150 simulation runs in the
        #paper there is no aggregate risk)
        genPossibleProbs = [0, 0.02]
        genProbsWeights =  [1,0]


        deathProb=0.014
        #ImmProb describes the probability in which each agent migrates to a
        #random dynasty in each year (denoted by \lambda_m in the paper)
        #Each of the following values was tested in 15 simulation runs.
        if rowindex==1:
            immProb=0.0002
        if rowindex==2:
            immProb=0.0004
        if rowindex==3:
            immProb=0.001
        if rowindex==4:
            immProb=0.002
```

```
if rowindex==5:
    immProb=0.004
if rowindex==6:
    immProb=0.007
if rowindex==7:
    immProb=0.01
if rowindex==8:
    immProb=0.013
if rowindex==9:
    immProb=0.016
if rowindex==10:
    immProb=0.018

#islandChgProb describes the probability in which each dynasty redraws
#a new value for its hertiable birth rate in each year (denoted by
#\lambda_x in the paper)
islandChgProb = 0.02-immProb

#probabilty of redrawing the value of the aggregate birth rate
#(not used in the 150 simulation runs)
genChgProb = 0

#Initializing arrays
islandList = [];
islandProbList = [];
populationGrowth = []
PopulationGrowthRate = []
PopulationMaxProb = []
islandPopulation = []
maxIslandPopulation = []
iterationIndex=0

#Randomly choosing the initial aggregate birth rate
genProb=choices(genPossibleProbs,genProbsWeights)[0]

countHighIsl=0;
# Randomly choosing the hertiable birth rate of each dynasty
for index in range(0, _numOfIslands):
    islandProbList += choices(islandPossibleProbs,islandProbWeights )
    islandList += [index]

#Keeping a copy of the initial population size.
initialNumOfPeople = _numOfPeople

#Randomly assiging a dyansty to each person in the initial population
islandIndexPerPerson = choices(islandList, k=_numOfPeople)

#Rabdomly assigining an idiosincratic birth rate to each person
ownProbPerPerson = choices(ownProbs, ownWeights, k=_numOfPeople)
#End of setup of the simulation.

#Starting the run of a single simualtion run.

#As long as we haven't reached the maximal number of years.
while iterationIndex < _numIterations:

    #countMaxProb counts the number of people with the maximal
    #heritable birth rate
    countMaxProb = 0

    #Initating arrays for new agents that will be born in this year
    newPersonIslandIndex = []
    newPersonOwnProb = []
    numNewPerson = 0
    index = 0

    #The simulation run stops if the population size become to small
```

```
#(less than 10)
#or too large (more than 1,000,000)
if (_numOfPeople < 10) or (_numOfPeople>1000000):
    _numIterations = iterationIndex
    break

#This loop goes on all agents in the population.
while index < _numOfPeople:
    #percanet is the total birth rate of the current person
    percent = islandProbList[islandIndexPerPerson[index]] + ownProbPerPerson[index] + genProb

    # Doing a lottery to decide if the agent has a new offspring,
    #according to the agent's birth rate
    shouldRep = choices ([True , False], [percent, 1-percent])

    #Adding a new person in the same dynasty as the parent
    if shouldRep[0]:
        #counting the number of new agents born in each dyansty.
        newPersonIslandIndex += [islandIndexPerPerson[index]]

        #randomly choosing a new idioisncratic birthrate to the new agent
        newPersonOwnProb += choices(ownProbs, ownWeights)
        #Counting the number of new agents born in this year.
        numNewPerson += 1;
        #Doing a lottery if the offspring migrates iimdeitaly when being born
        #(set to 0% in the simulation runs descriebd in the manuscript.)
        newbornimmigrates=choices ([True,False], [1-inheritProb,inheritProb,])

        #Implementing the offspring's migration.
        if newbornimmigrates[0]:
            newIsland = choices(islandList)[0]
            islandIndexPerPerson[index] = newIsland

    #Checking if the agent has to migrate to a new rnadom dynasty.
    shouldImmegrate = choices ([True , False], [immProb, 1-immProb])
    #Implementng the agent's migration.
    if shouldImmegrate[0]:
        #Assiginign a new location to the agent
        newIsland = choices(islandList)[0]
        islandIndexPerPerson[index] = newIsland

    #Checking if the current agent has the maximal local birth rate
    if islandProbList[islandIndexPerPerson[index]] == max (islandPossibleProbs):
        #Counting how many agents have the maximal local birth rate.
        countMaxProb +=1;

    #Checking if agent should die
    shouldDie = choices ([True , False], [deathProb, 1-deathProb])
    #Implementing the agent's death
    if shouldDie[0]:
        # delete the Person from all lists
        del islandIndexPerPerson[index]
        del ownProbPerPerson[index]
        #print (islandIndexPerPerson)
        #decreasing index due to removing the i-th person, and him
        #being replaced with the i+1th person
        index -= 1
        _numOfPeople -= 1

    index += 1

#adding the new agents from the temporary arary to the regular array
islandIndexPerPerson += newPersonIslandIndex
ownProbPerPerson += newPersonOwnProb
_numOfPeople += numNewPerson
```

```
            #Gatherng information for the graphs and the exported CSV file from
            #the current iteration
            PopulationMaxProb += [countMaxProb/_numOfPeople]
            populationGrowth += [_numOfPeople]
            #Starting calculating the growth rate after 100 periods.
            if  iterationIndex>100:
                growthRate=math.log(_numOfPeople/initialNumOfPeople)/(iterationIndex+1)
            else: growthRate=0
            PopulationGrowthRate +=[growthRate]
            iterationIndex += 1

            #Prinitng a point every 250 periods, so that the user will see a signal
            #about the simulation's progress
            if iterationIndex % 250 == 0:
                print (". ", end="" )

            #Counting how many agents are in each dynasty ("island")
            islandCounter = Counter(islandIndexPerPerson)
            islandCounterArray = []
            maxPopulation = 0
            #Calculating the size of the most populous dyansty ("island")
            for index in range(0, _numOfIslands):
                if islandCounter[index] :
                    islandCounterArray += [islandCounter[index]]
                    if islandCounter[index] > maxPopulation:
                        maxPopulation = islandCounter[index]
                else:
                    islandCounterArray += [0]

                #Ranodmly check if each dynasty has to get a new draw of its heritable birth rate.
                shouldIslandChgProb = choices ([True , False], [islandChgProb, 1-islandChgProb])
                if shouldIslandChgProb[0]:
                    newIslandProb = choices(islandPossibleProbs,islandProbWeights )[0]
                    islandProbList[index] = newIslandProb
            #Updaing the size of each dynasty due to the births, deaths and
            #migrations in the current round.
            islandPopulation += [ islandCounterArray ]
            maxIslandPopulation += [maxPopulation]

            #checking if there should be a new lottery for the aggregate birth rate
            shouldGenChgProb = choices ([True , False], [genChgProb, 1-genChgProb]);
            #Implementing a lottery for the aggregate birth rate.

            if shouldGenChgProb[0] :
                genProb=choices(genPossibleProbs,genProbsWeights)[0]

        #finished all calculations of the simulation tun.

        #Printing summary statistics of the simulation run
        print("Row: ",rowindex," Col: ",colindex," #islands=",_numOfIslands," ImmProb=",immProb, " deathprob=", deathProb,"
islChgProb",islandChgProb, " inheritProb",inheritProb)
        print ("Iter.:", iterationIndex, "population:", populationGrowth[iterationIndex-1],"max L birth:
",int(100*PopulationMaxProb[iterationIndex-1]),"%  LR growth: ",int(PopulationGrowthRate[iterationIndex-1]*100000)/1000, "%
max island ", maxIslandPopulation[iterationIndex-2])

#The following command lines allow to print graphs of the population size, population growth rate,
#the share of agents with high hertiable birth rate, and the share of agents in the most populated dynasty
        if colindex==0:
            plt.plot(populationGrowth)
            plt.ylabel('Population')
            plt.show()

            plt.plot(PopulationGrowthRate)
            plt.ylabel('Growth Rate')
            plt.show()
            plt.plot(PopulationMaxProb)
            plt.ylabel('Population with Max Prob')
            plt.show()

            plt.plot(maxIslandPopulation)
            plt.ylabel('Maximal Population in Island')
            plt.show()


        #Time string used for the file names.
        timestr = time.strftime("%Y%m%d-%H%M%S")

        #Creating a detailed CSV file describing the detailed results of the simulation run.
        #One should change the directory based on the local computer in which the simulation runs!
        #Yuval's laptop:'C:\\Users\\heller\\Dropbox\\Local-Risk-Shared\\simulation\\population/sim-results'
        #Yuval's office computer: 'C:/Users/user/Dropbox/risk-persistance/simulation/sim-results/'
        #Renana's laptop: '/Users/heller/Documents/population'
        with open('C:/Users/User/Dropbox/risk-persistance/simulation/sim-results/'+timestr+ '.csv', mode='w',newline='') as
population_file:
            population_file = csv.writer(population_file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
            population_file.writerow(['Row Index', rowindex])
            population_file.writerow(['Col Index', colindex])
            population_file.writerow(['Number Of Islands', _numOfIslands])
            population_file.writerow(['Island Probs' ] + islandPossibleProbs)
            population_file.writerow(['Island Weights'] + islandProbWeights)
            population_file.writerow(['Own Probs'] + ownProbs)
            population_file.writerow(['Own Weights'] + ownWeights)
            population_file.writerow(['General Probs'] + genPossibleProbs)
            population_file.writerow(['General Weights'] + genProbsWeights)
            population_file.writerow(['Immegaration Prob', immProb])
            population_file.writerow(['Island Change Probability', islandChgProb])
            population_file.writerow(['General Change Probability', genChgProb])
            population_file.writerow(['Death Probabilty', deathProb])
            population_file.writerow(['Iteration index', 'Population', 'Growth Rate',
                                      'Population with max Probabilty', ' Maximal Island Population'])
            index=0
            while index<_numIterations:
                population_file.writerow([index, populationGrowth[index], PopulationGrowthRate[index], PopulationMaxProb[index],
maxIslandPopulation[index]])
                index+=100
```