Python code of the simulations

1 Source for: Island model – monogenic selection

```
#! /usr/bin/python

#Object: Simulation of an island model model (monogenic version)
#Aim: Simulate data in order to assess BayeScEnv performances
#Author: Pierre de Villemereuil
#Decembre 2013

#---------------------------------Header--------------------------

print("Importing SimuPOP")

#Setting allele size
from simuOpt import setOptions
setOptions(alleleType='binary') #Binary for two alleles states (SNPs)

#Importing simuPOP
import simuPOP as sim
from simuPOP.utils import saveCSV
from simuPOP.sampling import drawRandomSample
from simuPOP.utils import migrIslandRates

print("Importing some other libraries")

from decimal import *
from math import *
import numpy
import random

print("Script starting")

#-----------------------------Simulation parameters-----------------
#Population(s) size(s)
popsize=500

#Number of generations
numgen=400
atgen=100

#Number of chromosomes
numchrom=10
#Number of loci per chrom.
numloc=500
#Create vector of loci numbers
vecloc=[numloc]*numchrom

#Initializing ancestral population frequencies
p=[0]*numloc*numchrom
for i in range(numloc*numchrom):
        p[i]=0.1+(0.9-0.1)*numpy.random.beta(2,2)       #Almost uniform, but not too much
            weight in the edges plus restricted to (0.1,0.9)

#Migration rate (probability for one individual to disperse)
m=0.0045

#Setting env values
vec_env=[0]*16
j=float(-1.5)
for i in range(16):
        vec_env[i]=round(j,1)
        j=j+0.2

#-----------------------Custom functions--------------------------
#Function naming the alleles
#Depends on the number of chromosomes (chrom) and loci (loc)
def allele_naming(chrom,loc):
        res=[]
        for i in range(chrom):
```

```
                for j in range(1,loc+1):
                        res.append(chr(65+i)+str(j))
        return res


#Function to attribute env values to individuals' info fields
def env_set(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
        numpop=len(subsize)
        for i in range(numpop):
                pop.setIndInfo(vec_env[i],'env',subPop=i)
        return True

#Link function between env value and selection
#Modified 'logitistic' function
def fit_func(x):
        res=(1-exp(-x))/(1+exp(-x))
        res=res/10
        return res

#Defining fitness according to selection
#(0,0) is the reference genotype
def fit_env(geno,env):
        s=fit_func(env)
        if geno[0]+geno[1]==0:
                w = 1
        if geno[0]+geno[1]==1:
                w = 1 + (s/2)
        if geno[0]+geno[1]==2:
                w = 1 + s
        return w


#Function defining a constant subPop size (popsize) for any number of subpops (demographic
    model)
def demo(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
#If subsize is of length 1, then it is a integer and len() does not work
        if type(subsize)==type(1):
                numpop=1
        else:
                numpop=len(subsize)
        vecsize=[popsize]*(numpop)
        return vecsize

#---------------------------Setting up the simulator----------------


for k in range(1,101):
    print('Simulation number '+str(k))
    #Creating initial population of size popsize and "caryotype" vecloc
    pop=sim.Population(size=[popsize]*16,loci=vecloc,infoFields=['migrate_to','fitness','
        env'],lociNames=allele_naming(numchrom,numloc))
    sim.initSex(pop)
    for i in range(numloc*numchrom):
            for j in range(16):
                    tmp_p=numpy.random.beta(p[i]/0.11,(1-p[i])/0.11)
                    sim.initGenotype(pop,freq=[tmp_p,1-tmp_p],loci=i,subPops=j)
    #---------------------------Main evolving process-----------------------
    pop.evolve(
            preOps=[
            sim.PyEval('"Gen: %d" % gen',step=10),
            sim.PyOutput('\n',step=10),
    #-------------------------------------------------
    #Migration using a simple stepping stone model
            sim.Migrator(rate=migrIslandRates(m,16)),
    #-------------------------------------------------
    #Selection process
    #Set environmental value (env infoField) for each individual in the population
    #Takes place at each generation after the first fission
```

```
            sim.PyOperator(env_set),
    #Selection occures at locus 50 according to env information field
            sim.PySelector(fit_env,loci=50,begin=atgen),
            ],
    #-------------------------------------------------
    #Mating at random (pangamy)
            matingScheme=sim.RandomMating(
    #Fixed population size (fixed at 'popsize')
            subPopSize=demo,
    #Recombination to avoid selective sweep
            ops=[sim.Recombinator(rates=0.002)]
            ),
            postOps=[
    #Mutation rate 10e-6
            sim.SNPMutator(u=0.000001,v=0.000001)
            ],
    #-------------------------------------------------
    #Evolve for a number 'numgen' of generations
            gen = numgen
    )
    sim.stat(pop,popSize=True)
    subsize=pop.dvars().subPopSize
    numpop=len(subsize)
    for i in range(numpop):
            pop.setIndInfo(vec_env[i],'env',subPop=i)
    sample = drawRandomSample(pop, sizes=[20]*numpop)
    sample.addInfoFields('pop_name')
    vecname=[]
    for i in range(1,numpop+1):
            vecname=vecname+[i]*20
    sample.setIndInfo(vecname,'pop_name')
    #sim.dump(pop,structure=False)
    print('Saving population')
    saveCSV(sample,filename="sims/sim"+str(k)+".csv",infoFields=['pop_name','env'],
        sexFormatter=None,affectionFormatter=None,header=False)
```

2 Source for: Island model – polygenic selection

```
#! /usr/bin/python

#Object: Simulation of an island model model (polygenic version)
#Aim: Simulate data in order to assess BayeScEnv performances
#Author: Pierre de Villemereuil
#Decembre 2013

#---------------------------------Header--------------------------

print("Importing SimuPOP")

#Setting allele size
from simuOpt import setOptions
setOptions(alleleType='binary') #Binary for two alleles states (SNPs)

#Importing simuPOP
import simuPOP as sim
from simuPOP.utils import saveCSV
from simuPOP.sampling import drawRandomSample
from simuPOP.utils import migrIslandRates

print("Importing some other libraries")

from decimal import *
from math import *
import numpy
import random

print("Script starting")

#----------------------------Simulation parameters----------------
#Population(s) size(s)
popsize=500
```

```
#Number of generations
numgen=400
atgen=100

#Number of chromosomes
numchrom=10
#Number of loci per chrom.
numloc=500
#Create vector of loci numbers
vecloc=[numloc]*numchrom

#Initializing ancestral population frequencies
p=[0]*numloc*numchrom
for i in range(numloc*numchrom):
        p[i]=0.1+(0.9-0.1)*numpy.random.beta(2,2)       #Almost uniform, but not too much
            weight in the edges plus restricted to (0.1,0.9)

#Migration rate (probability for one individual to disperse)
m=0.0045

#Setting env values
vec_env=[0]*16
j=float(-1.5)
for i in range(16):
        vec_env[i]=round(j,1)
        j=j+0.2

#Defining the selected loci
locisel
    =[2793,1850,583,4083,3349,860,4785,706,947,939,1819,925,403,2867,2897,97,3102,2618,708,1190,2471,15


#-----------------------Custom functions--------------------------
#Function naming the alleles
#Depends on the number of chromosomes (chrom) and loci (loc)
def allele_naming(chrom,loc):
        res=[]
        for i in range(chrom):
                for j in range(1,loc+1):
                        res.append(chr(65+i)+str(j))
        return res


#Function to attribute env values to individuals' info fields
def env_set(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
        numpop=len(subsize)
        for i in range(numpop):
                pop.setIndInfo(vec_env[i],'env',subPop=i)
        return True

#Link function between env value and selection
#Modified 'logitistic' function
def fit_func(x):
        res=(1-exp(-x))/(1+exp(-x))
        res=res/50
        return res

#Defining fitness according to selection
#(0,0) is the reference genotype
def fit_env(geno,env):
        N=len(geno)
        s=fit_func(env)
        t11=0
        t00=0
        for i in range(N/2):
                a1=geno[i*2]
                a2=geno[i*2+1]
                if (a1+a2==0):
                        t00=t00+1
                if (a1+a2==2):
```

```
                        t11=t11+1
        w=((1+s)**t11)*((1-s)**t00)
        return w


#Function defining a constant subPop size (popsize) for any number of subpops (demographic
    model)
def demo(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
#If subsize is of length 1, then it is a integer and len() does not work
        if type(subsize)==type(1):
                numpop=1
        else:
                numpop=len(subsize)
        vecsize=[popsize]*(numpop)
        return vecsize

#---------------------------Setting up the simulator----------------


for k in range(1,101):
    print('Simulation number '+str(k))
    #Creating initial population of size popsize and "caryotype" vecloc
    pop=sim.Population(size=[popsize]*16,loci=vecloc,infoFields=['migrate_to','fitness','
        env'],lociNames=allele_naming(numchrom,numloc))
    sim.initSex(pop)
    for i in range(numloc*numchrom):
            for j in range(16):
                    tmp_p=numpy.random.beta(p[i]/0.11,(1-p[i])/0.11)
                    sim.initGenotype(pop,freq=[tmp_p,1-tmp_p],loci=i,subPops=j)
    #-------------------------Main evolving process-------------------------
    pop.evolve(
            preOps=[
            sim.PyEval('"Gen: %d" % gen',step=10),
            sim.PyOutput('\n',step=10),
    #-------------------------------------------------
    #Migration using a simple stepping stone model
            sim.Migrator(rate=migrIslandRates(m,16)),
    #-------------------------------------------------
    #Selection process
    #Set environmental value (env infoField) for each individual in the population
    #Takes place at each generation after the first fission
            sim.PyOperator(env_set),
    #Selection occures at locus 50 according to env information field
            sim.PySelector(fit_env,loci=locisel,begin=atgen),
            ],
    #-------------------------------------------------
    #Mating at random (pangamy)
            matingScheme=sim.RandomMating(
    #Fixed population size (fixed at 'popsize')
            subPopSize=demo,
    #Recombination to avoid selective sweep
            ops=[sim.Recombinator(rates=0.002)]
            ),
            postOps=[
    #Mutation rate 10e-6
            sim.SNPMutator(u=0.000001,v=0.000001),
            ],
    #-------------------------------------------------
    #Evolve for a number 'numgen' of generations
            gen = numgen
    )
    sim.stat(pop,popSize=True)
    subsize=pop.dvars().subPopSize
    numpop=len(subsize)
    for i in range(numpop):
            pop.setIndInfo(vec_env[i],'env',subPop=i)
    sample = drawRandomSample(pop, sizes=[20]*numpop)
    sample.addInfoFields('pop_name')
    vecname=[]
    for i in range(1,numpop+1):
```

```
            vecname=vecname+[i]*20
    sample.setIndInfo(vecname,'pop_name')
    #sim.dump(pop,structure=False)
    print('Saving population')
    saveCSV(sample,filename="sims/sim"+str(k)+".csv",infoFields=['pop_name','env'],
        sexFormatter=None,affectionFormatter=None,header=False)
```

3 Source for: Stepping Stone model – polygenic selection

```
#! /usr/bin/python

#Object: Simulation of stepping stone model (polygenic version)
#Aim: Simulate data in order to assess BayeScEnv performances
#Author: Pierre de Villemereuil
#Decembre 2013

#---------------------------------Header--------------------------

print("Importing SimuPOP")

#Setting allele size
from simuOpt import setOptions
setOptions(alleleType='binary') #Binary for two alleles states (SNPs)

#Importing simuPOP
import simuPOP as sim
from simuPOP.utils import saveCSV
from simuPOP.sampling import drawRandomSample
from simuPOP.utils import migrIslandRates

print("Importing some other libraries")

from decimal import *
from math import *
import numpy
import random

print("Script starting")

#-----------------------------Simulation parameters-----------------
#Population(s) size(s)
popsize=500

#Number of generations
numgen=1000
atgen=700

#Number of chromosomes
numchrom=10
#Number of loci per chrom.
numloc=500
#Create vector of loci numbers
vecloc=[numloc]*numchrom

#Initializing ancestral population frequencies
p=[0]*numloc*numchrom
for i in range(numloc*numchrom):
        p[i]=0.1+(0.9-0.1)*numpy.random.beta(2,2)       #Almost uniform, but not too much
            weight in the edges plus restricted to (0.1,0.9)

#Migration rate (probability for one individual to disperse)
m=0.01

#Defining environment
vec_env
    =[-0.5198065,-0.8112612,-0.2124103,0.3941398,1.1103401,1.5185518,0.6871753,0.0878634,-0.0821056,-0.


#Defining the selected loci
locisel
    =[2793,1850,583,4083,3349,860,4785,706,947,939,1819,925,403,2867,2897,97,3102,2618,708,1190,2471,15
```

```
#-----------------------Custom functions--------------------------
#Function naming the alleles
#Depends on the number of chromosomes (chrom) and loci (loc)
def allele_naming(chrom,loc):
        res=[]
        for i in range(chrom):
                for j in range(1,loc+1):
                        res.append(chr(65+i)+str(j))
        return res


#Function to attribute env values to individuals' info fields
def env_set(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
        numpop=len(subsize)
        for i in range(numpop):
                pop.setIndInfo(vec_env[i],'env',subPop=i)
        return True

#Link function between env value and selection
#Modified 'logitistic' function
def fit_func(x):
        res=(1-exp(-x))/(1+exp(-x))
        res=res/50
        return res

#Defining fitness according to selection
#(0,0) is the reference genotype
def fit_env(geno,env):
        N=len(geno)
        s=fit_func(env)
        t11=0
        t00=0
        for i in range(N/2):
                a1=geno[i*2]
                a2=geno[i*2+1]
                if (a1+a2==0):
                        t00=t00+1
                if (a1+a2==2):
                        t11=t11+1
        w=((1+s)**t11)*((1-s)**t00)
        return w


#Function defining a constant subPop size (popsize) for any number of subpops (demographic
    model)
def demo(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
#If subsize is of length 1, then it is a integer and len() does not work
        if type(subsize)==type(1):
                numpop=1
        else:
                numpop=len(subsize)
        vecsize=[popsize]*(numpop)
        return vecsize


#---------------------------Setting up the simulator----------------

for k in range(1,101):
    print('Simulation number '+str(k))
    #Creating initial population of size popsize and "caryotype" vecloc
    pop=sim.Population(size=[popsize]*16,loci=vecloc,infoFields=['migrate_to','fitness','
        env'],lociNames=allele_naming(numchrom,numloc))
    sim.initSex(pop)
    sim.initSex(pop)
    for i in range(numloc*numchrom):
            for j in range(16):
                    tmp_p=numpy.random.beta(p[i]/0.11,(1-p[i])/0.11)
```

```
                    sim.initGenotype(pop,freq=[tmp_p,1-tmp_p],loci=i,subPops=j)
    #-------------------------Main evolving process-----------------------
    pop.evolve(
            preOps=[
            sim.PyEval('"Gen: %d" % gen',step=10),
            sim.PyOutput('\n',step=10),
    #-------------------------------------------------
    #Migration using a simple stepping stone model
            sim.Migrator(rate=migrSteppingStoneRates(m,16,circular=False)),
    #-------------------------------------------------
    #Selection process
    #Set environmental value (env infoField) for each individual in the population
    #Takes place at each generation after the first fission
            sim.PyOperator(env_set),
    #Selection occures at locus 50 according to env information field
            sim.PySelector(fit_env,loci=locisel,begin=atgen),
            ],
    #-------------------------------------------------
    #Mating at random (pangamy)
            matingScheme=sim.RandomMating(
    #Fixed population size (fixed at 'popsize')
            subPopSize=demo,
    #Recombination to avoid selective sweep
            ops=[sim.Recombinator(rates=0.002)]
            ),
            postOps=[
    #Mutation rate 10e-6
            sim.SNPMutator(u=0.000001,v=0.000001)
            ],
    #-------------------------------------------------
    #Evolve for a number 'numgen' of generations
            gen = numgen
    )
    sim.stat(pop,popSize=True)
    subsize=pop.dvars().subPopSize
    numpop=len(subsize)
    for i in range(numpop):
            pop.setIndInfo(vec_env[i],'env',subPop=i)
    sample = drawRandomSample(pop, sizes=[20]*numpop)
    sample.addInfoFields('pop_name')
    vecname=[]
    for i in range(1,numpop+1):
            vecname=vecname+[i]*20
    sample.setIndInfo(vecname,'pop_name')
    #sim.dump(pop,structure=False)
    print('Saving population')
    saveCSV(sample,filename="sims/sim"+str(k)+".csv",infoFields=['pop_name','env'],
        sexFormatter=None,affectionFormatter=None,header=False)
```

4 Source for: Hierarchically structured model – polygenic selection

```
#! /usr/bin/python

#Object: Simulation of hierarchically structured model (polygenic version)
#Aim: Simulate data in order to assess BayeScEnv performances
#Author: Pierre de Villemereuil
#Decembre 2013

#---------------------------------Header-------------------------

print("Importing SimuPOP")

#Setting allele size
from simuOpt import setOptions
setOptions(alleleType='binary') #Binary for two alleles states (SNPs)

#Importing simuPOP
import simuPOP as sim
from simuPOP.utils import saveCSV
from simuPOP.sampling import drawRandomSample
from simuPOP.utils import migrIslandRates
```

```
print("Importing some other libraries")

from decimal import *
from math import *
import numpy
import random

print("Script starting")

#-----------------------------Simulation parameters-----------------
#Population(s) size(s)
popsize=500

#Number of generations
numgen=600
atgen=[50,150,200,300]

#Number of chromosomes
numchrom=10
#Number of loci per chrom.
numloc=500
#Create vector of loci numbers
vecloc=[numloc]*numchrom

#Migration rate (probability for one individual to disperse)
m=0.0045

#---------------------------Initializing some variables----------------

#Environmental values for the two first populations
vec_env
    =[-0.5198065,-0.8112612,-0.2124103,0.3941398,1.1103401,1.5185518,0.6871753,0.0878634,-0.0821056,-0.


#Defining the loci under selection
locisel
    =[2793,1850,583,4083,3349,860,4785,706,947,939,1819,925,403,2867,2897,97,3102,2618,708,1190,2471,15


#-----------------------
#Function naming the alleles
#Depends on the number of chromosomes (chrom) and loci (loc)
def allele_naming(chrom,loc):
        res=[]
        for i in range(chrom):
                for j in range(1,loc+1):
                        res.append(chr(65+i)+str(j))
        return res

#Function operating the migration for each generation
def migration(pop):
#Extract the number of populations 'numpop'
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
        numpop=len(subsize)
#First iteration step is 2
#i reflects the number of pops (2,4,8,16...)
        i=2
#j reflects the number of steps (1,2,3...)
        j=1
#Seeding iterative process to construct migration rate matrix
#a is a 2x2 matrix
        a=zeros((2,2))
        a[1][0]=1
        a[0][1]=1
        while i<numpop:
#while the number of pops is not reached
#note that for numpop=2, the loop is not activate
        #i doubles
                i=2*i
        #incrementing j
                j+=1
```

```
        #tmp is a submatrix 2x2 containing the 'migration coefficient' i/2 (1/2 for 4 pops,
             1/4 for 8 pops, etc..)
                tmp=zeros(((i/2),(i/2)))+(float(1)/(i/2))
        #a is updated to contain the coefficients in anti-diag submatrices
        #a is now 4x4, then 8x8 for 16 pops
                a=hstack((vstack((a,tmp)),vstack((tmp,a))))
#End while
#the matrix needs to be scaled to sum to 1
#the sum to 1 is needed for m (migration rate) to be a relevant biological parameter
#Scaled by the number of steps j (each step adding exactly 1 to each row (1, or 2*1/2 or
    4*1/4, etc...))
        res=(a/j)*m
#res is an array, we need a nested list
        A=res.tolist()
#And now migration finally happens !
        sim.migrate(pop,rate=A)
        return True

#Function to update the environmental values for each population
#Env values are stored in the global variable vec_env
#New values are drawn from a normal distribution
#with the env value of the mother pop as mean
def env_update(pop):
        global vec_env
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
        numpop=len(subsize)
#Already fixed for numpop==2
        if numpop>2:
        #k is the number to create the two new values (x=x0+k ou x=x0-k)
                k=1.6/float(numpop)
        #tmp will recieve the new env values
                tmp=[0]*numpop
                for i in range(numpop):
                        #if we are left to the old value (x0)
                        if (i%2==0):
                                #i/2 is the result of an euclidian division
                                tmp[i]=round(vec_env[i/2]-k,1)
                        #else, we are right to the old value (x0)
                        else:
                                tmp[i]=round(vec_env[i/2]+k,1)
                vec_env=tmp
        return True

#Function to attribute env values to individuals' info fields
def env_set(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
        numpop=len(subsize)
        for i in range(numpop):
                pop.setIndInfo(vec_env[i],'env',subPop=i)
        return True

#Link function between env value and selection
#Modified 'logitistic' function
def fit_func(x):
        res=(1-exp(-x))/(1+exp(-x))
        res=res/50
        return res


#Defining fitness according to selection
#(0,0) is the reference genotype
def fit_env(geno,env):
        N=len(geno)
        s=fit_func(env)
        t11=0
        t00=0
        for i in range(N/2):
                a1=geno[i*2]
                a2=geno[i*2+1]
                if (a1+a2==0):
```

```
                        t00=t00+1
                if (a1+a2==2):
                        t11=t11+1
        w=((1+s)**t11)*((1-s)**t00)
        return w


#Function defining a constant subPop size (popsize) for any number of subpops (demographic
    model)
def demo(pop):
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
#If subsize is of length 1, then it is a integer and len() does not work
        if type(subsize)==type(1):
                numpop=1
        else:
                numpop=len(subsize)
        vecsize=[popsize]*(numpop)
        return vecsize



#---------------------------Setting up the simulator----------------

for k in range(1,101):
        #Environmental values for the two first populations
        print('Simulation number '+str(k))
        #Creating initial population of size popsize and "caryotype" vecloc
        pop=sim.Population(size=[popsize],loci=vecloc,infoFields=['migrate_to','fitness','
            env'],lociNames=allele_naming(numchrom,numloc))
        #--------------------------Main evolving process-----------------------
        pop.evolve(
        #Initializing sex and genotype
                initOps=[
                        sim.InitSex(),
                        sim.InitGenotype(freq=[0.5,0.5]),
                ],
                preOps=[
        #Splitting each population into two at 'atgen' generations
                sim.SplitSubPops(proportions=[0.5,0.5], at=atgen),
                sim.PyOutput('--------*-Split-*---------\n',at=atgen),
                sim.PyEval('"Gen: %d" % gen',step=10),
                sim.PyOutput('\n',step=10),
        #Calling function 'migration' for individuals migration
        #(Operator Migrator does not allow for varying number of subpopulations)
                sim.PyOperator(migration,begin=atgen[0]),
        #Selection process
        #Set environmental value (env infoField) for each individual in the population
        #Takes place at each generation after the first fission
                sim.PyOperator(env_set,begin=atgen[3]),
        #Selection occures at locus locisel according to env information field
                sim.PySelector(fit_env,loci=locisel,begin=atgen[3]),
                ],
        #Mating at random (pangamy)
                matingScheme=sim.RandomMating(
        #Fixed population size (fixed at 'popsize')
                subPopSize=demo,
        #Recombination to avoid selective sweep
                ops=[sim.Recombinator(rates=0.002)]
                ),
                postOps=[
        #Mutation rate 10e-6
                sim.SNPMutator(u=0.000001,v=0.000001)
                ],
        #Evolve for a number 'numgen' of generations
                gen = numgen
        )
        sim.stat(pop,popSize=True)
        subsize=pop.dvars().subPopSize
        numpop=len(subsize)
        for i in range(numpop):
                pop.setIndInfo(vec_env[i],'env',subPop=i)
```

```
        sample = drawRandomSample(pop, sizes=[20]*numpop)
        sample.addInfoFields('pop_name')
        vecname=[]
        for i in range(1,numpop+1):
                vecname=vecname+[i]*20
        sample.setIndInfo(vecname,'pop_name')
        #sim.dump(pop,structure=False)
        print('Saving population')
        saveCSV(sample,filename="sims/sim"+str(k)+".csv",infoFields=['pop_name','env'],
            sexFormatter=None,affectionFormatter=None,header=False)
```