Markov Chains and Genetic Drift

MATH 2270, Spring 2018

Chase Stolworthy

In [23]:
```
import numpy as np
import pandas as pd
from scipy.stats import binom
from sympy.stats import Binomial, density
from sympy import *
import seaborn as sns
init_printing()
```

Markov Chain

A Markov chain is a mathematical model used to describe processes where certain events repeatedly occur in discrete time intervals. The process starts with an initial state, and the next state only depends on the previous state. A Markov chain can be represented using matrices and vectors. The *state space* is the set of all possible states of the system. A *probability vector* is a vector whose entries sum to 1. A *transition matrix* is a square matrix that describes the probability of transitioning between any two states in the state space. Each column of the transition matrix is a probability vector. A *state vector* describes the current state of the system. It is a probability vector where each entry is the probability of the system being in that particular state. The Markov process is modeled by the repeated multiplication of the transition matrix and state vectors. That is, given an initial state vector $\mathbf{x_0}$ and transition matrix P,

$$\mathbf{x_1} = P\mathbf{x_0}, \quad \mathbf{x_2} = P\mathbf{x_1}, \quad \ldots, \quad \mathbf{x_k} = P\mathbf{x_{k-1}}$$

Genetic Drift

Genetic drift is an evolutionary process that is always occuring in every population. Rather than being caused from natural selection, it is the result of the random sampling of organisms during reproduction. It is often said that genetic drift is the "sampling error" that occurs during reproduction. A common model used to understand genetic drift is the Wright-Fisher model. This model makes the assumptions that the population is at a constant size of $2N$, there are two alleles or gene variants in the population, denoted A_1 and A_2, and reproduction of the entire population happens in discrete non-overlapping intervals. A new generation is created by choosing $2N$ alleles uniformly at random with replacement from the previous generation. Thus, the number of each type of allele in the next generation is a binomial random variable. Let n be the number of A_1 in the current generation. Then, the probability of having x A_1 alleles in the next generation is given by

$$p(x) = \binom{2N}{x}\left(\frac{n}{2N}\right)^x\left(1-\frac{n}{2N}\right)^{2N-x}$$

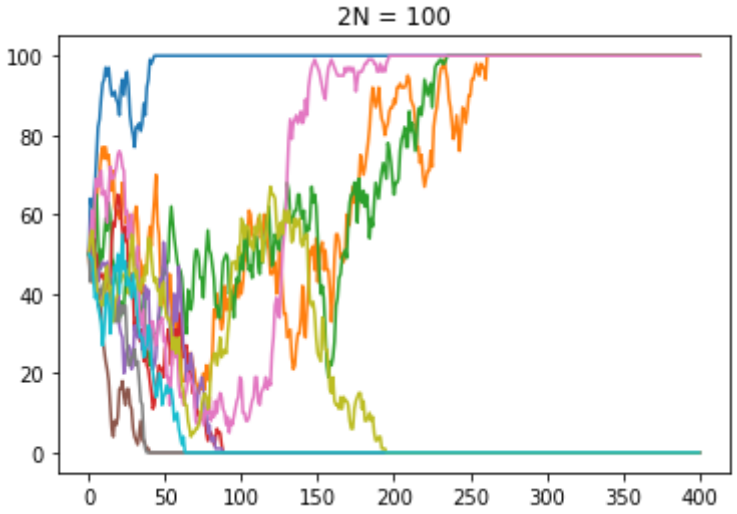
Just using a binomial random variable, we can simulate genetic drift.

In [6]:
```
def simulate_drift(popsize, num_alleles, generations, trials):
    df = pd.DataFrame()
    for trial in range(trials):
        nums = [num_alleles]
        for x in range(generations):
            p = nums[-1]/popsize
            nums.append(np.random.binomial(popsize, p))
        df[trial] = nums
    return df
```

The following simulation uses a population size of 100 individuals, 50 of which are type A_1. The simulation is performed for 400 generations and is repeated 10 times. As you can see, eventually, all the individiuals are either all type A_1 or all type A_2. This is a natural property of genetic drift and we will explore it further by using a Markov chain.

In [22]:
```
p100 = simulate_drift(100, 50, 400, 10)
p100.plot(title="2N = 100", legend=False)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x11355ba58>



Markov Chain Model

Since there are $2N$ individuals in the population and two gene variants, the state space has a size of $2N$. Each entry of the transition matrix p_{ij} is the probability of choosing i alleles of type A_1 from a population of j alleles of type A_1, that is,

$$p_{ij} = \binom{2N}{i}\left(\frac{j}{2N}\right)^i\left(1-\frac{j}{2N}\right)^{2N-i}$$

In [24]:
```
def create_trans_matrix(popsize):
    row_vectors = [[binom.pmf(x, popsize, n/popsize) for n in range(pops
ize+1)] for x in range(popsize+1)]
    return np.matrix(row_vectors)
```

Here is an example of transition matrix for a population of 2 individuals:

In [25]:
```
A = create_trans_matrix(2)
Matrix(A)
```

Out[25]:
$$\begin{bmatrix} 1.0 & 0.25 & 0.0 \\ 0.0 & 0.5 & 0.0 \\ 0.0 & 0.25 & 1.0 \end{bmatrix}$$

As you can see, each column in the transition matrix is a probability vector. Each entry of the state vector x_{k_i} is the probability that there are i alleles of type A_1 in state k. For the initial state vector, we know the number of alleles in the population, lets call it n, so the n-th entry of the vector would be 1 and all the other entries would be 0.

In [26]:
```
def create_init_vec(popsize, n):
    return np.matrix([[1] if n == x else [0] for x in range(popsize+1)])
```

Here is an example of an initial state vector for population of 2 individuals, and there is 1 individual of type A_1 and 1 individual of type A_2.

In [27]:
```
x0 = create_init_vec(2, 1)
Matrix(x0)
```

Out[27]:
$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Running the Markov Chain

We can use the following code to run the Markov chain n times.

In [28]:
```
def markov_chain(n, A, x0):
    x = x0
    for i in range(n):
        x = A*x
    return x
```

Let's create a population with 20 individuals, and where there are 15 individuals of type A_1.

In [29]:
```
A = create_trans_matrix(20)
x = create_init_vec(20, 15)
```

Here is the Markov chain at x_0, x_{10}, x_{100}:

In [30]:
```
Matrix(markov_chain(0, A, x)), Matrix(markov_chain(10, A, x)), Matrix(ma
rkov_chain(100, A, x))
```

Out[30]:

$$\left(\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0.0132830478583497\\0.00803643990906102\\0.0106791420775963\\0.0130980794555976\\0.0156415170192074\\0.0183419125925093\\0.0211912791542938\\0.0241769847425858\\0.0272817277345263\\0.030484184502242\\0.0337590670543826\\0.0370764581837244\\0.0404004569594089\\0.043686647813364\\0.0468767768940618\\0.0498859582779421\\0.0525785098403364\\0.0547290762332845\\0.0555762266705487\\0.0510431843486633\\0.352173322678301\end{bmatrix}, \begin{bmatrix}0.246889940354622\\0.00027736353552148\\0.000314339326267758\\0.000325647755941447\\0.000331360285717322\\0.000334940347358267\\0.000337258902562926\\0.000338786894199656\\0.000339764840671898\\0.000340313629100173\\0.000340492206280348\\0.000340318455779993\\0.000339774467920762\\0.000338801266074201\\0.000337277924072361\\0.00033496386416274\\0.000331388043679198\\0.000325679327047057\\0.000314373705096583\\0.000277396756038421\\0.746889818111852\end{bmatrix}\right)$$

As you can see, eventually the population acheives one of two states: either all A_1 individuals or all A_2 individuals. This is competely inline with the computer simulation run before. We can can see interesting properties of this process by looking closer at the transistion matrix.

Steady-State Vector

A steady-state vector for a transition matrix P is a vector $\mathbf{q}$ such that $P\mathbf{q} = \mathbf{q}$. The following theorem (Lay Theorem 4.18) describes an interesting property about the transition matrix of a Markov chain and its steady-state vector.

Theorem: If P is an $n \times n$ regular stochastic matrix, then P has a unique steady-state vector $\mathbf{q}$. Further, if $\mathbf{x_0}$ is any initial state and $\mathbf{x_{k+1}} = P\mathbf{x_k}$ for $k = 1, 2, \ldots,$ then the Markov chain $(\mathbf{x_k})$ converges to $\mathbf{q}$ as $k \to \infty$.

Thus every transition matrix has a steady-state vector, and the Markov chain converges to this vector. This falls inline with what we saw earlier, that is, eventually all the genetic diversity is lost and the population arrives to a steady state. To find the steady-state vector, we can use eigenanalysis. Since we are looking for a probability vector, we are interested in the eigenvectors corresponsing to the eigenvalue of 1. For a population of size 5:

In [31]:
```
A = create_trans_matrix(5)
w, v = np.linalg.eig(A)
print('Eigenvalues:', w)
Matrix(v)
```

```
Eigenvalues: [1.     1.     0.8    0.48   0.0384 0.192 ]
```

Out[31]:

$$\begin{bmatrix} 1.0 & 0.0 & 0.576923076923076 & 0.377168254570678 & 0.0629940788348712 \\ 0.0 & 0.0 & -0.269230769230769 & -0.551245910526377 & -0.314970394174356 \\ 0.0 & 0.0 & -0.307692307692308 & -0.232103541274264 & 0.629940788348712 \\ 0.0 & 0.0 & -0.307692307692308 & 0.232103541274263 & -0.629940788348712 \\ 0.0 & 0.0 & -0.26923076923077 & 0.551245910526377 & 0.314970394174356 \\ 0.0 & 1.0 & 0.576923076923077 & -0.377168254570678 & -0.0629940788348712 \end{bmatrix}$$

From this, we see that the transition matrix has an eigenvalue of 1 with mulitplicity 2. This corresponding eigenvectors are $(1, 0, \cdots, 0, 0)$ and $(0, 0, \cdots, 0, 1)$. What this means is that eventually the population will reach a steady-state of having no genetic variation. That is, either all A_1 individuals or all A_2 individuals.

Eigendecomposition of Transition Matrix

Since the transition matrix is diagonizable, we can look at the eigenvector decomposition of each state vector to see what happens when k goes to infinity. Suppose we arrange the eigenvalues in descending order, that is $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$ and the set $\{\mathbf{v_1}, \mathbf{v_2} \ldots, \mathbf{v_n}\}$ are the eigenvectors corresponding to this order. Then the kth state vector can be written as (Lay Ch. 5.6),

$$\mathbf{x_k} = c_1(\lambda_1)^k \mathbf{v_1} + \cdots + c_n(\lambda_n)^k \mathbf{v_n}$$

If we let $k \to \infty$, since all eigenvalues except λ_1 are less than one, then all the terms will go zero except the first term. This corresponds to what we have already seen, that the population will reach a steady-state given by an eigenvector corresponding to $\lambda = 1$. If we factor out λ_1^k, we get

$$\mathbf{x_k} = \lambda_1^k(c_1 \mathbf{v_1}) + c_1(\lambda_2/\lambda_1)^k \mathbf{v_1} + \cdots + c_n(\lambda_n/\lambda_1)^k \mathbf{v_n}$$

This shows that the steady-state of the population is being approached exponentially at a rate of $\lambda_2/\lambda_1 = \lambda_2$. Other theorems in population genetics using methods other than linear algebra agree with this result and show that the steady-state is approached at a rate of $1 - 1/2N$. Indeed, if we compare the value of λ_2 with this value we see that the values are very close.

In [32]:
```
A = create_trans_matrix(25)
w, v = np.linalg.eig(A)
print('lambda_2 = ', w[2])
print('1 - 1/2N = ', 1 - 1/25)
```

```
lambda_2 =  0.9599999999999991
1 - 1/2N =  0.96
```

References

1. David Lay, Steven Lay, Judi McDonald, *Linear Algebra and Its Applications*, 5th edition
2. John Gillespie, *Population Genetics*, 2nd edition
3. http://mathworld.wolfram.com/EigenDecomposition.html (http://mathworld.wolfram.com/EigenDecomposition.html)
4. http://mathworld.wolfram.com/MarkovChain.html (http://mathworld.wolfram.com/MarkovChain.html)