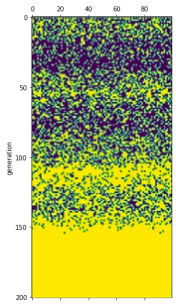
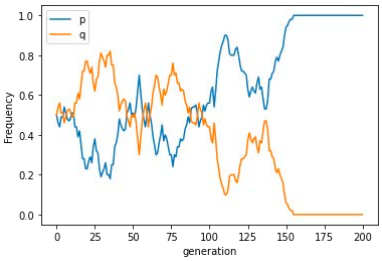
数理生物学演習

第６回　ランダムな現象：
遺伝的浮動，ライト-フィッシャーモデル





野下 浩司（Noshita, Koji）
noshita@morphometrics.jp
https://koji.noshita.net
理学研究院 数理生物学研究室

第６回：ランダムな現象：遺伝的浮動，ライト-フィッシャーモデル

本日の目標

- ハーディ-ワインベルグ平衡の導出
- ライト-フィッシャー モデルの解析
- 擬似乱数

ハーディー-ワインベルグ モデル

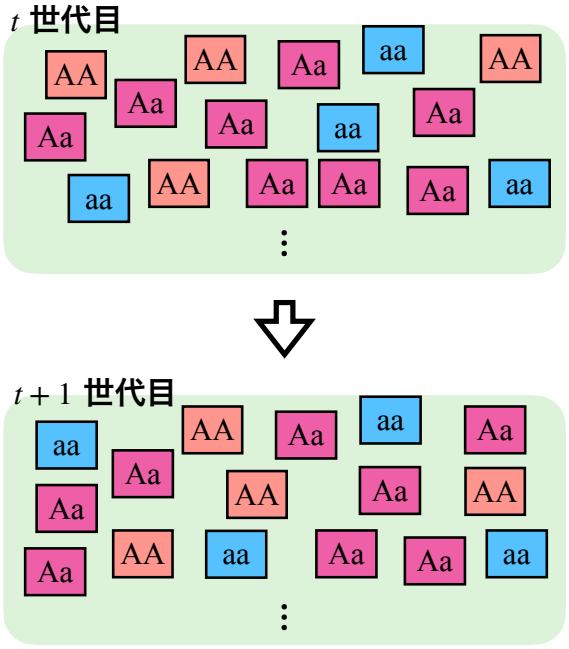
世代を越えて遺伝子型頻度が維持されるケース

仮定

二倍体の有性生殖する生物を考える.
次のような性質を持つと仮定する.

- ランダム交配する
- 世代は重ならない
- 突然変異は起こらない

この生物の十分大きな集団において
中立な対立遺伝子AとaIに注目する.



ある世代における遺伝子型AA，Aa，aaそれぞれの頻度は，前の世代の配偶子中のAとaの頻度 pとq（ただし$p+q=1$）を用いて，

$$\mathrm{AA}:\mathrm{Aa}:\mathrm{aa}=p^2:2pq:q^2$$

となる.

この遺伝子型頻度は世代を越えて維持され，ハーディ-ワインベルグ平衡と呼ばれる.

次世代の遺伝子型頻度を計算して
遺伝子型頻度が本当に維持されているか
確認してみよう

補足

t世代目　$\mathrm{AA}:\mathrm{Aa}:\mathrm{aa}=p^2:2pq:q^2$

t世代目の配偶子　$\begin{cases}\text{Aの頻度}=\\ \text{aの頻度}=\end{cases}$?

t + 1世代目　$\mathrm{AA}:\mathrm{Aa}:\mathrm{aa}=$?

t世代目

$$\mathrm{AA} : \mathrm{Aa} : \mathrm{aa} = p^2 : 2pq : q^2$$

t世代目の配偶子

$$\begin{cases} \textbf{Aの頻度} = p^2 + pq = p^2 + p(1-p) = p \\ \textbf{aの頻度} = pq + q^2 = (1-q)q + q^2 = q \end{cases}$$

t + 1世代目

$$\mathrm{AA} : \mathrm{Aa} : \mathrm{aa} = p^2 : 2pq : q^2$$

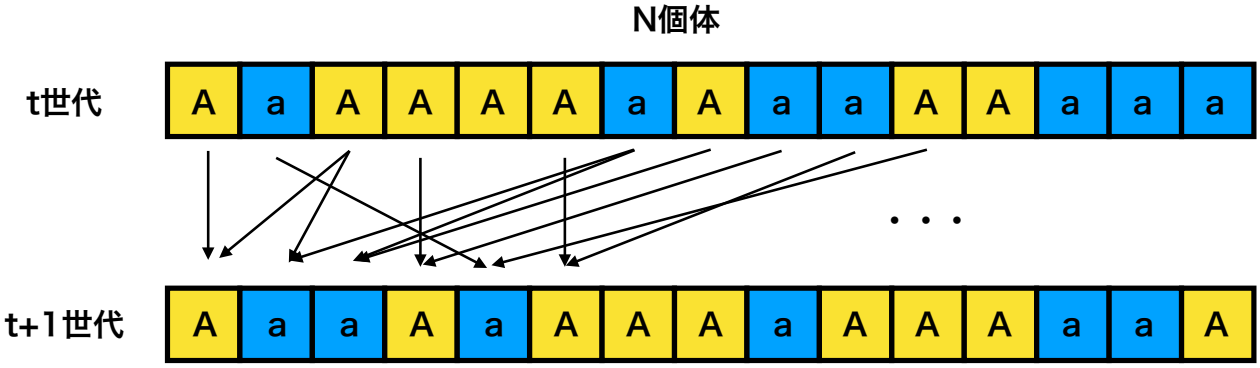
ライト-フィッシャー モデル

遺伝的浮動を考えてみる

半数体の生物**N個体からなる集団**について，ある中立な対立遺伝子A, aに注目する.
次のような性質を持つとする.

- ランダム交配する
- 世代は重ならない
- （追加での）突然変異は起こらない

もう少し詳しく知りたい人はHartl & Clark, Principles of Population Genetics 4th ed.の第3章がおすすめ



集団サイズはN個体で一定
親個体をランダムに選び，どちらの遺伝子を引き継ぐかはランダムに決まる

擬似乱数

擬似乱数

- ある確率分布に従うランダムな数値の系列（乱数列）を生成したいが，“真に”ランダムな数値を得ることは難しい
- **決定論的なアルゴリズム**によって本当の乱数列と（特定の目的上）区別がつかない数値（擬似乱数）列を生成し，これで代替することが一般的
- Pythonのrandomモジュールではメルセンヌ・ツイスタと呼ばれる疑似乱数生成器が使われている

```
# 01-01. 擬似乱数列の生成
import random

for i in range(100):
    r = random.randrange(1,1001)
    print(r)
```

出力
291
602
997
. . .
891

毎回異なる結果が表示される

！注意：疑似乱数関係のモジュール（randomモジュール）をimportする

randomモジュール
- randrange(終了)
- randrange(開始, 終了)
- randrange(開始, 終了, ステップ)
  それぞれ，range(終了)，range(開始, 終了)，range(開始, 終了, ステップ)の要素からランダムに一つ要素を返す．
  例）random.randrange(100)：0〜99の中からランダムに返す．

擬似乱数の種（シード）

- シードを指定すると擬似乱数列は一意に決まる．
- 同じ疑似乱数列を利用できると便利なケースがある（テスト，シミュレーションの再現性など）．
- シードをどのように設定するかは状況によるが，本演習では通常はNoneでの初期化（システム時刻）を用いる．それ以外のものを特別に設定する場合は指示する．

randomモジュール
- seed(シード)
  乱数生成器の初期化．乱数の種（シード）を設定する

```
# 01-02. シード
import random

random.seed(1)
for i in range(100):
    r = random.randrange(1,1001)
    print(r)
```

出力
```
138
583
868
 . . .
311
```

乱数の種（シード）が同じなので毎回同じ結果が表示される

乱数生成：シーケンス（1）

シーケンス（リストなど）に対してランダムな処理（シャッフルやサンプリング）をおこなう

```
# 01-03. シーケンス操作 choice, choices
a_list = [23, 22, 32, 12, 31, 30, 3, 35, 26, 36]

# choice
print("# random.choice")
for i in range(30):
    r = random.choice(a_list)
    print(r)

# choices
weights = [0,1,1,1,1,1,1,1,1,10]
print("# random.choices")
for i in range(30):
    r = random.choices(a_list, weights=weights, k = 5)
    print(r)
```

重みを指定しない場合は，すべて同じ重み

randomモジュール
- choice(シーケンス)
  シーケンスからランダムに要素を返す
- choises(シーケンス，weights=重み，k=要素数)
  シーケンスから相対的な重み weight に基づき，（重複を許し）ランダムにk個の要素からなるリストを返す

乱数生成：シーケンス（2）

シーケンス（リストなど）に対してランダムな処理（シャッフルやサンプリング）をおこなう

```
# 01-04. シーケンス操作 sample
a_list = [23, 22, 32, 12, 31, 30, 3, 35, 26, 36]

# sample
print("# random.sample")
for i in range(30):
    r = random.sample(a_list, k = 5)
    print(r)

# シャッフル
print("# random.sample シャッフル")
for i in range(30):
    r = random.sample(a_list, k = len(a_list))
    print(r)
```

randomモジュール
- sample(シーケンス, k)
  シーケンスから重複のないk個の要素からなるリストを返す

len(シーケンス)
シーケンスのサイズ（要素数）を返す

乱数生成：連続確率分布

特定の確率分布に従う擬似乱数列を生成する

```
# 01-05. 連続確率分布

# 一様分布
# random
r_list_uniform_1 = []
for i in range(100):
    r = random.random()
    r_list_uniform_1.append(r)
print("一様分布（0~1）：", r_list_uniform_1)

# uniform
r_list_uniform_2 = []
for i in range(100):
    r = random.uniform(5, 10)
    r_list_uniform_2.append(r)
print("一様分布（5~10）：", r_list_uniform_2)

# 正規分布
r_list_Gauss = []
for i in range(100):
    r = random.gauss(0, 1)
    r_list_Gauss.append(r)
print("正規分布：", r_list_Gauss)
```

randomモジュール
- random()
  [0,1)の範囲の浮動小数点数（float型）の値をランダムに返す（一様分布）.
- uniform(a, b)
  一様分布．a≦x≦b（a>bならb≦x≦a）の範囲のランダムな浮動小数点数xを返す．
- gauss(mu, sigma)
  平均mu，標準偏差sigmaの正規分布に従う浮動小数点数を返す．

5以上10以下の一様乱数

平均0，標準偏差1の正規分布

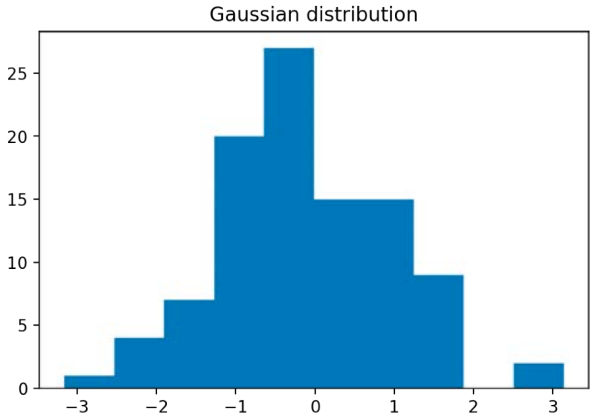
他にも利用できる連続確率分布関数があるので興味のある人は調べてみよう

ヒストグラム

サンプリングされた分布を可視化する

```
# 01-06. 分布のプロット
import matplotlib.pyplot as plt

plt.hist(r_list_Gauss)
plt.title("Gaussian distribution")
```

先程の正規分布からサンプリングした結果（r_list_Gauss）をプロット



matplotlib.pyplotモジュール
- hist(シーケンス)
  シーケンスのヒストグラムをプロット.

他の分布やパラメータを変えていろいろプロットしてみよう！

ループからの脱出：break文

- ループから強制的に抜け出したいときがある
- forの範囲や条件分岐をうまく設定できれば良いが,
  細かく継続判定を設定し分割して判定するよう処理したい場合などに用いる

- break文
  一番内側のループを終了し,
  そのループの次の文へ処理が進む

```
# 01-07. ループの中断
for i in range(100):
    print(i)
    if i == 10:
        break
print("ループ終了")
```

ループから脱出する
(i>10でforループは実行されない)

```
出力
0
1
2
3
4
5
6
7
8
9
10
ループ終了
```

forなどのループで利用できる

```
# 01-08. ネストされたループの中断
for i in range(20):
    for j in range(20):
        print(i,", ",j)
        if j == 5:
            break
```

breakした一番内側のループが終了する（その外側は通常どおり）

```
出力
0 ,  0
0 ,  1
0 ,  2
0 ,  3
0 ,  4
0 ,  5
1 ,  0
1 ,  1
1 ,  2
1 ,  3
1 ,  4
1 ,  5
2 ,  0
2 ,  1
...
19 ,  4
19 ,  5
```

待ち時間，ランダムウォーク

サイコロの目の総和が100を超えるまでの待ち時間

```
# 02-01. サイコロの目の総和が100を超えるまでの待ち時間
import random

x = 0
for i in range(100):
    r = random.randrange(1,7)
    x = x + r
    print(str(i+1)+"回目：",x)
    if x >= 100:
        print(str(i+1)+"回目で100を超えた．")
        break
```

x：サイコロの目の総和

xが100以上ならばステップ数を出力してループを脱出

サイコロ（6面ダイス）のモデル

```
random.randrange(1,7)
```

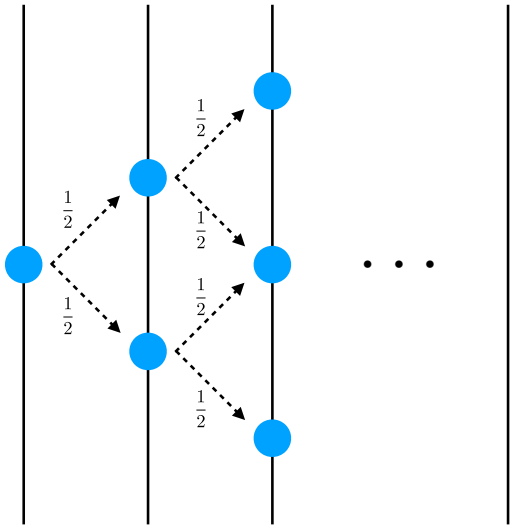
1以上6以下の整数をランダムに（一様分布に従って）返す

```
# 出力
1回目： 4
2回目： 9
3回目： 12
…
25回目： 96
26回目： 101
26回目で100を超えた．
```

（〜回目の部分は）実行するたびに変わる

ランダムウォーク

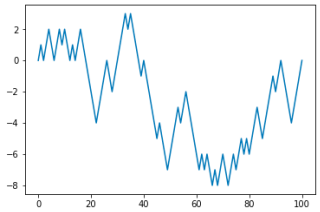
以下のような1次元の単純ランダムウォークを
シミュレーションしてみよう.




```
# 02-02a. ランダムウォーク choice
x = 0
x_list = [x]
for i in range(100):
    r = random.choice([-1,1])
    x = x + r
    x_list.append(x)
```

```
# 02-02b. ランダムウォーク randrange
x = 0
x_list = [x]
for i in range(100):
    r = random.randrange(-1,2,2)
    x = x + r
    x_list.append(x)
```

結果をプロットしてみよう



解釈例：

- コインをなげて表が出れば+1点，裏が出れば-1点を得るゲームの点数の推移
- 右と左にそれぞれ1/2の確率で移動する生物の移動軌跡

ランダムウォークの待ち時間

ある値に到達するまでの繰り返し回数を計算する

```
# 02-02a. ランダムウォーク choice
x = 0
x_list = [x]
for i in range(100):
    r = random.choice([-1,1])
    x = x + r
    x_list.append(x)
```

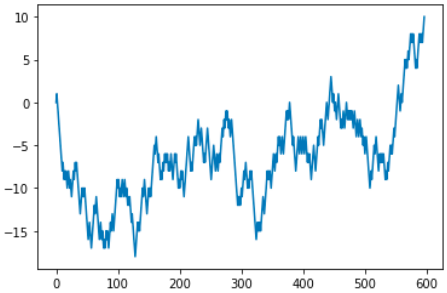
- len(シーケンス)
  シーケンスの長さを返す

```
# 02-03. ランダムウォークの待ち時間
x = 0
x_list = [x]
for i in range(10000):
    r = random.choice([-1,1])
    x = x + r
    x_list.append(x)
    if x >= 10:
        break
print("待ち時間：", len(x_list)-1)
plt.plot(x_list)
```

xが10に到達したら
計算を打ち切る

繰り返し回数（リストの長さ-1）
を表示する

```
# 出力
待ち時間： 596
```



ライト-フィッシャー モデルのシミュレーション と突然変異固定までの待ち時間

ライト-フィッシャー モデル

```
# 02-04. ライト-フィッシャー モデル
pop_size = 100 # 個体数
gen_end = 200 # 最終世代

# aの初期値の設定
a = []
for i in range(pop_size):
    if i % 2 == 0:
        a.append(0)
    else:
        a.append(1)

a_list = [a.copy()]

for t in range(gen_end):
  # a_newの初期化
  a_new = []
  for i in range(pop_size):
    p1 = random.randrange(pop_size)
    p2 = random.randrange(pop_size)
    r = random.choice([a[p1],a[p2]])
    a_new.append(r)

  a = a_new.copy()
  a_list.append(a.copy())

# 結果の表示
for a in a_list:
  print(a)
```

p1番目とp2番目の個体が親として選ばれる

どちらの親から遺伝子を引き継ぐか，確率1/2でランダムに決まる

初期値として半分の個体が『0』もう半分の個体が『1』を持つとする

a: 集団の注目している対立遺伝子を記録するリスト
a_new: 次世代集団の注目している対立遺伝子を記録するリスト
a_list: 各世代のaを記録するリスト

```
# 出力
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, …]
[1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, …]
[1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, …]
⋮
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …]
```

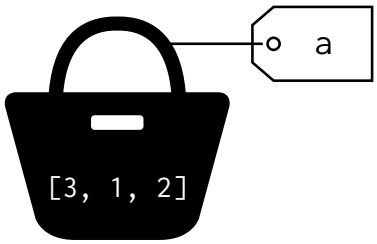
長さがpop_sizeのリスト

最終的に0もしくは1だけからなるリストに落ち着く（突然変異の固定）

- リスト.copy()
  リストの浅いコピーを返す
  （詳しくは後述）

リストのコピー（1）

復習：Pythonにおける変数と代入



変数

オブジェクト

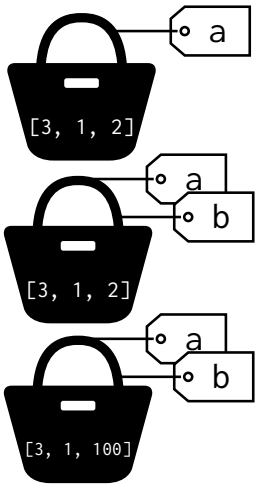
- オブジェクト：「数値」や「文字」の“容器”
- 変数：オブジェクトにつけられた”ラベル”

Pythonの代入操作ではオブジェクトがコピーされるわけではなく，オブジェクトに新たなラベル（変数）をつけている

例.

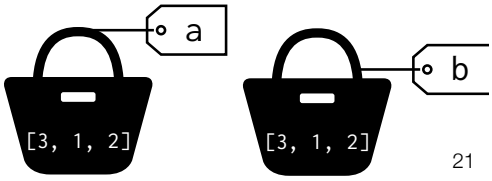
```
a  = [3, 1, 2]

b = a

b[2] = 100

print(a)
```

```
# 出力
[3, 1, 100]
```



結果，何が起こるか？
→ 新たな変数側で要素を変更すると，前の変数側にも影響が出る

ネストされたリストだともう少しややこしくなる



リストのコピー（2）

```
# 問題にならないケース
a_list = []
for i in range(10):
  a = [i, i, i]
  a_list.append(a)

print(a_list)
```

```
# 出力
[[0, 0, 0],
 [1, 1, 1],
 [2, 2, 2],
 [3, 3, 3],
 [4, 4, 4],
 [5, 5, 5],
 [6, 6, 6],
 [7, 7, 7],
 [8, 8, 8],
 [9, 9, 9]]
```

aはforループ内のローカル変数で，ループが終了するとオブジェクトとのつながりが消える

```
# 問題になるケース
a_list = []
a = [0,0,0]
for i in range(10):
  for j in range(3):
    a[j] = i
  a_list.append(a)

print(a_list)
```

```
# 出力
[[9, 9, 9],
 [9, 9, 9],
 [9, 9, 9],
 [9, 9, 9],
 [9, 9, 9],
 [9, 9, 9],
 [9, 9, 9],
 [9, 9, 9],
 [9, 9, 9],
 [9, 9, 9]]
```

a_listに追加されたaがすべて同じオブジェクトを参照している

```
# 修正
a_list = []
a = [0,0,0]
for i in range(10):
  for j in range(3):
    a[j] = i
  a_list.append(a.copy())

print(a_list)
```

```
# 出力
[[0, 0, 0],
 [1, 1, 1],
 [2, 2, 2],
 [3, 3, 3],
 [4, 4, 4],
 [5, 5, 5],
 [6, 6, 6],
 [7, 7, 7],
 [8, 8, 8],
 [9, 9, 9]]
```

copyによりオブジェクトの各要素がコピーされ渡される

ライト-フィッシャー モデルの結果の可視化

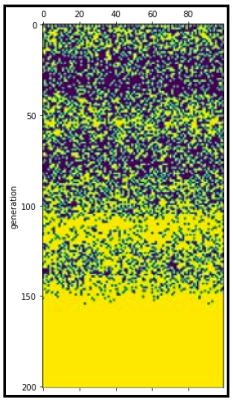
```
# 出力
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, …]
[1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, …]
[1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, …]
⋮
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …]
```

長さがpop_sizeのリスト

最終的に0もしくは1だけからなるリスト
に落ち着く（突然変異の固定）

このような結果でも理解できないことはないが，
あまり直感的ではないので他の可視化を試してみよう

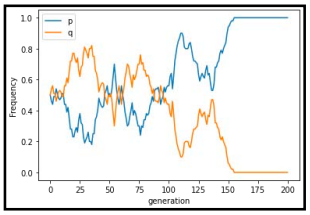
```
# 02-05. matshowによる可視化
plt.matshow( a_list, interpolation=None,
             vmin=0, vmax=1)
plt.ylabel("generation")
```



1世代の遺伝子型の配列を横1列に並べたセルで表現

matplotlib.pyplotモジュール
- matshow(配列ライク，オプション)
  配列ライク（ネストされたリストなど）を行列とみなして，その値で色付けしてプロットする．
  - interpolationオプションで補間をしないことを指示
  - vmin及びvmaxで値の範囲を指定

```
# 02-06. 頻度の世代を経た変化
# 頻度の計算
p_list = []
q_list = []
for a in a_list:
    p = sum(a)/len(a)
    p_list.append(p)
    q_list.append(1-p)

plt.plot(p_list, "-", q_list, "-")
plt.legend(["p", "q"])
plt.xlabel("generation")
plt.ylabel("Frequency")
```



対立遺伝子の頻度を計算してプロット

突然変異固定までの待ち時間

ライト-フィッシャー モデルを仮定し，突然変異固定までの待ち時間を計算する

```
# 02-07. 突然変異固定までの平均待ち時間を10回分について計算してみよう
```

何回分の平均を計算するか？

↓

変異個体数の初期値を設定

↓

配列の初期化・変異配列の決定

↓

ライト-フィッシャーモデルを仮定した世代交代

↓

変異の固定/消失の判定

その他
ループの継続

消失
ループのやり直し

↓ **固定**

breakして待ち時間を記録 ……▶ **10回分の待ち時間を計算し終えたらその平均を計算する**

本日の課題 ノーマル

1. サイコロの目の総和が100を超えるまでの平均待ち時間を試行数10回，100回，1000回，10000回についてそれぞれ計算せよ．
2. ランダムウォークの結果を5つ重ねてプロットせよ
3. ライト-フィッシャーモデルを仮定し，N 個体の集団内に占める突然変異対立遺伝子を持つ個体の数を k，その頻度を$p\left(=\frac{k}{N}\right)$とする．初期状態で $p=0.5$のとき，世代交代を繰り返し，集団に突然変異が固定する場合について，その100回分の平均待ち時間 T を求めよ
4. 質問，意見，要望等をどうぞ．

課題をノートブック（.ipynbファイル）にまとめて，Moodleにて提出すること
ファイル名は[回数，01~15]_[難易度，ノーマル nかハード h].ipynb．例．06_n.ipynb

次回予告
第７回：理論形態モデル
6月5日

復習推奨

- 指数増殖モデル
- 回転行列

本日の課題 ハード

1. ライト-フィッシャーモデルを仮定し，N 個体の集団内に占める突然変異対立遺伝子を持つ個体の数を k，その頻度を$p\left(=\frac{k}{N}\right)$とする．$N=100$，$N=200$の場合について，それぞれ k を $1\sim N$ まで $\frac{N}{10}$ 刻み程度で変化させ，突然変異の初期頻度 p に対する平均待ち時間 T を10個ずつプロットせよ．

2. 半数体生物に対して突然変異固定までの平均待ち時間 T の解析解が

$$T(p)=-\frac{1}{p}\left\{2N(1-p)\log_e(1-p)\right\}$$

で与えられるとき，このグラフを$N=100$，$N=200$について描き，同じグラフ上で上記のプロットと比較し，考察せよ．

課題をノートブック（.ipynbファイル）にまとめて，Moodleにて提出すること

ファイル名は[回数，01~15]_[難易度，ノーマル nかハード h].ipynb．例．06_h.ipynb

突然変異固定までの平均待ち時間

再現したい図

（ただし課題では集団サイズが異なる）

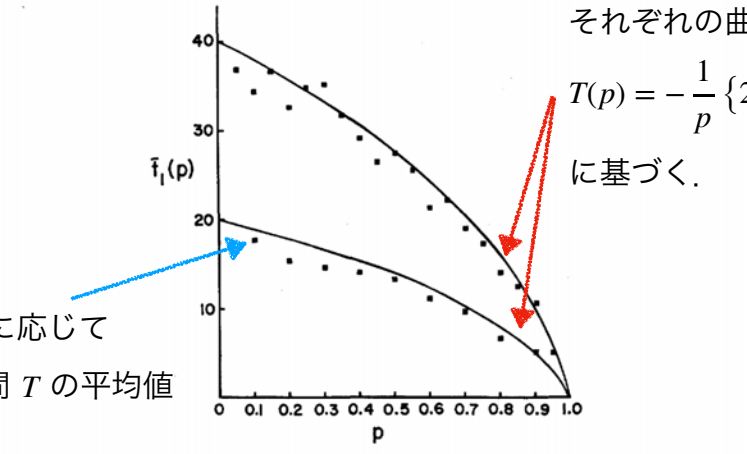


FIGURE 1.—Average number of generations until fixation of a selectively neutral mutant gene as a function of its initial frequency. The theoretical values are represented by curves and those of Monte Carlo simulation by dots. $2N_e$=20 in the upper curve and $2N_e$=10 in the lower one.

それぞれの曲線は

$$T(p)=-\frac{1}{p}\left\{2N(1-p)\log_e(1-p)\right\}$$

に基づく．

各点は p の初期値に応じて計算された待ち時間 T の平均値

Kimura & Ohta, 1969, *Genetics*

- Kimura, M., Ohta, T. (1969). The Average Number of Generations until Fixation of a Mutant Gene in a Finite Population. *Genetics* **61**(3), 763-71. https://www.ncbi.nlm.nih.gov/pubmed/17248440