数理生物学演習

第１１回　応用例１：確率過程による変異蓄積モデル

富本 創
sou.tomimoto@gmail.com
九州大学大学院システム生命科学府
数理生物学研究室

第11回：応用例(１)：
確率過程による変異蓄積モデル

本日の目標

- マルコフ連鎖
- 体細胞変異の蓄積過程

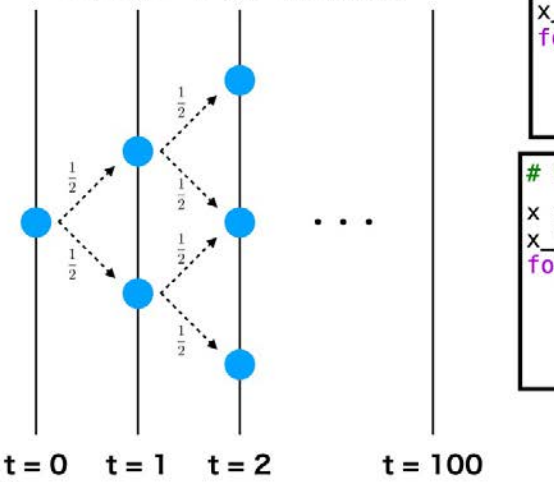
確率過程

時間の経過と共にランダムに変化してゆく量を確率論の立場から表現するものを確率過程という*。

e.g.

・ランダムウォーク（第6回の授業）

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.



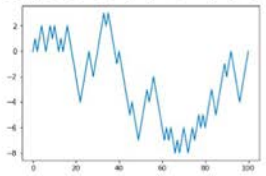
解釈例：

- 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

```
# 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



15

* 小林道正, Mathematica 確率

マルコフ連鎖

離散時間 $\{X_n; n = 0, 1, \dots\}$ で 離散的な状態空間 $S = \{X_0, X_1, \dots\}$の確率過程で
すべての$n \geq 0$, すべての$j \in S$に対しマルコフ性をみたすもの

マルコフ性（Markov property）

$$P(X_{n+1} = j \mid X_0, X_1, \dots, X_n) = P(X_{n+1} = j \mid X_n)$$

未来の状態 X_{n+1}は、現在の状態 X_n によってのみ（確率的に）決まり
それ以前の状態 $X_{n-1}, \dots, X_1$ には依存しない。

e.g. 大学生Aの晩ご飯：昨日のメニューだけ考慮し、今日が決まる

状態空間 S = {カレー, 親子丼, …}

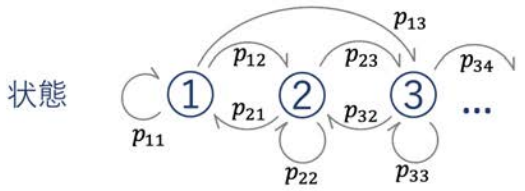
マルコフ連鎖

推移確率行列（transition probability matrix）

$$P(X_{n+1} = j \mid X_n = i\,) = p_{ij}$$

は状態 i から状態 j への推移確率を表し、すべての状態 $i\,,j$ について P_{ij} を並べてできる行列を、推移確率行列という。

$$\boldsymbol{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \cdots \\ p_{21} & p_{22} & p_{23} \cdots \\ p_{31} & p_{32} & p_{33} \cdots \\ \vdots & \vdots & \vdots \ \ddots \end{bmatrix}$$



すべての要素は非負で、行の和は必ず 1 に等しい（$\sum_j p_{kj} = 1$）

マルコフ連鎖

状態ベクトル（state vector）

マルコフ連鎖が時刻tにおいて状態 i にある期待値を $\pi_t(i)$ とし、
状態順に並べたベクトル

$$\boldsymbol{\pi_t} = (\pi_t(1), \pi_t(2), \pi_t(3), \dots)$$

は、時刻tにおける状態の確率分布を表す（$\sum_i \pi_t(i) = 1$）

初期状態を $\boldsymbol{\pi_0}$とおくと、推移確率行列 $\boldsymbol{P}$ により、時刻tにおける状態ベクトルは

$$\boldsymbol{\pi_t} = \boldsymbol{\pi_0} \boldsymbol{P}^t$$ で表される。

確率的な挙動をただの
行列の積で求められる！

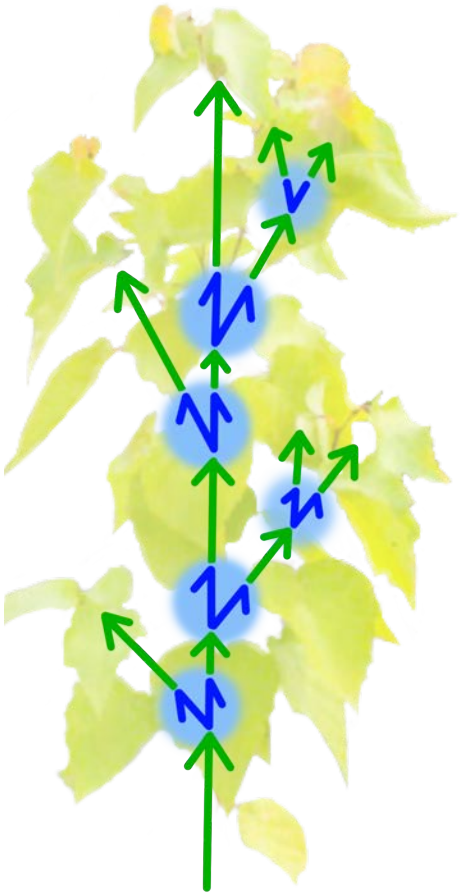
exercise1

$\boldsymbol{\pi_0} = (\pi_0(1), \pi_0(2))$, $\boldsymbol{P} = \begin{bmatrix} p_{11} p_{12} \\ p_{21} p_{22} \end{bmatrix}$ から

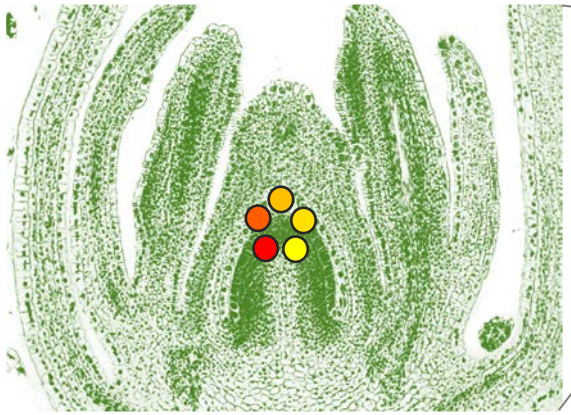
$\boldsymbol{\pi_0 P}$ を計算して $\boldsymbol{\pi_1}$ をもとめ、推移を確かめよ

樹木の成長のモデル化

樹木の分枝構造は↑**Elongation** と **Branching** の繰り返しにより形づくられる

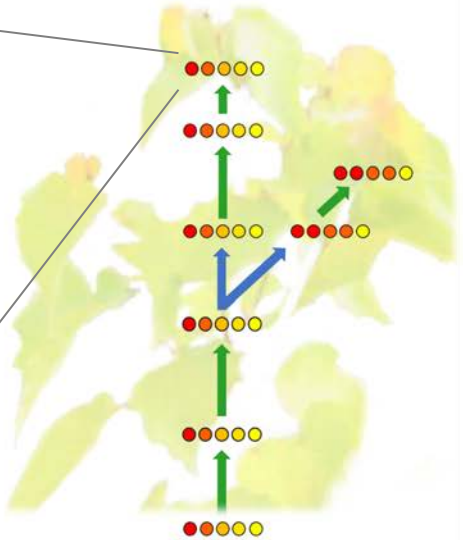


茎頂分裂組織（shoot meristem）の幹細胞に蓄積した体細胞変異のみに注目。



shoot meristem

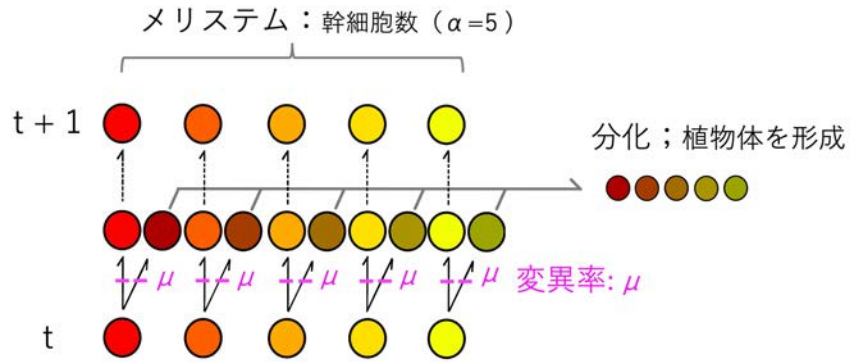
Satina et al. 1940 Amer J Bot より変更



Elongation と Branching において、
shoot meristem の幹細胞に変異が蓄積する過程を
推移確率行列で表す。

Elongation

Shoot meristem 内の幹細胞の分裂による枝の伸長



- shoot meristem にはα個の幹細胞
- α個の幹細胞が同期して分裂する
- 分裂で生じた片方の娘細胞のみ残る
- 変異は細胞分裂の際に確率μで生じる

Elongationの推移確率行列

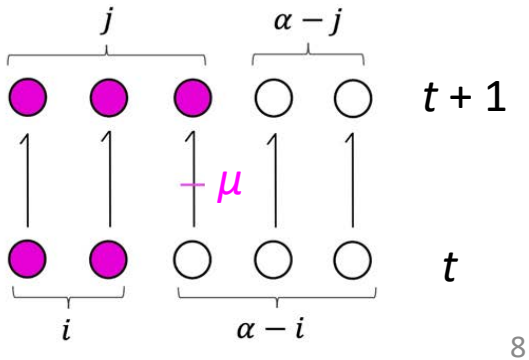
$$\boldsymbol{L} = [l_{ij}] \quad 0 \le i,j \le \alpha$$

! 注：状態は0～α

$$l_{ij} = \begin{cases} \binom{\alpha - i}{j - i} \mu^{j-i} (1-\mu)^{\alpha - j} & if\ j \ge i \\ 0 & if\ j < i \end{cases}$$

μ: mut rate per site per div

i はあるサイトに変異を持つ細胞の数。
状態 iから状態 j への推移 l_{ij} は変異をもつ幹細胞が iから j 個に増える確率



Elongation

Elongationの推移確率行列 $\boldsymbol{L} = [l_{ij}]$

$$l_{ij} = \binom{\alpha - i}{j - i} \mu^{j-i} (1-\mu)^{\alpha - j}$$ … 二項分布 $\mathrm{Bin}(\alpha - i, \mu)$ のかたち

```
import numpy as np
from scipy.stats import binom  #scipyモジュール

def stru_el_matrix(num_stem, mut_rate):
    tr_matrix = np.zeros((num_stem+1, num_stem+1))
    for i in range(num_stem+1):
        for j in range(num_stem+1):

            tr_matrix[i, j] = binom.pmf(j-i, num_stem-i, mut_rate)

    return tr_matrix
```

The probability mass function for **binom** is:

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k \in \{0, 1, \ldots, n\}, 0 \leq p \leq 1$

binom takes n and p as shape parameters, where p is the probability of a single success and $1 - p$ is the probability of a single failure.

binom.pmf(k, n, p)

```
#確認
print(stru_el_matrix(3, 0.5))
```

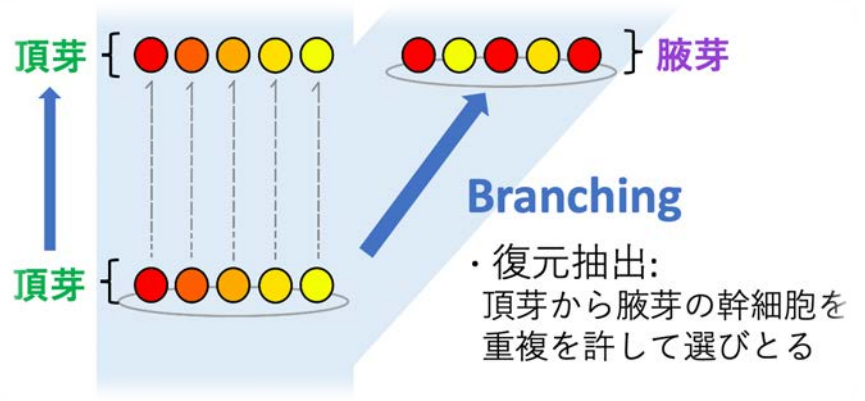
```
# 出力
[[0.125 0.375 0.375 0.125]
 [0.    0.25  0.5   0.25 ]
 [0.    0.    0.5   0.5  ]
 [0.    0.    0.    1.   ]]
```

Numpy については、第 7 回前半・第 9 回を参照

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.binom.html

Branching

頂芽の meristem から側芽の幹細胞が選び取られる過程

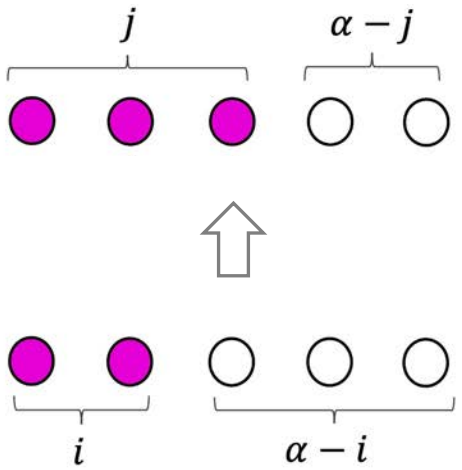


・復元抽出（選ばれ易さは比にのみ依存）
・頂芽は変化しない（***B***は腋芽への推移）
・分枝の際に生じる変異は無視

Branching の推移確率行列

$$\boldsymbol{B} = [b_{ij}] \quad 0 \le i,j \le \alpha$$

$$b_{ij} = \binom{\alpha}{j} (i/\alpha)^j (1 - i/\alpha)^{\alpha - j}$$



Branching

Branching の推移確率行列

$$\boldsymbol{B} = [b_{ij}] \qquad b_{ij} = \binom{\alpha}{j} (i/\alpha)^j (1 - i/\alpha)^{\alpha - j}$$

```
def br_matrix(num_stem):
    tr_matrix = np.zeros((num_stem+1, num_stem+1))
    for i in range(num_stem+1):
        for j in range(num_stem+1):

            tr_matrix[i, j] = binom.pmf(j, num_stem, i/num_stem)

    return tr_matrix
```

二項分布

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$ binom.pmf(k, n, p)

```
#確認
print(br_matrix(3))
```

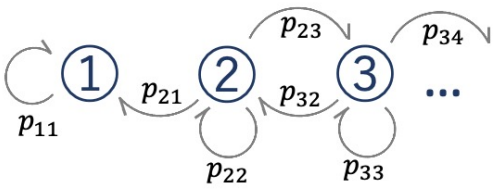
```
# 出力 (省略してます)
[[1.    0.    0.    0.   ]
 [0.296 0.444 0.222 0.037]
 [0.037 0.222 0.444 0.296]
 [0.    0.    0.    1.   ]]
```

境界条件

$l_{00} < 1$; 反射壁

$l_{\alpha\alpha} = 1$; 吸収壁

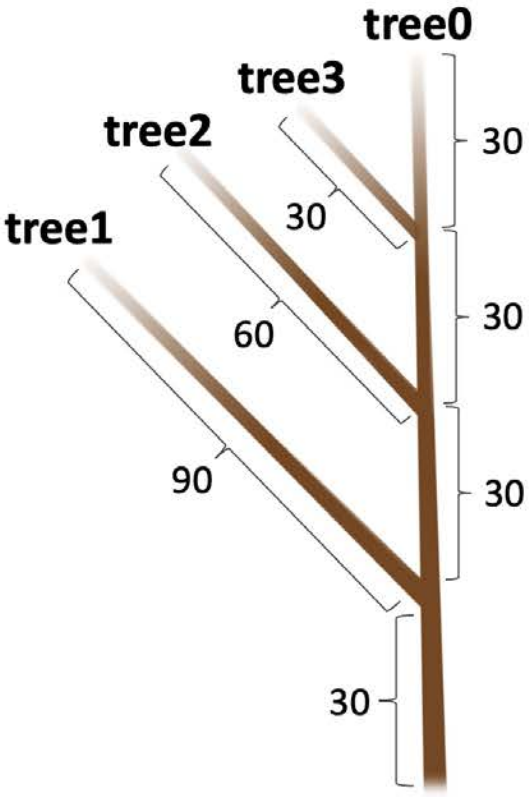
$b_{00}, b_{\alpha\alpha} = 1$; 吸収壁



樹木の成長のモデル化

ElongationとBranchingの推移確率行列を組み合わせて
樹木の成長に伴う体細胞変異の蓄積を表す

シンプルな分枝構造の樹形で表してみよう！



下準備

```
stemCells = 5 #幹細胞数＝ 5
mutRate = 10**(-2) #変異率

#推移確率行列
struElMat = stru_el_matrix(stemCells, mutRate)
```

```
#初期状態
stateVector = np.zeros(stemCells+1, dtype=float)
stateVector[0] = 1

print(stateVector) #確認
```

```
#出力  [1. 0. 0. 0. 0. 0.]
```

はじめの状態では変異を持つ幹細胞なし

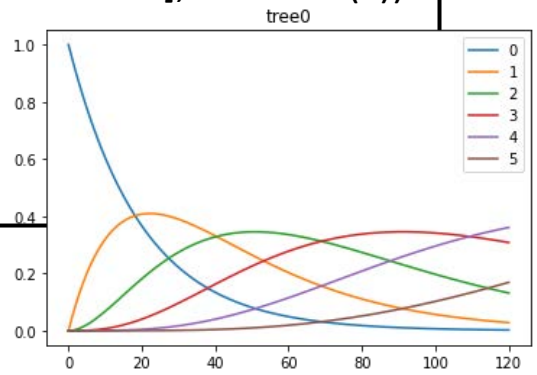
$$\boldsymbol{\pi_0 = (1, 0, 0, 0, 0, 0)}$$

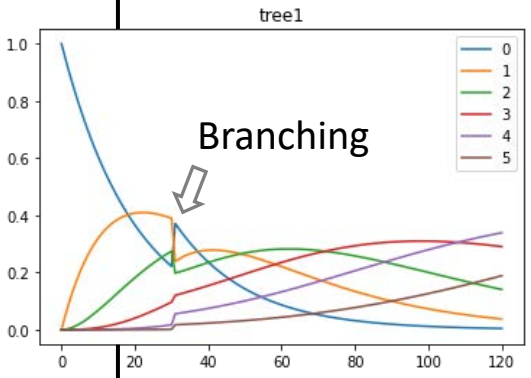
樹木の成長のモデル化

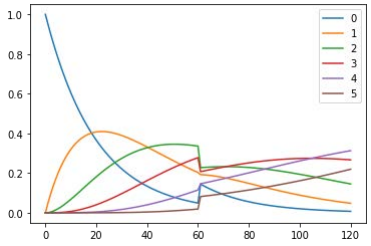
```
#tree0
tree0 = [np.copy(stateVector)] #記録用のリスト
st_V = np.copy(stateVector)

for i in range(120): #elingation だけ
    st_V = np.dot(st_V, struElMat) #内積
    tree0.append(st_V)
```

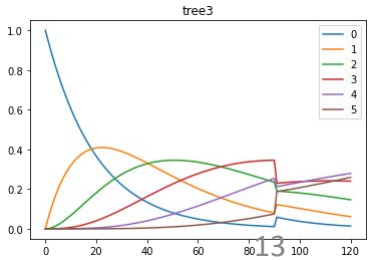
```
#tree1
tree1 = [np.copy(stateVector)]
st_V = np.copy(stateVector) #リセット

for i in range(30): #elingation1
    st_V = np.dot(st_V,struElMat)
    tree1.append(st_V)

st_V = np.dot(st_V, brMat) #branching
tree1.append(st_V)

for i in range(90-1): #elingation2
    st_V = np.dot(st_V,struElMat)
    tree1.append(st_V)
```

```
import matplotlib.pyplot as plt

#plot
for k in range(stemCells+1):
    plt.plot([i[k] for i in tree0], label=str(k))

plt.legend()
plt.title('tree0')
plt.show()
```









tree2, tree3も同様

Simulation model：マルチサイトへの拡張

マルコフ連鎖では、ある１サイトで起こる変異にのみ注目し、
その変異が幹細胞の集団内に占める期待値だけを表していた。

↓

シミュレーションモデルにより、複数のサイトに変異が蓄積する過程を
確率的なゆらぎを伴って表すことのできるモデルを考える

enumetrate()関数

シーケンス＋ループカウンタ

```
#リストをイテラブルとしたforループ

a_list = ['a','b','c']

count = 0  #ループカウンタ

for i in a_list:
    print(count, i)
    count += 1
```

```
# 出力
0 a
1 b
2 c
```

```
#range()をイテラブルとしたforループ

for i in range(len(a_list)):
    print(i, a_list[i])
```

```
# 出力
0 a
1 b
2 c
```

→　複雑になってしまう

enumerate()関数とは

for ループカウンタ, イテレータ in **enumerate**(リスト):
　　文１…
　　文２…

リストをrange()の様に使うことができる！

```
#enumerate()関数

a_list = ['a','b','c']

for icut, i in enumerate(a_list):
    print(icut,i)
```

icut #ループカウンタ
i　　#イテレータ

```
# 出力
0 a
1 b
2 c
```

リスト内包表記

for ループを使わずにリストをつくる（復習）

```
# forループによるリストの生成

b_List = []
for i in range(10):
    b_List.append(i)

print(b_List)
```

#出力 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
#リスト内包表記

c_List = [i for i in range(10)]

print(c_List)
```

#出力 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

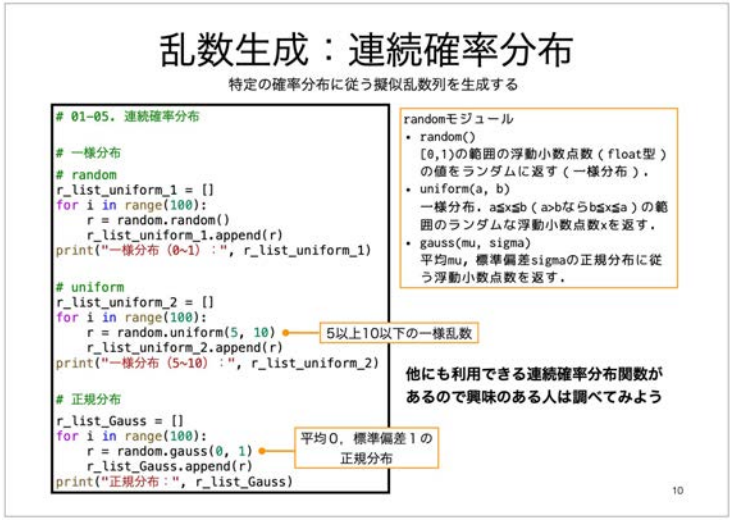
リスト内包表記のメリット
・一行だけ
・早い

```
# おまけ
#リスト内包表記で if文

d_List = [i for i in range(10) if i%2==0]
print(d_List)
```

#出力 [0, 2, 4, 6, 8]

乱数生成

random モジュール（第 6 回の復習）



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)
```

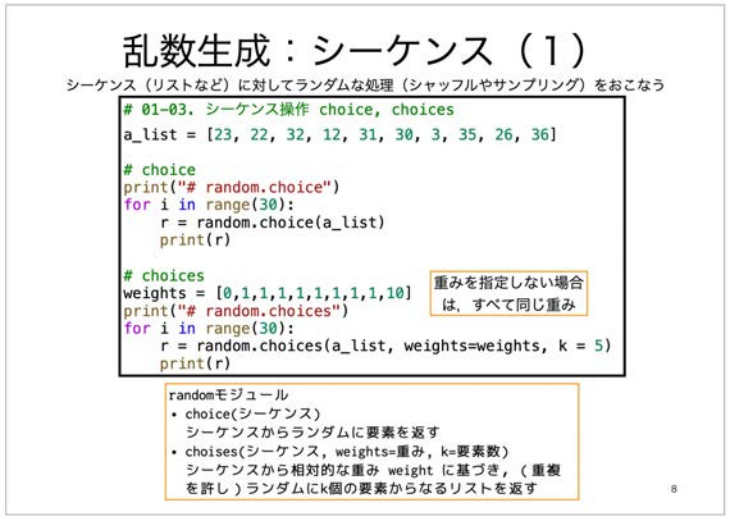
randomモジュール
- random()
  [0,1)の範囲の浮動小数点数（float型）の値をランダムに返す（一様分布）.
- uniform(a, b)
  一様分布．a≦x≦b（a>bならb≦x≦a）の範囲のランダムな浮動小数点数xを返す．
- gauss(mu, sigma)
  平均mu，標準偏差sigmaの正規分布に従う浮動小数点数を返す．

5以上10以下の一様乱数

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

平均0，標準偏差1の正規分布

10



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

8

#randomモジュールのimport
import random

#乱数生成
random.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)

#復元抽出
random.choices(list, k = num)

listから、重複を許してnum 個選びリストとして返す

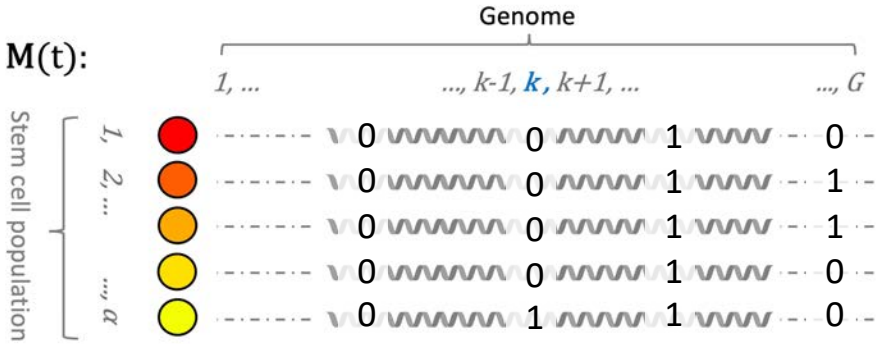
ゲノムの状態を行列で表す

変異行列

$$\mathbf{M}(t) = [m_{ij}(t)]$$

$$m_{ij}(t) = \begin{cases} 0 \ldots & \text{non-mutated} \\ 1 \ldots & \text{mutated} \end{cases}$$

Stem cells : $i = 1, 2, \ldots, \alpha$ (number of stem cells)
Genome sequence : $j = 1, 2, \ldots, G$ (genome size)




```
import numpy as np

#ndarray で genome matrixをつくる
genomeSize = 10**3
stemCells = 5

#genome matrix
M_t = np.zeros((stemCells,
genomeSize))

print(M_t)
```

#出力

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

＃おまけ
省略された形でprintされる。
全て確認したい時は

```
np.set_printoptions(threshold=np.inf)
print(M_t)
```

thresholdで範囲を指定できる

Simulation model

```
# Elongation
def stru_elongation(m_t, mut_rate, num_time):
    m_tList = [] #記録用リスト
    m_t1 = np.copy(m_t)

    for num in range(num_time): #elongationの繰り返し
        m_t2 = np.copy(m_t1)
        #DNAの複製
        for icut, i in enumerate(m_t1): #幹細胞 i
            for jcut, j in enumerate(i): #ゲノム j
                if random.random() <= mut_rate: #変異が起きる
                    m_t2[icut][jcut] = 1

        m_tList.append(m_t2)
        m_t1 = m_t2

    return m_tList
```

m_t が変化しない様に copy する（第 6 回参照）

仮定：Back mutation は起こらない

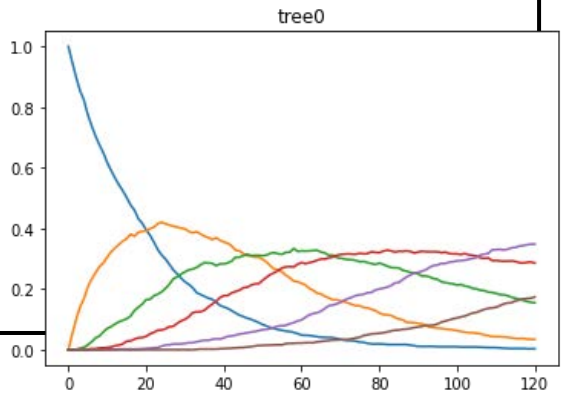
変異行列の時間変化をリストとして返す

```
# Branching
def branching(m_t, num_stem):
    index_list = random.choices(np.arange(len(m_t)), k=num_stem) # len(m_t)=num_stem

    return np.array([m_t[i] for i in index_list]) # with replacement
```

リストの要素ではなくindexを
復元抽出により重複を許して選びとる。

Simulation modelの確認

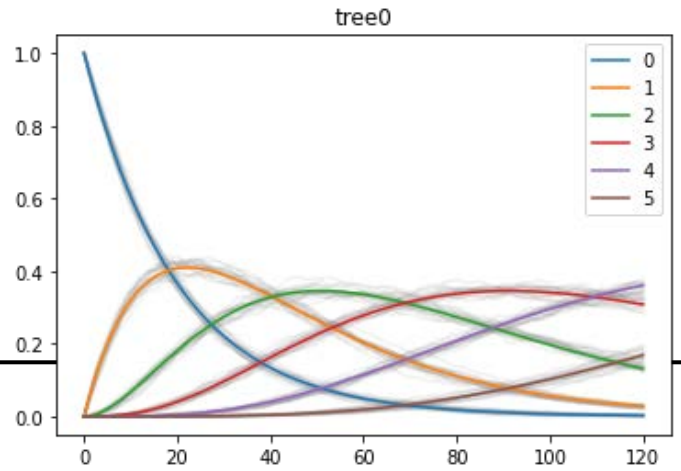
```
#simulation tree0
sim_tree0 = [np.copy(M_t)] + stru_elongation(M_t, mutRate, 120) #リストの連結

#変異行列から状態ベクトルを計算
simuState = [[0 for i in range(stemCells+1)] for j in range(120+1)]

for icut, i in enumerate(sim_tree0):
   for jcut,j in enumerate(i.T): #transpose
      simuState[icut][int(sum(j))] += 1 #int(); indexがfloatになるのを防ぐ

simuStateVector = np.array(simuState)/genomeSize #サイトで平均化
plt.plot(simuStateVector)
plt.title('tree0')
plt.show()
```



マルコフ連鎖のモデルの結果を期待値で
はなく、確率的なゆらぎを持って表せる

Simulation modelの確認

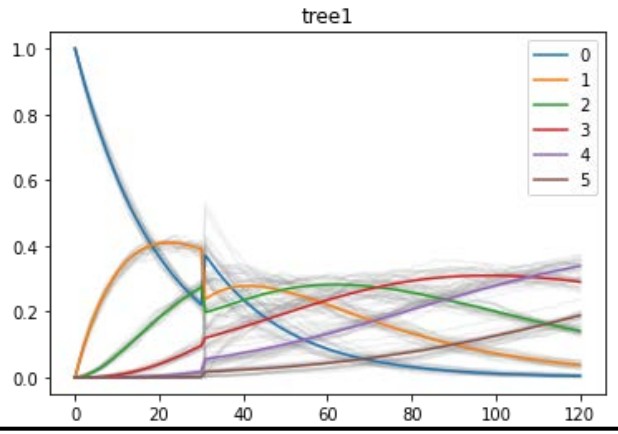
```
#マルコフ連鎖のモデルによるsimulationの確認
#tree0

#simulation
for k in range(20): #繰り返し
   simuState = [[0 for i in range(stemCells+1)] for j in range(120+1)]

   sim_tree0 = [np.copy(M_t)] + stru_elongation(M_t, mutRate, 120) #リストの連結

   for icut, i in enumerate(sim_tree0):
      for jcut,j in enumerate(i.T): #transpose
         simuState[icut][int(sum(j))] += 1 #int(); indexがfloatになるのを防ぐ

   simuStateVector = np.array(simuState)/genomeSize
   plt.plot(simuStateVector, alpha=1/10,c='gray') #alpha; 不透明度

#math model
for k in range(stemCells+1):
   plt.plot([i[k] for i in tree0],label=str(k))

plt.legend()
plt.title('tree0')
plt.show()
```



Simulation modelの確認

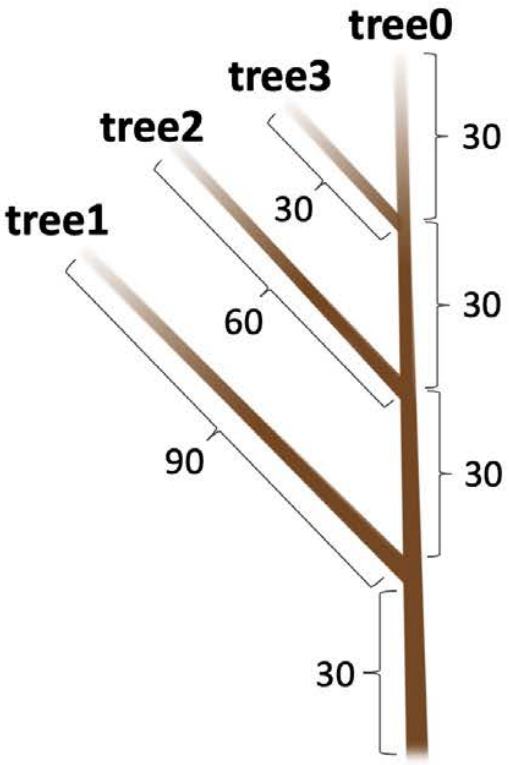
tree2, tree3も同様に確認できる

```
#tree1

#simulation
for k in range(20): #繰り返し
   simuState = [[0 for i in range(stemCells+1)] for j in range(120+1)]

   #simulation
   sim_tree1 = [np.copy(M_t)]
   sim_tree1_el1 = stru_elongation(M_t, mutRate, 30) #type: list
   sim_tree1_br = branching(sim_tree1_el1[-1], stemCells) #type: ndarray
   sim_tree1_el2 = stru_elongation(sim_tree1_br, mutRate, 90-1) #type: list

   sim_tree1 = sim_tree1 + sim_tree1_el1
   sim_tree1.append(sim_tree1_br)
   sim_tree1 = sim_tree1 + sim_tree1_el2

   for icut, i in enumerate(sim_tree1):
      for jcut,j in enumerate(i.T): #transpose
         simuState[icut][int(sum(j))] += 1 #int(); indexがfloatになるのを防ぐ

   simuStateVector = np.array(simuState)/genomeSize
   plt.plot(simuStateVector, alpha=1/10,c='gray')

#math model
for k in range(stemCells+1):
  plt.plot([i[k] for i in tree1],label=str(k))

plt.legend()
plt.title('tree1')
plt.show()
```



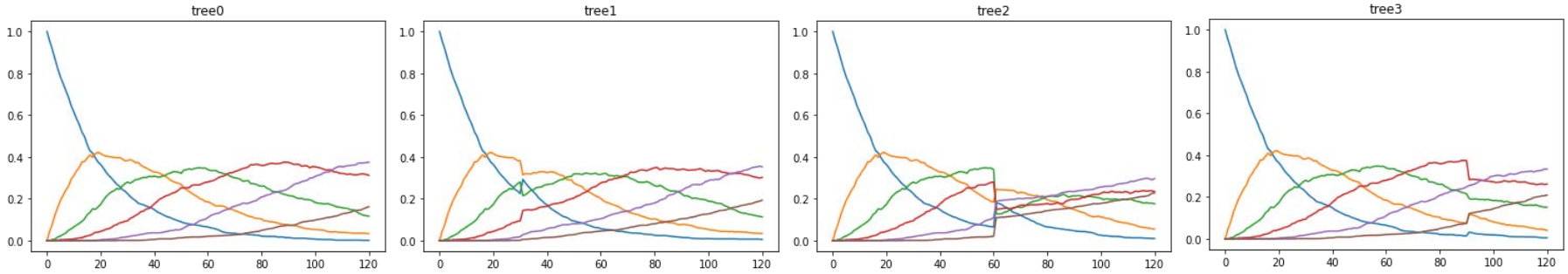
Simulation：樹木の成長のモデル化

```
#成長のモデル化
#tree0
sim_tree0_el1 = stru_elongation(M_t, mutRate, 30)
sim_tree0_el2 = stru_elongation(sim_tree0_el1[-1], mutRate, 30)
sim_tree0_el3 = stru_elongation(sim_tree0_el2[-1], mutRate, 30)
sim_tree0_el4 = stru_elongation(sim_tree0_el3[-1], mutRate, 30)

#tree1
sim_tree1_br = branching(sim_tree0_el1[-1], stemCells)
sim_tree1_el2 = stru_elongation(sim_tree1_br, mutRate, 90-1)

#tree2
sim_tree2_br = branching(sim_tree0_el2[-1], stemCells)
sim_tree2_el2 = stru_elongation(sim_tree2_br, mutRate, 60-1)

#tree3
sim_tree3_br = branching(sim_tree0_el3[-1], stemCells)
sim_tree3_el2 = stru_elongation(sim_tree3_br, mutRate, 30-1)

#まとめ
sim_tree0 = [np.copy(M_t)] + sim_tree0_el1 + sim_tree0_el2 + sim_tree0_el3 + sim_tree0_el4
sim_tree1 = [np.copy(M_t)] + sim_tree0_el1 + [sim_tree1_br] + sim_tree1_el2
sim_tree2 = [np.copy(M_t)] + sim_tree0_el1 + sim_tree0_el2 + [sim_tree2_br] + sim_tree2_el2
sim_tree3 = [np.copy(M_t)] + sim_tree0_el1 + sim_tree0_el2 + sim_tree0_el3 + [sim_tree3_br] + sim_tree3_el2
```



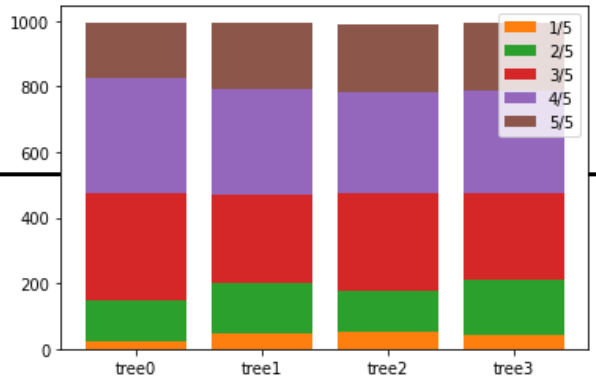
Simulation：樹木の成長のモデル化

```
#simulation plot
for kcut,k in enumerate([sim_tree0, sim_tree1, sim_tree2, sim_tree3]):
    simuState = [[0 for i in range(stemCells+1)] for j in range(120+1)]

    for icut, i in enumerate(k):
        for jcut,j in enumerate(i.T): #transpose
            simuState[icut][int(sum(j))] += 1 #int(); indexがfloatになるのを防ぐ

    simuStateVector = np.array(simuState)/genomeSize
    plt.plot(simuStateVector) #alpha; 不透明度
    plt.title('tree'+str(kcut))
    plt.show()
```



Simulation：枝末端に蓄積した変異

```
#枝末端に蓄積した変異に注目

#データ整理
endMut = [[0 for i in range(stemCells+1)] for i in range(4)] #枝末端数＝4

for kcut,k in enumerate([sim_tree0, sim_tree1, sim_tree2, sim_tree3]):
   for jcut,j in enumerate(k[-1].T): #transpose
      endMut[kcut][int(sum(j))] += 1 #int(); indexがfloatになるのを防ぐ

#plot
treeName = ['tree'+str(i) for i in range(4)]
bottom_data = [0 for i in range(4)] #barグラフの積み上げ用
cmap = plt.get_cmap("tab10") #グラフの色調整

for icut,i in enumerate(np.array(endMut).T[1:]): #「変異なし」を除く
   plt.bar(treeName,i,bottom=bottom_data,label=str(icut+1)+'/'+str(stemCells), color=cmap(icut+1))
      bottom_data += i

plt.legend() #(bbox_to_anchor=(1.01,1),loc='upper left')
plt.show()
```



変異率が高いので、あまり差が見られない。

本日の課題

ノーマル

1. 変異率を小さくした場合（mutRate = 10^(-4)前後）について、枝末端に蓄積した変異をplotせよ. さらにシミュレーションを20回ほど繰り返して平均をもとめ、その結果（枝末端に蓄積した変異）をplotせよ.

ハード（2, 3 のいずれか一方）

2. ノーマル課題 1.について、３つの異なる分枝構造の下で同様の実装を行い樹形が変異の蓄積に与える影響を考察せよ.

3. 後記の参考にあるElongation(2)のモデルについて、
   マルコフ連鎖とsimulationを合わせてplotせよ（p21,22を参考）
   さらに枝末端に蓄積した変異をplotし、講義で扱ったモデルと比較せよ.

参考：Elongation (2)

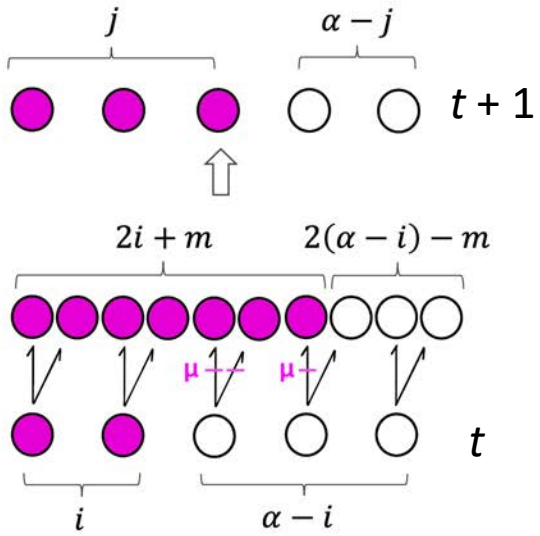
推移確率行列；Elongation

$$\boldsymbol{L} = [l_{ij}]$$

超幾何分布

$$l_{ij} = \begin{cases} \sum_{m=0}^{2(\alpha-i)} a_{2(\alpha-i),m} \dfrac{\binom{2i+m}{j}\binom{2(\alpha-i)-m}{\alpha-j}}{\binom{2\alpha}{\alpha}} & if\ j \geq i \\ 0 & if\ j < i \end{cases}$$

ただし　$a_{2(\alpha-i),m} = \binom{2(\alpha-i)}{m} \mu^m (1-\mu)^{2(\alpha-j)-m}$



μ: mut rate per site per div

参考：Elongation (2)

$$l_{ij} = \sum_{m=0}^{2(\alpha-i)} a_{2(\alpha-i),m} \frac{\binom{2i+m}{j}\binom{2(\alpha-i)-m}{\alpha-j}}{\binom{2\alpha}{\alpha}}$$

ただし $a_{2(\alpha-i),m} = \binom{2(\alpha-i)}{m} \mu^m (1-\mu)^{2(\alpha-j)-m}$

```
from scipy.stats import hypergeom

def stoc_el_matrix(num_stem, mut_rate):
    tr_matrix = np.zeros((num_stem+1, num_stem+1))
    for i in range(num_stem+1):
        for j in range(num_stem+1):

            tr_matrix[i, j] = sum([ binom.pmf(m, 2*(num_stem-i), mut_rate)*
                                    hypergeom.pmf(j, 2*num_stem, 2*i+m, num_stem)
                                      for m in range(2*(num_stem-i)+1)])

    return tr_matrix
```

参考：simulation Elongation (2)

```
def stoc_elongation(m_t, mut_rate, num_time):
    m_tList = []
    m_t1 = np.copy(m_t)

    for num in range(num_time):
        m_t2_1 = np.copy(m_t1)
        m_t2_2 = np.copy(m_t1)

        #DNAの複製
        for icut, i in enumerate(m_t1):
            for jcut, j in enumerate(i):
                if np.random.rand() <= mut_rate:
                    m_t2_1[icut][jcut] = 1
                if np.random.rand() <= mut_rate:
                    m_t2_2[icut][jcut] = 1

        #random selection
        index_list = np.random.choice(2*len(m_t), len(m_t), replace=False)
        m_t2 = np.array([np.vstack((m_t2_1, m_t2_2))[i] for i in index_list])

        m_tList.append(m_t2)
        m_t1 = m_t2

    return m_tList
```