Q matrixを用いたMarkov過程のシミュレーション

Keiji Imoto
Department of Information Physiology
National Institute for Physiological Sciences
Okazaki, Japan

07-Dec-1997, revised 22-Feb-2011

1　はじめに

イオンチャネルには、閉じた状態と開いた状態という少なくとも 2 つの状態がある。しかし多くのチャネルではたとえば不活性化状態などのように 2 つ以上の状態が存在することが考えられる。いくつかの状態を仮定しそれらの状態間の遷移率から予測されるチャネルの挙動を、実際の実験で得られたデータと比較し、仮定をより確かなものにして行くというシミュレーションの手法は、チャネルの機能を理解する上で重要な地位を占めてきた。この手法を実際に用いるにはコンピューターが不可欠である。計算速度の驚異的な発達により、パーソナルコンピューターでもチャネル開閉のシミュレーションを容易に行えるようになった。

シミュレーションの基本的な仮定は、ある状態から他の状態への遷移 transition（推移ともいう）は、過去にどのような経路でその状態に至ったかに依存せず、また遷移率が時刻に無関係に決まっていることである。このような確率過程は”時間的に一様なマルコフ過程 homogeneous Markov process”と呼ばれる。

$$
\begin{array}{ccccccc}
 & 3\alpha_m & & 2\alpha_m & & \alpha_m & \\
\mathbf{C}_0 & \rightleftarrows & \mathbf{C}_1 & \rightleftarrows & \mathbf{C}_2 & \rightleftarrows & \mathbf{O} \\
 & \beta_m & & 2\beta_m & & 3\beta_m & \\
\alpha_h \uparrow\downarrow \beta_h & & \alpha_h \uparrow\downarrow \beta_h & & \alpha_h \uparrow\downarrow \beta_h & & \alpha_h \uparrow\downarrow \beta_h \\
 & 3\alpha_m & & 2\alpha_m & & \alpha_m & \\
\mathbf{I}_0 & \rightleftarrows & \mathbf{I}_1 & \rightleftarrows & \mathbf{I}_2 & \rightleftarrows & \mathbf{I}_3 \\
 & \beta_m & & 2\beta_m & & 3\beta_m &
\end{array}
$$

上の図のようなチャネルの開閉のモデルを論文で見かけることは少なくない。いくつかのチャネルが閉じた状態 $\mathbf{C}_i$ とチャネルが開いた状態 $\mathbf{O}$ および不活性化した状態 $\mathbf{I}_i$ を想定したチャネルの開閉を、実際にシミュレーションを行って解析しようする場合、どうしても自分でプログラムを書かなくてはならない。その時の助けとなるように、いろいろな論文・著書から必要と思われることを抜粋して集めた。実用性を重視して数学的な証明は省いた。Q-matrix を用いる方法は、Colquhoun D & Hawkes AG [文献 1] に記載されているものである.

2 指数関数

まず単純な系を考える。一定の割合で減少していく現象は、微分方程式で

$$dy(t)/dt = -\lambda y$$

と表わされる。この微分方程式の一般解は、指数関数

$$y(t) = a_0 e^{-\lambda t}$$

である。λ の逆数 $1/\lambda$ は時定数 τ である。τ は $y(t+\tau)/y(t) = 1/e$ となる時間であり、また τ はその状態にある平均時間である。十分な時間が経ったとき、状態は平衡状態になる事が予想される。すなわち $dy(t)/dt$ は 0 に近づき、$y(t)$ も 0 に近づいて行く。

3 Two-state model

2 つの状態 1 と 2 を考え、それぞれの状態にある確率を p_1 と p_2 とする。p_1 と p_2 は時刻 t の関数であり、$p_1(t)$ と $p_2(t)$ と表わされる。$p_1(t)$ と $p_2(t)$ は確率であるから、

$$p_1(t) + p_2(t) = 1$$

が常に成り立つ。

いま状態 1 から状態 2 への遷移率を α とし、状態 2 から状態 1 への遷移率を β とする。

$$state\ 1 \underset{\beta}{\overset{\alpha}{\rightleftarrows}} state\ 2$$

$p_1(t)$ と $p_2(t)$ を表わす微分方程式は、

$$\begin{aligned} \frac{dp_1(t)}{dt} &= -\alpha p_1(t) + \beta p_2(t) \\ \frac{dp_2(t)}{dt} &= \alpha p_1(t) - \beta p_2(t) \end{aligned}$$

となる。上の指数関数は、two-state model のなかの $\alpha = \lambda$、$\beta = 0$ という特殊な例である。十分な時間が経ったとき平衡状態に近づくから、

$$\begin{aligned} \frac{dp_1(\infty)}{dt} &= -\alpha p_1(\infty) + \beta p_2(\infty) = 0 \\ \frac{dp_2(\infty)}{dt} &= \alpha p_1(\infty) - \beta p_2(\infty) = 0 \end{aligned}$$

となる。ここで $p_1(\infty) + p_2(\infty) = 1$ であるから、$p_1(t)$ と $p_2(t)$ はそれぞれ、

$$\begin{aligned} p_1(\infty) &= \frac{\beta}{\alpha+\beta} \\ p_2(\infty) &= \frac{\alpha}{\alpha+\beta} \end{aligned}$$

に近づいて行く。$p_1(\infty)$ と $p_2(\infty)$ の値は初期値 $p_1(0)$、$p_2(0)$ に依存しない。

また微分方程式を解くと、

$$\begin{aligned} p_1(t) &= (p_1(0) - \frac{\beta}{\alpha+\beta})e^{-(\alpha+\beta)t} + \frac{\beta}{\alpha+\beta} \\ p_2(t) &= (p_2(0) - \frac{\alpha}{\alpha+\beta})e^{-(\alpha+\beta)t} + \frac{\alpha}{\alpha+\beta} \end{aligned}$$

となる。状態の数が小さいときは、このように解析的に微分方程式を容易に解くことができるが、状態の数が多くなってくると、微分方程式を解析的に解くことは困難であるし、直感的にも把握しにくい。

4 n-state model

n 個の状態がある場合に拡大して考える。それぞれの状態にある確率を $p_i(t)(i = 1, \ldots, n)$ とする。$p_i(t)$ は確率であるから

$$\sum_{k=1}^{n} p_k(t) = 1$$

が常に成り立つ。状態 i から状態 j へ遷移率 transition rate を $q_{ij}(i \neq j)$ とする。また $q_{ii} = -\sum_{k \neq i} q_{ik}$ とする。この値は状態 i から他の状態へ出て行く遷移率である。それぞれの状態の微分方程式は、

$$\frac{d}{dt}p_i(t) = q_{1i}p_1(t) + q_{2i}p_2(t) + \ldots + q_{ni}p_n(t)$$

あるいは、簡略に

$$\frac{d}{dt}p_i(t) = \sum_{k=1}^{n} q_{ki}p_i(t)$$

と表わされる。いま状態を表す行ベクトル row vector $\mathbf{P}$ を、

$$\mathbf{P}(t) = (p_1(t)\ p_2(t)\ \ldots\ p_n(t))$$

とし、また遷移率行列 transition rate matrix Q を、

$$Q = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1n} \\ q_{21} & q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1} & q_{n2} & \cdots & q_{nn} \end{pmatrix}$$

と定義する。行列 Q は、(q_{ij}) と略記されることもある。"時間的に一様なマルコフ過程" では行列 Q の各要素は定数であり時刻に依存しない。ベクトルや行列の微分はそれぞれの要素の微分と定義されているから、

$$\frac{d}{dt}\mathbf{P}(t) = \left(\frac{d}{dt}p_1(t)\ \frac{d}{dt}p_2(t)\ \ldots\ \frac{d}{dt}p_n(t)\right)$$

である。したがって状態の微分方程式は、

$$\begin{aligned} \frac{d}{dt}\mathbf{P}(t) &= (p_1(t)\, p_2(t)\, \ldots\, p_n(t)) \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1n} \\ q_{21} & q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1} & q_{n2} & \cdots & q_{nn} \end{pmatrix} \\ &= \mathbf{P}(t)Q \end{aligned}$$

すなわち、

$$\frac{d}{dt}\mathbf{P}(t) = \mathbf{P}(t)Q$$

と簡略に記述することができる。この微分方程式を解くことにより状態 $\mathbf{P}(t)$ の時間経過を知ることが出来る。

5 n-state modelの数値予測

まず単純な数値予測の方法を考える。非常に短い時間 Δt の間に $p_i(t)$ が $p_i(t+\Delta t)$ に変化するとき、$p_i(t+\Delta t)$ は、

$$p_i(t+\Delta t) = p_i(t) + \left(\sum_{k=1}^{n} p_k(t)q_{ki}\right)\Delta t$$

と表わされる。これを行列を用いて表わすと、

$$\begin{aligned}\mathbf{P}(t+\Delta t) &= \mathbf{P}(t) + \mathbf{P}(t)Q\Delta t \\ &= \mathbf{P}(t)\left(I_n + Q\Delta t\right)\end{aligned}$$

I_n は単位マトリクス unit matrix である。行列 $I_n + Q\Delta t$ は遷移確率行列 transition probability matrix とも呼ばれる。ここでは $\Pi = (\pi_{ij})$ と記すことにする。（遷移確率行列は、$P = (p_{ij})$ と表されることが多いが、混乱を避けるために付録では Π と表記している）$I_n + Q\Delta t$ を具体的に表わすと、

$$\begin{pmatrix} 1-\sum_{k\neq 1} q_{k1}\Delta t & q_{12}\Delta t & \dots & q_{1n}\Delta t \\ q_{21}\Delta t & 1-\sum_{k\neq 2} q_{k2}\Delta t & \dots & q_{2n}\Delta t \\ \vdots & \vdots & \ddots & \vdots \\ q_{n1}\Delta t & q_{n2}\Delta t & \dots & 1-\sum_{k\neq n} q_{kn}\Delta t \end{pmatrix}$$

であり、この行列のいずれの要素も定数である。

$$\mathbf{P}(t) = \mathbf{P}(0)\underbrace{(I+Q\Delta t)(I+Q\Delta t)\dots(I+Q\Delta t)}_{t/\Delta t \text{ 回掛け合わせる}}$$

と近似できるから、時刻 t における状態 $\mathbf{P}(t)$ は、初期条件 $\mathbf{P}(0)$ が与えられれば、

$$\mathbf{P}(t) = \mathbf{P}(0)(I+Q\Delta t)^{t/\Delta t}$$

と計算できる。計算はコンピューターにやらせるとしても、この方法では計算時間を要するだけでなく誤差がどの程度出ているか不確かである。

6 n-state modelの数値解

行列 Q の微分方程式 $d\mathbf{P}(t)/dt = \mathbf{P}(t)Q$ の解は、$\mathbf{P}(t) = \mathbf{P}(0)e^{Qt}$ となる。e^{Qt} は、

$$e^{Qt} = I + \frac{Qt}{1!} + \frac{Q^2t^2}{2!} + \dots + \frac{Q^kt^k}{k!} + \dots$$

と定義されている。したがって、時刻 t における状態 $\mathbf{P}(t)$ を知るには、$\mathbf{P}(t) = \mathbf{P}(0)e^{Qt}$ を計算すればよい。展開した無限級数をそのまま計算するのも一つの方法であるが、計算精度・計算速度の観点から好ましい方法ではない。そのため行列の性質を利用した計算方法がいろいろ工夫されている。その一つは固有値 eigenvalue と固有ベクトル eigenvector を用いる方法である。

ここで扱っている $n \times n$ の正方遷移率行列の場合、行列 Q は特異マトリクス singular matrix である。すなわち n 個の実数の固有値を持ち、そのうちの一つは 0 である。固有値が負の値にならないようにという理由だけのために、今後の計算では行列 $-Q$ を用いる。

いま行列 $-Q$ の固有値を λ_m $(m = 1,\ 2,\ \ldots,\ n)$ とし, λ_m に対する固有ベクトルを $\mathbf{X}_m$ とする。すなわち $Q\mathbf{X}_m = \lambda_m \mathbf{X}_m$ である。$\mathbf{X}_m$ は列ベクトル column vector であり、

$$\mathbf{X}_m = \begin{pmatrix} X_1^{(m)} \\ X_2^{(m)} \\ \vdots \\ X_n^{(m)} \end{pmatrix}$$

とあらわされる。いま n 個の固有ベクトルを横に並べてできる行列を X とする。

$$X = (\mathbf{X}_1\ \mathbf{X}_2\ \ldots\ \mathbf{X}_n)$$

行列 X はモードマトリクスと呼ばれる。また、λ_m を対角要素に持つ対角行列を $-\Lambda$ とする。このとき

$$\begin{aligned} e^{Qt} &= X e^{-\Lambda t} X^{-1} \\ &= X \begin{pmatrix} e^{-\lambda_1 t} & 0 & \ldots & 0 \\ 0 & e^{-\lambda_2 t} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & e^{-\lambda_n t} \end{pmatrix} X^{-1} \end{aligned}$$

とあらわされる。X^{-1} は X の逆行列である。$X^{-1} = Y = (y_{mi})$、とし、

$$w_{mi} = \sum_{k=1}^{n} p_k(0) x_k^{(m)} y_{mi}$$

とすると、

$$p_i(t) = \sum_{m=1}^{n} w_{mi} e^{-\lambda_m t}$$

となる。すなわちり $p_i(t)$ は指数関数の和で表わされる。行列 $-Q$ の n 個の固有値のうち、一つは 0 であるから、$\lambda_1 = 0,\ p_i(\infty) = w_{1i}$ の場合、

$$p_i(t) = p_i(\infty) + \sum_{m=2}^{n} w_{mi} e^{-\lambda_m t}$$

となり、$p_i(t)$ は $p_i(\infty)$ へと収束する。固有値 λ_m に対して、$a_{ki}^{(m)} = x_k^{(m)} y_{mi}$ を要素とする行列を帯行列 spectral matrix A_m と呼ぶ。

7　数値計算の方針

$p_i(t)$ を計算するには、次の順序で計算を行う。w_{mi} をいったん求めると、時刻 t に関係なく $p_i(t)$ の計算に用いることが出来る。

1. 遷移率行列 $-Q$ の固有値 λ_m と固有ベクトル $\mathbf{X}_m$ を求める。
2. 固有ベクトル $\mathbf{X}_m$ から構成されるモードマトリクス X の逆行列 Y を求める。
3. 行列 X、行列 Y より帯行列 A_m を計算する。
4. 初期条件 $\mathbf{P}(0)$ と帯行列 A_m より w_{mi} を計算する。

7.1　MATLAB を用いた計算

このような計算を行うには、MATLAB もしくは Octave を用いるのが圧倒的に便利である。$ndim$-state で **Q** と初期値 **p0** がわかっているとすれば、下記のスクリプトで用が足る。

```
%---------------------------
  [X,EV]=eig(-Q);
  Y=inv(X);
  W=zeros(ndim,ndim);
  for i=[1:1:ndim]
    W(:,i) = p0*X(:,i)*Y(i,:);
  end
%---------------------------
```

7.2　C 言語等を用いる場合

このような計算はさほど時間がかかるものではなく、通常は Matlab/Octave で用が足りるはずである。しかし、大規模なプログラムから呼ばなければならない様な場合は、C/C++などで書かなくてはならないこともありうる。そのときの参考に。

7.2.1　固有値 λ_m と固有ベクトル $\mathbf{X}_m$

固有値および固有ベクトルを求める計算方法は、いろいろなパッケージライブラリーにあるが、Lapack を用いるのが一番正統な方法であろう。

Lapack はマトリクス計算のプログラムを集めた Fortran パッケージライブラリーであり、ソースコードが公開されている。また Lapack を f2c (fortrac to C converter) で C に変換した clapack もソースコードが公開されている。MacOS の場合、Accelarate Framework に Lapack が含まれているので、ヘッダーファイルを加えるだけで C/C++からの使用可能。

clapack を使用する上で注意すべきことは、clapack は Fortran で書かれたもののの C 言語訳であるためと思われるが、多次元の配列での添数の順が逆になることである。すなわち、clapack で用いられている配列 $a(i,j)$ は, 通常の C 言語で用いられる $a[j][i]$ に相当する。また配列の最初の要素が $a[1]$ であるのか $a[0]$ であるのかには常に気を配らなくてはならない。

clapack の一部である dgeev は、一般マトリクスの固有値と固有ベクトルを計算する。dgeev は、計算が double(d)、一般マトリクス (g, general)、固有値 (e, eigenvalue)、固有ベクトル (ev, eigenvector) という意味。

7.2.2 逆行列 Y の計算

dgeev で得られたモードマトリクスの逆行列を求める。逆行列を求めるルーチンは一般的なルーチンで用が足りる。Lapack の dgetrf と dgetri を用いている。

7.2.3 帯行列 Spectral matrix A_m の計算

モードマトリクスとその逆行列から容易に帯行列が計算できる。

8 例題

チャネルが薬剤によりブロックされる場合のシミュレーション。ブロッカーがない状態で、はじめチャネルは静止状態にある (**R**: state 1)。脱分極により、チャネルは ν_o の率で開いた状態 (O: State 3) へ推移する。**O** の状態からは、ν_c の率で **C** へ戻るとともに、ν_i の率で不活性化した状態 (**I**: state 5) へ移行する。ブロッカーの存在下では、ブロックする率はブロッカーの濃度 c に比例すると考え、**R** にあるチャネルがブロッキングを起こす率を $c\kappa_r$、**O** にあるチャネルがブロッキングを起こす率を $c\kappa_o$ で表すこととする。またブロッカーは、$\mathbf{R}_b$ (state 2) にあるチャネルおよび **B** (state 4) にあるチャネルから、それぞれ ν_{-rb} および ν_{-b} の率で離れるとする。ここでは、ブロックされたチャネルは、開閉しないし不活性化もしないとしておく。

$$
\begin{array}{ccccccc}
 & \mathbf{R}_b(2) & & & \mathbf{B}(4) & & \\
 & c\kappa_r \uparrow\downarrow \nu_{-rb} & & & c\kappa_o \uparrow\downarrow \nu_{-b} & & \\
 & & \nu_o & & & \nu_i & \\
\mathbf{R}(1) & & \rightleftarrows & \mathbf{O}(3) & & \longrightarrow & \mathbf{I}(5) \\
 & & \nu_c & & & &
\end{array}
$$

遷移率行列 Q は、

$$
Q = \begin{pmatrix}
-(c\kappa_r + \nu_o) & c\kappa_r & \nu_o & 0 & 0 \\
\nu_{-rb} & -\nu_{-rb} & 0 & 0 & 0 \\
\nu_c & 0 & -(\nu_c + c\kappa_o + \nu_i) & c\kappa_o & \nu_i \\
0 & 0 & \nu_{-b} & -\nu_{-b} & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

と表される。$\nu_o = 1000/\mathrm{s}$, $\nu_c = 200/\mathrm{s}$, $\nu_i = 50/\mathrm{s}$, $c = 1\mu\mathrm{M}$, $\kappa_r = 1 \times 10^6/\mathrm{sM}$, $\kappa_o = 1.5 \times 10^8/\mathrm{sM}$, $\nu_{-rb} = 0.5/\mathrm{s}$, $\nu_{-b} = 100/\mathrm{s}$, $p_1(0) = 1/3$, $p_2(0) = 2/3$, $p_3(0) = p_4(0) = p_5(0) = 0$ とする。計算精度を上げるため、実際の計算には行列 Q の要素を $1/1000$ にして行う。

9 ソースコード

9.1 Matlab/Octave

```
%---------------------------
ndim = 5;
nu_o = 1000;
nu_c = 200;
nu_i = 50;
c = 1e-6;
kappa_r = 1e6;
kappa_o = 1.5e8;
nu_rb = 0.5;
nu_b = 100;

p0=[1/3, 2/3, 0, 0, 0];

Q = [-(c*kappa_r+nu_o),   c*kappa_r,   nu_o,  0,    0; \
     nu_rb,       -nu_rb,        0,        0,       0; \
     nu_c,  0, -(nu_c+c*kappa_o+nu_i), c*kappa_o, nu_i;\
     0,        0,       nu_b ,       -nu_b,          0;\
     0,        0,         0,          0,             0];

Q = -Q/1000;

[X,EV]=eig(Q);
Y=inv(X);

W=zeros(ndim,ndim);

for i=[1:1:ndim]
  W(:,i) = p0*X(:,i)*Y(i,:);
end
W
%---------------------------
```

9.2 C++

```
//-----------------------------------
#include <iostream>
#include <cmath>
#include <vector>
#include <Accelerate/Accelerate.h> // mac-os
```

```
#define NDIM 5

using namespace std;

void convertCtoF(int m, int n, double* F, vector<vector<double> > C)
{
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
             F[i+n*j] = C[i][j];
        }
    }
}

void scaleMat(int m, int n, double a, vector<vector<double> >& C)
{
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            C[i][j] = a * C[i][j];
        }
    }
}

int main()
{
    int n=NDIM;
    double p0[]={1.0/3.0, 2.0/3.0, 0.0, 0,0, 0,0};  // intial values

    vector< vector<double> > Q(NDIM, vector<double>(NDIM, 0.0));
    double nu_o = 1000.0;
    double nu_c = 200.0;
    double nu_i = 50.0;
    double c = 1.0e-6;
    double kappa_r = 1.0e6;
    double kappa_o = 1.5e8;
    double nu_rb = 0.5;
    double nu_b = 100.0;

    Q[0][0] = -(c*kappa_r+nu_o);
    Q[0][1] =  c*kappa_r;
    Q[0][2] =  nu_o;
    Q[1][0] =  nu_rb;
    Q[1][1] = -nu_rb;
```

```
Q[2][0] =  nu_c;
Q[2][2] = -(nu_c+c*kappa_o+nu_i);
Q[2][3] =  c*kappa_o;
Q[2][4] =  nu_i;
Q[3][2] =  nu_b;
Q[3][3] = -nu_b;

double X[NDIM*NDIM];
double Y[NDIM*NDIM];
double W[NDIM][NDIM];
double A[NDIM][NDIM][NDIM];
double eigenval[NDIM];
double QC[NDIM*NDIM];

int lda = n;
int ldvl = 1;
int ldvr = n;

double work[4*NDIM*NDIM];
double wi[NDIM];
double vr[NDIM*NDIM];
double *vl = NULL;
int ipiv[NDIM];
int lwork = 4*n;
int info;

scaleMat(n, n, -0.001, Q);
convertCtoF(n, n, QC, Q);

dgeev_((char*)"N", (char*)"V", &n, QC, &lda, eigenval, wi, vl, &ldvl, vr, &ldvr,
          work, &lwork, &info); // eigenvalues and eigenvectors

printf("++++ Eigenvalues ++++\n");
for(int i=0;i<n;i++){
    printf("%d : %10.4f\n", i, eigenval[i]);
}

for(int i=0;i<n*n;i++){
    Y[i] = X[i] = vr[i]; // vr contains eigenvectors
}

printf("++++ eigenvectors ++++\n");
for(int i=0;i<n;i++){
```

```
    for(int j=0;j<n;j++){
        printf("%10.5f", X[i + n*j]);
    }
    printf("\n");
}


lwork = n;
dgetrf_(&n, &n, Y, &lda, ipiv, &info); // inversion of eigenvectors
dgetri_(&n, Y, &lda, ipiv, work, &lwork, &info);

printf("++++ inverse of eigenvectors ++++\n");
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        printf("%10.5f", Y[i + n*j]);
    }
    printf("\n");
}

for(int m=0 ; m<n ; m++){
    for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
            A[m][k][i] = X[k+n*m] * Y[m+n*i]; // spectral matrices
        }
    }
}

for(int i=0;i<n;i++) {
    for(int m=0;m<n;m++){
        double sum = 0.0;
        for(int k=0;k<n;k++){
            sum += p0[k]*A[m][k][i];  // W_mi matrix
        }
        W[m][i] = sum;
    }
}

printf("++++ W_mi matrices ++++\n");
for(int i=0;i<n;i++) {
    for(int m=0;m<n;m++){
        printf("%10.5f", W[m][i]);
    }
    printf("\n");
```

```
        }

        return 0;
}
```

9.3 Python

numpy を使用。

```
import numpy

ndim = 5
nu_o = 1000.0
nu_c = 200.0
nu_i = 50.0
c = 1.0e-6
kappa_r = 1.0e6
kappa_o = 1.5e8
nu_rb = 0.5
nu_b = 100.0

p0=[1.0/3.0, 2.0/3.0, 0.0, 0.0, 0.0]

Q = numpy.array( [ (-(c*kappa_r+nu_o),c*kappa_r, nu_o, 0.0, 0.0),
    (nu_rb,-nu_rb, 0.0, 0.0, 0.0),
    (nu_c, 0.0, -(nu_c+c*kappa_o+nu_i), c*kappa_o, nu_i),
    (0.0, 0.0, nu_b, -nu_b, 0.0),
    (0.0, 0.0, 0.0, 0.0, 0.0)] )

Q = -Q/1000.0
(EV, X)=numpy.linalg.eig(Q)

Y=numpy.linalg.inv(X)
W=numpy.zeros((ndim,ndim));
for i in range(ndim):
    W[:,i] = numpy.dot(numpy.dot(p0,X[:,i]), Y[i,:])

print W
```

9.4 実行結果

固有値 λ_m と w_{mi} は計算の結果次のようになる。

$\lambda_1 = 0.00000000$, $\lambda_2 = 0.00049746$, $\lambda_3 = 0.01671314$, $\lambda_4 = 0.24206213$, $\lambda_5 = 1.24222727$,

$$(w_{mi}) = \begin{pmatrix} 0.000000 & 0.001701 & 0.019163 & 0.055137 & 0.257333 \\ 0.000000 & 0.668284 & -0.001182 & -0.000228 & -0.000207 \\ 0.000000 & 0.006836 & 0.094312 & 0.209230 & -0.310378 \\ 0.000000 & 0.010306 & 0.169856 & -0.220921 & 0.040760 \\ 1.000000 & -0.687127 & -0.282148 & -0.043218 & 0.012493 \end{pmatrix}$$

行列の第 1 列は、$t = \infty$ における各の状態の確率 $p_i(\infty)$ にあたる。チャネルが開いている状態 **O** すなわち state 3 にチャネルがある確率の時間経過（t の単位は ms）は、

$$\begin{aligned} p_3(t) &= p_3(\infty) + 0.006836e^{-\frac{1}{2010}t} + 0.094312e^{-\frac{1}{59.8}t} \\ &\quad +0.209230e^{-\frac{1}{4.1}t} - 0.310378e^{-\frac{1}{0.81}t} \\ p_3(\infty) &= 0 \end{aligned}$$

と表される。

付 録

A Markov 過程

マルコフ過程とは現在の状態がわかっていれば過去に無関係に将来が予測されうる確率過程 stochastic process である。特にすぐ前の時刻のみに関係している場合を、単純 Markov 過程、i 個の時刻まで関係しているときは、i 重 Markov 過程という。通常 Markov 過程は単純 Markov 過程を指す。いま時刻系列 t_i $(i = 0, 1, 2, \ldots, n-1)$ で、$X(t)$ が $a_0, a_1, a_2, \ldots, a_{n-1}$ の値をとったとき、時刻 t_n で $X(t_n) = a_n$ となる条件付き確率を、

$$Prob(\ X(t_n) = a_n \mid X(t_0) = a_0, X(t_1) = a_1, \ldots, X(t_{n-1}) = a_{n-1}\ )$$

と表す。Markov 過程では、

$$\begin{aligned} &Prob(\ X(t_n) = a_n \mid X(t_0) = a_0, X(t_1) = a_1, \ldots, X(t_{n-1}) = a_{n-1}\ ) \\ &\quad = Prob(\ X(t_n) = a_n \mid X(t_{n-1}) = a_{n-1}\ ) \end{aligned}$$

が成り立つ。この確率を遷移確率 transition probability という。時刻 t_1 から t_2 の間に、状態 i から j に遷移する確率を $\pi_{ij}(t_1, t_2)$ とする。すなわち

$$\pi_{ij}(t_1, t_2) = Prob(\ X(t_2) = a_j \mid X(t_1) = a_i)$$

$\pi_{ij}(t_1, t_2)$ は確率であるから、

$$\sum_{k=1}^{n} \pi_{ik}(t_1, t_2) = 1$$

$$Prob(\ X(t_2) = a_j\ ) = \sum_{k=1}^{n} Prob(\ X(t_1) = a_k\ )\pi_{kj}(t_1, t_2)$$

が成り立つ。また時刻 $t_1 < t_2 < t_3$ のとき、

$$\pi_{ij}(t_1, t_3) = \sum_{k=1}^{n} \pi_{ik}(t_1, t_2)\pi_{kj}(t_2, t_3)$$

が成り立つ。この式は Chapman-Kolmogoroff equation と呼ばれる。

もし Markov 過程が homogeneous process であれば、π_{ij} は、時刻差 $\tau = t_2 - t_1$ にのみ依存する。$\alpha = t_3 - t_2$ とすると、Chapman-Kolmogoroff equation は、

$$\pi_{ij}(\tau + \alpha) = \sum_{k=1}^{n} \pi_{ik}(\tau)\pi_{kj}(\alpha)$$

となる。π_{ij} を要素とする行列を Π とすると、

$$\Pi(\tau + \alpha) = \Pi(\tau)\Pi(\alpha)$$

と表すことができる。$\pi_{ij}(\tau)$ を τ について微分したときの微分係数を要素とする行列を $Q = (q_{ij})$ とする。すなわち、

$$Q = \lim_{\Delta t \to +0} \frac{\Pi(\Delta t) - \Pi(0)}{\Delta t}$$

と定義する。行列 Q の各要素は定数である。$\sum_{k=1}^{n} \pi_{kj}(\tau) = 1$ であるから、

$$\sum_{k=1}^{n} q_{kj} = 0$$

すなわち

$$q_{ii} = -\sum_{k \neq i} q_{ik}$$

である。また

$$\pi_{ij}(0) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

であるから、$\Pi(0)$ は単位マトリクス I_n である。$\Pi(t)$ を t について微分すると、

$$\begin{aligned} \frac{d}{dt}\Pi(t) &= \lim_{\Delta t \to 0} \frac{\Pi(t + \Delta t) - \Pi(t)}{\Delta t} \\ &= \lim_{\Delta t \to 0} \frac{\Pi(t)\Pi(\Delta t) - \Pi(t)\Pi(0)}{\Delta t} \\ &= \Pi(t) \left(\lim_{\Delta t \to 0} \frac{\Pi(\Delta t) - \Pi(0)}{\Delta t} \right) \\ &= \Pi(t)Q \end{aligned}$$

となり、

$$\frac{d}{dt}\Pi(t) = \Pi(t)Q$$

という関係が成り立つことが分かる。

いま状態を表す行ベクトルを

$$\mathbf{P}(t) = (p_1(t)\ p_2(t)\ \ldots\ p_n(t))$$

とする。$p_i(t) = Prob(X(t) = a_i)$ である。また Π の定義から、

$$\mathbf{P}(t) = \mathbf{P}(0)\Pi(t)$$

であるから、

$$\begin{aligned} \frac{d}{dt}\mathbf{P}(t) &= \frac{d}{dt}\left(\mathbf{P}(0)\Pi(t)\right) \\ &= \mathbf{P}(0)\frac{d}{dt}\Pi(t) \\ &= \mathbf{P}(0)\Pi(t)Q \\ &= \mathbf{P}(t)Q \end{aligned}$$

B e^{Qt} の微分

微分方程式 $d\mathbf{P}(t)/dt = \mathbf{P}(t)Q$ の解は、$\mathbf{P}(t) = \mathbf{P}(0)e^{Qt}$ である。確認のために e^{Qt} を展開して微分してみると、

$$\frac{d}{dt}e^{Qt} = Q + \frac{Q^2 t}{1!} + \frac{Q^3 t^2}{2!} + \ldots + \frac{Q^k t^{k-1}}{(k-1)!} + \frac{Q^{k+1} t^k}{k!} + \ldots$$

$$
\begin{aligned}
&= Q\left(I + \frac{Qt}{1!} + \frac{Q^2t^2}{2!} + \ldots + \frac{Q^{k-1}t^{k-1}}{(k-1)!} + \frac{Q^kt^k}{k!} + \ldots\right) \\
&= \left(I + \frac{Qt}{1!} + \frac{Q^2t^2}{2!} + \ldots + \frac{Q^{k-1}t^{k-1}}{(k-1)!} + \frac{Q^kt^k}{k!} + \ldots\right)Q \\
&= Qe^{Qt} \\
&= e^{Qt}Q
\end{aligned}
$$

となる。したがって、

$$
\begin{aligned}
\frac{d}{dt}\mathbf{P}(t) &= \frac{d}{dt}\left(\mathbf{P}(0)e^{Qt}\right) \\
&= \mathbf{P}(0)\frac{d}{dt}e^{Qt} \\
&= \mathbf{P}(0)e^{Qt}Q \\
&= \mathbf{P}(t)Q
\end{aligned}
$$

であり、$d\mathbf{P}(t)/dt = \mathbf{P}(t)Q$ を満たしている。

C　固有値と行列の階乗

行列 M の固有値 eigenvalue を λ_m $(m = 1, 2, \ldots, n)$ とし、それぞれに対応する固有ベクトル eigenvector を $\mathbf{X}_m$ とする。すなわち $M\mathbf{X}_m = \lambda_m\mathbf{X}_m$ ここで行列 X を $\mathbf{X}_m$ のモードマトリクスとする。すなわち

$$
X = (\mathbf{X}_1\,\mathbf{X}_2\,\ldots\,\mathbf{X}_n)
$$

また固有値 λ_m が対角要素である対角行列を

$$
\Lambda = \begin{pmatrix}
\lambda_1 & 0 & \ldots & 0 \\
0 & \lambda_2 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \lambda_n
\end{pmatrix}
$$

とすると、

$$
\begin{aligned}
MX &= M\,(\mathbf{X}_1\,\mathbf{X}_2\,\ldots\,\mathbf{X}_n) \\
&= (M\mathbf{X}_1\,M\mathbf{X}_2\,\ldots\,M\mathbf{X}_n) \\
&= (\lambda_1\mathbf{X}_1\,\lambda_2\mathbf{X}_2\,\ldots\,\lambda_n\mathbf{X}_n) \\
&= X\Lambda
\end{aligned}
$$

の関係が成り立つ。X の逆行列を X^{-1} とする。$XX^{-1} = I$ であるから、$X\Lambda = MX$ の両辺に右から X^{-1} を掛けて、

$$
X\Lambda X^{-1} = MXX^{-1}
$$

すなわち、

$$
X\Lambda X^{-1} = M
$$

となる。この $M = X\Lambda X^{-1}$ という性質を利用すると、行列 M の階乗が容易に計算できる。

$$\begin{aligned} M^n &= \underbrace{(X\Lambda X^{-1})(X\Lambda X^{-1})\ldots(X\Lambda X^{-1})}_{n \text{ 回掛け合わせる}} \\ &= X\Lambda^n X^{-1} \end{aligned}$$

D e^{Qt} の計算方法

遷移率行列 Q に戻って、$-Q$ の固有値と固有ベクトルを λ_m と $\mathbf{X}_m$ とし、λ_m が対角要素である対角行列を Λ とする。$-Q$ を用いるのは、λ_i が負の値をとらないようにするためである。e^{Qt} を $Q^n = X(-\Lambda)^n X^{-1}$ すなわち $(Qt)^n = X(-\Lambda t)^n X^{-1}$ を利用して計算する。

$$\begin{aligned} e^{Qt} &= I + \frac{Qt}{1!} + \frac{(Qt)^2}{2!} + \ldots + \frac{(Qt)^k}{k!} + \ldots \\ &= I + X(-\Lambda t)X^{-1} + \frac{X(-\Lambda t)^2 X^{-1}}{2!} + \ldots + \frac{X(-\Lambda t)^k X^{-1}}{k!} + \ldots \\ &= X\left(I + (-\Lambda t) + \frac{(-\Lambda t)^2}{2!} + \ldots + \frac{(-\Lambda t)^k}{k!} + \ldots\right)X^{-1} \\ &= Xe^{-\Lambda t}X^{-1} \end{aligned}$$

となる。$e^{\Lambda t}$ の対角要素は、

$$1 + (-\lambda_m t) + \frac{(-\lambda_m t)^2}{2!} + \ldots + \frac{(-\lambda_m t)^k}{k!} + \ldots$$

となる。ここで x がスカラー量であるとき、

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^k}{k!} + \ldots$$

であるから、$e^{\Lambda t}$ の対角要素は、$e^{-\lambda_m t}$ に等しい。したがって

$$e^{Qt} = X\begin{pmatrix} e^{-\lambda_1 t} & 0 & \ldots & 0 \\ 0 & e^{-\lambda_2 t} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & e^{-\lambda_n t} \end{pmatrix}X^{-1}$$

で表わされる。

E 帯行列 Spectral matrix

行列 X はモードマトリクスである。すなわち固有ベクトルを並べて出来る行列である。

$$X = (\mathbf{X}_1\ \mathbf{X}_2\ \ldots\ \mathbf{X}_n)$$

行列 $-Q$ から数値計算により、X を求めることが可能である。またその逆行列 X^{-1} も数値計算によって求めることが出来る。X，X^{-1} は時刻 t に依存しない。

$$X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \ldots & x_n^{(n)} \end{pmatrix}$$

$$
\begin{aligned}
Y &= X^{-1} \\
&= \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{pmatrix}
\end{aligned}
$$

表記することとする。e^{Qt} を計算すると、

$$
\begin{aligned}
e^{Qt} &= Xe^{-\Lambda t}Y \\
&= \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(n)} \end{pmatrix} \begin{pmatrix} e^{-\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{-\lambda_2 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{-\lambda_n t} \end{pmatrix} \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{pmatrix} \\
&= \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(n)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(n)} \end{pmatrix} \begin{pmatrix} y_{11}e^{-\lambda_1 t} & y_{12}e^{-\lambda_1 t} & \cdots & y_{1n}e^{-\lambda_1 t} \\ y_{21}e^{-\lambda_2 t} & y_{22}e^{-\lambda_2 t} & \cdots & y_{2n}e^{-\lambda_2 t} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1}e^{-\lambda_n t} & y_{n2}e^{-\lambda_n t} & \cdots & y_{nn}e^{-\lambda_n t} \end{pmatrix} \\
&= \begin{pmatrix} \sum_{m=1}^{n} x_1^{(m)}y_{m1}e^{-\lambda_m t} & \sum_{m=1}^{n} x_1^{(m)}y_{m2}e^{-\lambda_m t} & \cdots & \sum_{m=1}^{n} x_1^{(m)}y_{mn}e^{-\lambda_m t} \\ \sum_{m=1}^{n} x_2^{(m)}y_{m1}e^{-\lambda_m t} & \sum_{m=1}^{n} x_2^{(m)}y_{m2}e^{-\lambda_m t} & \cdots & \sum_{m=1}^{n} x_2^{(m)}y_{mn}e^{-\lambda_m t} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{m=1}^{n} x_n^{(m)}y_{m1}e^{-\lambda_m t} & \sum_{m=1}^{n} x_n^{(m)}y_{m2}e^{-\lambda_m t} & \cdots & \sum_{m=1}^{n} x_n^{(m)}y_{mn}e^{-\lambda_m t} \end{pmatrix} \\
&= \sum_{m=1}^{n} \begin{pmatrix} x_1^{(m)}y_{m1} & x_1^{(m)}y_{m2} & \cdots & x_1^{(m)}y_{mn} \\ x_2^{(m)}y_{m1} & x_2^{(m)}y_{m2} & \cdots & x_2^{(m)}y_{mn} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(m)}y_{m1} & x_n^{(m)}y_{m2} & \cdots & x_n^{(m)}y_{mn} \end{pmatrix} e^{-\lambda_m t}
\end{aligned}
$$

固有値 $-\lambda_m$ および固有ベクトル $\mathbf{X}_m$ に対する帯行列 spectral matrix A_m を、$a_{ki}^{(m)} = x_k^{(m)}y_{mi}$ すなわち

$$
\begin{aligned}
A_m &= \begin{pmatrix} a_{11}^{(m)} & a_{12}^{(m)} & \cdots & a_{1n}^{(m)} \\ a_{21}^{(m)} & a_{22}^{(m)} & \cdots & a_{2n}^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(m)} & a_{n2}^{(m)} & \cdots & a_{nn}^{(m)} \end{pmatrix} \\
&= \begin{pmatrix} x_1^{(m)}y_{m1} & x_1^{(m)}y_{m2} & \cdots & x_1^{(m)}y_{mn} \\ x_2^{(m)}y_{m1} & x_2^{(m)}y_{m2} & \cdots & x_2^{(m)}y_{mn} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(m)}y_{m1} & x_n^{(m)}y_{m2} & \cdots & x_n^{(m)}y_{mn} \end{pmatrix}
\end{aligned}
$$

と定義する。また

$$w_{mi} = \sum_{k=1}^{n} p_k(0) a_{ki}^{(m)}$$

とすると、

$$\begin{aligned}
\mathbf{P}(t) &= \mathbf{P}(0) e^{Qt} \\
&= \mathbf{P}(0) X e^{-\Lambda t} Y \\
&= \mathbf{P}(0) \sum_{m=1}^{n} \left(A_m e^{-\lambda_m t} \right) \\
&= \sum_{m=1}^{n} \left(\mathbf{P}(0) A_m e^{-\lambda_m t} \right)
\end{aligned}$$

すなわち

$$p_i(t) = \sum_{m=1}^{n} w_{mi} e^{-\lambda_m t}$$

が得られる。

参考文献

[1] Colquhoun D, Hawkes AG. 1995. A Q-matrix cookbook: How to write only one program to calculate the single-channel and macroscopic prediction for any kinetic mechanism. In: *Single-channel recording, 2nd ed.* (Sakmann B and Neher E ed.) Plenum Press, New York.

[2] Papoulis A. 1991. Probability, Random Variables, and Stochastic Processes. 3rd ed. McGraw Hill

[3] 小島紀男, 矢沢志雄作, 本間光一. マトリクスとシステム: 工学のためのマトリクス 3. 1990. 東海大学出版会.

[4] Press WH, Teukolsky SA, Vetterling WT, Flannery BP.1992, Numerical Recipes in C: the art of scientific computing, 2nd ed. Cambridge University Press, Cambridge, UK.

[5] Anderson E, Bai Z, Bishof C, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S,McKenny A, Ostrouchov S, Sorensen D. 1995. LAPACK user's guide, 2nd ed. Society for Industrial and Applied Mathematics. 小国 力 訳 行列計算パッケージ LAPACK 利用の手引.1995 丸善.