

Julia で学ぶ計算論的神経科学

山本 拓都

2023 年 7 月 10 日

目次

第 1 章	はじめに	5
第 2 章	神経細胞のモデル	7
第 3 章	シナプス伝達のモデル	9
第 4 章	神経回路網の演算処理	11
第 5 章	局所学習則	13
第 6 章	生成モデルとエネルギーベースモデル	15
第 7 章	貢献度分配問題の解決策	17
第 8 章	運動制御	19
8.1	無限時間最適フィードバック制御モデル	19
8.1.1	モデルの構造	19
8.1.2	実装	19
8.1.3	Target jump	25
第 9 章	強化学習	29
第 10 章	神経回路網によるベイズ推論	31

第 1 章

はじめに

第 2 章

神経細胞のモデル

第 3 章

シナプス伝達のモデル

第 4 章

神経回路網の演算処理

第 5 章

局所學習則

第 6 章

生成モデルとエネルギーベースモデル

第 7 章

貢献度分配問題の解決策

第 8 章

運動制御

8.1 無限時間最適フィードバック制御モデル

8.1.1 モデルの構造

() [?]

$$dx = (Ax + Bu)dt + Yudy + Gd\omega \quad (8.1)$$

$$dy = Cxdt + Dd\xi \quad (8.2)$$

$$d\hat{x} = (A\hat{x} + Bu)dt + K(dy * C\hat{x}dt) \quad (8.3)$$

8.1.2 実装

ライブラリの読み込みと関数の定義.

```
using Parameters: @unpack
using LinearAlgebra, Kronecker, Random, BlockDiagonals, PyPlot
rc("axes.spines", top=false, right=false)
rc("font", family="Arial")
```

定数の定義

$$\alpha_1 = \frac{b}{t_a t_e I}, \quad \alpha_2 = \frac{1}{t_a t_e} + \left(\frac{1}{t_a} + \frac{1}{t_e} \right) \frac{b}{I} \quad (8.4)$$

$$\alpha_3 = \frac{b}{I} + \frac{1}{t_a} + \frac{1}{t_e}, \quad b_u = \frac{1}{t_a t_e I} \quad (8.5)$$

```

@kwdef struct SaccadeModelParameter
    n = 4 # number of dims
    i = 0.25 # kgm^2,
    b = 0.2 # kgm^2/s
    ta = 0.03 # s
    te = 0.04 # s
    L0 = 0.35 # m

    bu = 1 / (ta * te * i)
    α1 = bu * b
    α2 = 1/(ta * te) + (1/ta + 1/te) * b/i
    α3 = b/i + 1/ta + 1/te

    A = [zeros(3) I(3); -[0, α1, α2, α3]']
    B = [zeros(3); bu]
    C = [I(3) zeros(3)]
    D = Diagonal([1e-3, 1e-2, 5e-2])

    Y = 0.02 * B
    G = 0.03 * I(n)

    Q = Diagonal([1.0, 0.01, 0, 0])
    R = 0.0001
    U = Diagonal([1.0, 0.1, 0.01, 0])
end

```

$$X \triangleq \begin{pmatrix} x \\ \tilde{x} \end{pmatrix}, d\bar{\omega} \triangleq \begin{pmatrix} d\omega \\ d\xi \end{pmatrix}, \bar{A} \triangleq \begin{pmatrix} A * BL & BL \\ 0 & A * KC \end{pmatrix} \quad (8.6)$$

$$\bar{Y} \triangleq \begin{pmatrix} *YL & YL \\ *YL & YL \end{pmatrix}, \bar{G} \triangleq \begin{pmatrix} G & 0 \\ G & *KD \end{pmatrix} \quad (8.7)$$

とする．元論文では F, \bar{F} が定義されていたが， $F = 0$ とするため，以後の式から削除した．

$$P \triangleq \begin{pmatrix} P_{11} & P_{12} \\ P_{12} & P_{22} \end{pmatrix} = \mathbb{E}(XX^T) \quad (8.8)$$

$$V \triangleq \begin{pmatrix} Q + L^T RL & *L^T RL \\ *L^T RL & L^T RL + U \end{pmatrix} \quad (8.9)$$

aaa

$$K = P_{22} C^T (D D^T)^{*1} \quad (8.10)$$

$$L = (R + Y^T (S_{11} + S_{22}) Y)^{*1} B^T S_{11} \quad (8.11)$$

$$\bar{A}^T S + S \bar{A} + \bar{Y}^T S \bar{Y} + V = 0 \quad (8.12)$$

$$\bar{A} P + P \bar{A}^T + \bar{Y} P \bar{Y}^T + \bar{G} \bar{G}^T = 0 \quad (8.13)$$

$A = (a_{ij})$ を $m \times n$ 行列, $B = (b_{kl})$ を $p \times q$ 行列とすると、それらのクロネッカー積 $A \otimes B$ は

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix} \quad (8.14)$$

で与えられる $mp \times nq$ 区分行列である.

Roth's column lemma (vec-trick)

$$(B^T \otimes A) \text{vec}(X) = \text{vec}(AXB) = \text{vec}(C) \quad (8.15)$$

によりこれを解くと,

$$S = * \text{vec}^{*1} \left((I \otimes \bar{A}^T + \bar{A}^T \otimes I + \bar{Y}^T \otimes \bar{Y}^T)^{*1} \text{vec}(V) \right) \quad (8.16)$$

$$P = * \text{vec}^{*1} \left((I \otimes \bar{A} + \bar{A} \otimes I + \bar{Y} \otimes \bar{Y})^{*1} \text{vec}(\bar{G} \bar{G}^T) \right) \quad (8.17)$$

となる. ここで $I = I^T$ を用いた.

K, L, S, P の計算

K, L, S, P の計算は次のようにする.

1. L と K をランダムに初期化
2. S と P を計算
3. L と K を更新
4. 収束するまで 2 と 3 を繰り返す.

収束スピードはかなり速い.

```
function infinite_horizon_ofc(param::SaccadeModelParameter, maxiter=1000, ϵ=1e-8)
    @unpack n, A, B, C, D, Y, G, Q, R, U = param

    # initialize
```

```

L = rand(n)' # Feedback gains
K = rand(n, 3) # Kalman gains
I2n = I(2n)

for _ in 1:maxiter
    A- = [A-B*L B*L; zeros(size(A)) (A-K*C)]
    Y- = [-ones(2) ones(2)] ⊗ (Y*L)
    G- = [G zeros(size(K)); G (-K*D)]
    V = BlockDiagonal([Q, U]) + [1 -1; -1 1] ⊗ (L'* R * L)

    # update S, P
    S = -reshape((I2n ⊗ (A-)' + (A-)' ⊗ I2n + (Y-)' ⊗ (Y-)') \ vec(V), 2n, 2n)
    P = -reshape((I2n ⊗ A- + A- ⊗ I2n + Y- ⊗ Y-) \ vec(G- * (G-)'), 2n, 2n)

    # update K, L
    P22 = P[n+1:2n, n+1:2n]
    S11 = S[1:n, 1:n]
    S22 = S[n+1:2n, n+1:2n]

    Kt-1 = copy(K)
    Lt-1 = copy(L)

    K = P22 * C' / (D * D')
    L = (R + Y' * (S11 + S22) * Y) \ B' * S11
    if sum(abs.(K - Kt-1)) < ε && sum(abs.(L - Lt-1)) < ε
        break
    end
end
return L, K
end

```

```

param = SaccadeModelParameter()
L, K = infinite_horizon_ofc(param);

```

シミュレーション

関数を書く.

```

function simulation(param::SaccadeModelParameter, L, K, dt=0.001, T=2.0, 2n,
    init_pos=-0.5; noisy=true)
    @unpack n, A, B, C, D, Y, G, Q, R, U = param
    nt = round(Int, T/dt)

```

```

X = zeros(n, nt)
u = zeros(nt)
X[1, 1] = init_pos # m; initial position (target position is zero)

if noisy
    sqrt_dt = sqrt(dt)
    X_hat = zeros(n, nt)
    X_hat[1, 1] = X[1, 1]
    for t in 1:nt-1
        u[t] = -L * X_hat[:, t]
        X[:, t+1] = X[:, t] + (A * X[:, t] + B * u[t]) * dt + sqrt_dt * (Y *
            * u[t] * randn() + G * randn(n))
        dy = C * X[:, t] * dt + D * sqrt_dt * randn(n-1)
        X_hat[:, t+1] = X_hat[:, t] + (A * X_hat[:, t] + B * u[t]) * dt + K * (dy *
            - C * X_hat[:, t] * dt)
    end
else
    for t in 1:nt-1
        u[t] = -L * X[:, t]
        X[:, t+1] = X[:, t] + (A * X[:, t] + B * u[t]) * dt
    end
end
return X, u
end

```

理想状況でのシミュレーション

```

dt = 1e-3
T = 1.0

```

```

Xa, ua = simulation(param, L, K, dt, T, noisy=false);

```

ノイズを含むシミュレーション

ノイズを含む場合.

```

n = 4
nsim = 10
XSimAll = []
uSimAll = []
for i in 1:nsim
    XSim, u = simulation(param, L, K, dt, T, noisy=true);
    push!(XSimAll, XSim)
end

```

```

    push!(uSimAll, u)
end

```

結果の描画

```

tarray = collect(dt:dt:T)
label = [L"Position ($m$)", L"Velocity ($m/s$)", L"Acceleration ($m/s^2$)", L"Jerk ($m/s^3$)"]

fig, ax = subplots(1, 3, figsize=(10, 3))
for i in 1:2
    for j in 1:nsim
        ax[i].plot(tarray, XSimAll[j][i,:]', "tab:gray", alpha=0.5)
    end

    ax[i].plot(tarray, Xa[i,:]', "tab:red")
    ax[i].set_ylabel(label[i]); ax[i].set_xlabel(L"Time ($s$)");
    ax[i].set_xlim(0, T); ax[i].grid()
end

for j in 1:nsim
    ax[3].plot(tarray, uSimAll[j], "tab:gray", alpha=0.5)
end
ax[3].plot(tarray, ua, "tab:red")
ax[3].set_ylabel(L"Control signal ($N\cdot m$)"); ax[3].set_xlabel(L"Time ($s$)"); ax[3].set_xlim(0, T); ax[3].grid()

tight_layout()

```

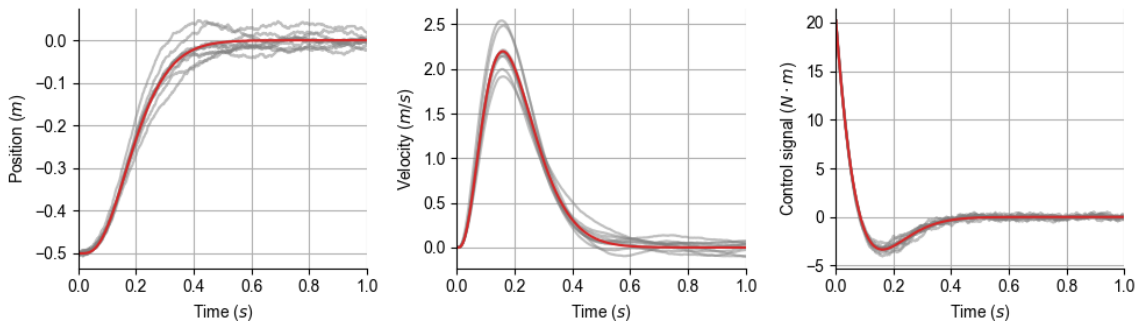


図 8.1 cell017.png

8.1.3 Target jump

target jump する場合の最適制御 [?]. 状態に target 位置も含むモデルであれば target 位置をずらせばよいが、ここでは自己位置をずらし target との相対位置を変化させることで target jump を実現する.

```
function target_jump_simulation(param::SaccadeModelParameter, L, K, ↵
    dt=0.001, T=2.0,
        Ttj=0.4, tj_dist=0.1,
        init_pos=-0.5; noisy=true)
    # Ttj : target jumping timing (sec)
    # tj_dist : target jump distance
    @unpack n, A, B, C, D, Y, G, Q, R, U = param
    nt = round(Int, T/dt)
    ntj = round(Int, Ttj/dt)
    X = zeros(n, nt)
    u = zeros(nt)
    X[1, 1] = init_pos # m; initial position (target position is zero)

    if noisy
        sqrt_dt = sqrt(dt)
        X^ = zeros(n, nt)
        X^[1, 1] = X[1, 1]
        for t in 1:nt-1
            if t == ntj
                X[1, t] -= tj_dist # When k == ntj, target ↵
                    jumpさせる (実際には現在の位置をずらす)
                X^[1, t] -= tj_dist
            end
            u[t] = -L * X^[:, t]
            X[:, t+1] = X[:, t] + (A * X[:, t] + B * u[t]) * dt + sqrt_dt * (Y ↵
                * u[t] * randn() + G * randn(n))
            dy = C * X[:, t] * dt + D * sqrt_dt * randn(n-1)
            X^[:, t+1] = X^[:, t] + (A * X^[:, t] + B * u[t]) * dt + K * (dy ↵
                - C * X^[:, t] * dt)
        end
    else
        for t in 1:nt-1
            if t == ntj
                X[1, t] -= tj_dist # When k == ntj, target ↵
                    jumpさせる (実際には現在の位置をずらす)
            end
            u[t] = -L * X[:, t]
            X[:, t+1] = X[:, t] + (A * X[:, t] + B * u[t]) * dt
        end
    end
end
```

```

    X[1, 1:ntj-1] .-= tj_dist;
    return X, u
end

```

```

Ttj = 0.4
tj_dist = 0.1
nt = round(Int, T/dt)
ntj = round(Int, Ttj/dt);

```

```

Xtj, utj = target_jump_simulation(param, L, K, dt, T, noisy=false);

```

```

XtjAll = []
utjAll = []
for i in 1:nsim
    XSim, u = target_jump_simulation(param, L, K, dt, T, noisy=true);
    push!(XtjAll, XSim)
    push!(utjAll, u)
end

```

```

target_pos = zeros(nt)
target_pos[1:ntj-1] .-= tj_dist;

fig, ax = subplots(1, 3, figsize=(10, 3))
for i in 1:2
    ax[1].plot(tarray, target_pos, "tab:green")
    for j in 1:nsim
        ax[i].plot(tarray, XtjAll[j][i,:]', "tab:gray", alpha=0.5)
    end
    ax[i].axvline(x=Ttj, color="gray", linestyle="dashed")
    ax[i].plot(tarray, Xtj[i,:]', "tab:red")
    ax[i].set_ylabel(label[i]); ax[i].set_xlabel(L"Time ($s$)");
    ax[i].set_xlim(0, T); ax[i].grid()
end
for j in 1:nsim
    ax[3].plot(tarray, utjAll[j], "tab:gray", alpha=0.5)
end
ax[3].axvline(x=Ttj, color="gray", linestyle="dashed")
ax[3].plot(tarray, utj, "tab:red")
ax[3].set_ylabel(L"Control signal ($N \cdot m$)"); ax[3].set_xlabel(L"Time ($s$)");
ax[3].set_xlim(0, T); ax[3].grid()

```

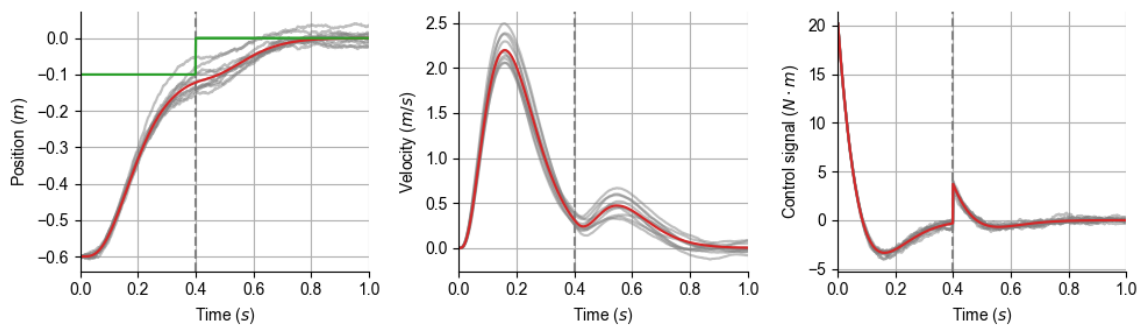
`tight_layout()`

図 8.2 cell023.png

第 9 章

強化学習

第 10 章

神経回路網によるベイズ推論

索引

■ 記号・数字 ■		■ さ ■	
.ist	4	索引スタイルファイル	4
■ ■		■ は ■	
index style	4	ぼくのかんがえたさいきょうの	4
infinite-horizon optimal feedback control model	19	■ ま ■	
		無限時間最適フィードバック制御モデル	19