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

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

局所学習則

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 躍度最小モデル

躍度最小モデル (minimum-jerk model; [**?**]) を実装する．解析的に求まるが以下では二次計画法を用いて数値的に求める．

8.1.1 等式制約下の二次計画法 (Equality Constrained Quadratic Programming)

n 個の変数があり，m 個の制約条件がある等式制約二次計画問題を考える．$\mathbf{x} \in \mathbb{R}^n$, 対称行列 $\mathbf{P} \in \mathbb{R}^{n\times n}$, $\mathbf{q} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$．このとき，問題は次のようになる．

$$\text{Minimize} \quad \frac{1}{2}\mathbf{x}^\top \mathbf{P}\mathbf{x} + \mathbf{q}^\top \mathbf{x} \tag{8.1}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{8.2}$$

Lagrange の未定乗数法を用いると解は

$$\begin{bmatrix} \mathbf{P} & \mathbf{A}^\top \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} -\mathbf{q} \\ \mathbf{b} \end{bmatrix} \tag{8.3}$$

の解として与えられる．ここで $\lambda \in \mathbb{R}^m$ は Lagrange 乗数のベクトルである．

```
using LinearAlgebra, Random, ToeplitzMatrices, PyPlot
rc("axes.spines", top=false, right=false)
```

```
# Equality Constrained Quadratic Programming
function solve_quad_prog(P, q, A, b)
    """
    minimize   : 1/2 * x'*P*x + q'*x
```

```
    subject to : A*x = b
    """
    K = [P A'; A zeros(size(A)[1], size(A)[1])] # KKT matrix
    sol = K \ [-q; b]
    return sol[1:size(A)[2]]
end
```

ちなみに julia では $Ax = b$ の解を出すとき，`x=A^-1 * b` よりも `x=A \ b` とした方がよい．

```
P = diagm([1.0, 0.0])
q = [3.0, 4.0]
A = [1.0, 1.0]'
b = [1.0]
x = solve_quad_prog(P, q, A, b);
```

8.1.2 躍度最小モデルの実装

1 次元における運動を考えよう．この仮定ではサッカードするときの眼球運動などが当てはまる．以下では [?] での問題設定を用いる．Toeplitz 行列を用いた実装は Yazdani らの Python で cvxopt を用いた実装を参考にして作成した．

問題設定は以下のようにする．

$$\underset{u(t)}{\text{minimize}} \quad \|u(t)\|_2 \tag{8.4}$$

$$\text{subject to} \quad \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t) \tag{8.5}$$

ただし，$\|\cdot\|_2$ は L_2 ノルムを意味し，$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \mathbf{x}(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \\ \ddot{x}(t) \end{bmatrix}, u(t) = \dddot{x}(t)$ とする．すなわち，制御信号 $u(t)$ は躍度 $\dddot{x}(t)$ と等しいとする．

```
T = 1.0 # simulation time (sec)
dt = 1e-2 # time step (sec)
nt = Int(T/dt) # number of samples
trange = range(0, 1, length=nt); # range of time
```

```
row_jerk = [[-1, 3, -3, 1]; zeros(nt-4)]
col_jerk = [-1; zeros(nt-4)];
D_jerk = Toeplitz(col_jerk, row_jerk);
```

```
# = diagm(0 => -ones(nt-3), 1 =>3*ones(nt-3), 2=>-3*ones(nt-3), ↵
    3=>ones(nt-3))[1:end-3, :]
```

実際には D_jerk には (1/dt)^3 を乗じるべきであるが，二次計画法の数値的な安定性のために結果の描画の際にのみ乗じる．

```
init_pos = [1; zeros(nt-1)]'
final_pos = [zeros(nt-1); 1]'
init_vel = [[-1, 1]; zeros(nt-2)]'
final_vel = [zeros(nt-2); [-1, 1]]'
init_accel = [[1, -2, 1]; zeros(nt-3)]'
final_accel = [zeros(nt-3); [1, -2, 1]]';

Aeq = [init_pos; final_pos; init_vel; final_vel; init_accel; final_accel];

beq = zeros(6) # (init or final) or (pos, vel, acc) = 2*3
beq[1] = 0     # initial position (m)
beq[2] = 2;    # final position (m)
```

二次計画法を解く．

```
sol_pos = solve_quad_prog(D_jerk' * D_jerk, zeros(nt), Aeq, beq);
```

位置解を速度，加速度，躍度に変換する．

```
# set D_vel and D_accel
row_vel = [[-1, 1]; zeros(nt-2)]
col_vel = [-1; zeros(nt-2)]
D_vel = (1/dt) * Toeplitz(col_vel, row_vel);

row_accel = [[1,-2,1]; zeros(nt-3)]
col_accel = [1; zeros(nt-3)]
D_accel = (1/dt)^2 * Toeplitz(col_accel, row_accel);

# compute solution of vel, accel and jerk
sol_vel = D_vel * sol_pos;
sol_accel = D_accel * sol_pos;
sol_jerk = (1/dt)^3 * D_jerk * sol_pos;
```

結果を描画する．

```
figure(figsize=(8, 4))
subplot(2,2,1)
plot(trange, sol_pos)
```

```
ylabel(L"Position ($m$)"); grid()

subplot(2,2,2)
plot(trange[1:nt-1], sol_vel)
ylabel(L"Velocity ($m/s$)"); grid()

subplot(2,2,3)
plot(trange[1:nt-2], sol_accel)
ylabel(L"Acceleration ($m/s^2$)"); xlabel("Time (s)"); grid()

subplot(2,2,4)
plot(trange[1:nt-3], sol_jerk)
ylabel(L"Jerk ($m/s^3$)"); xlabel("Time (s)"); grid()

tight_layout()
```

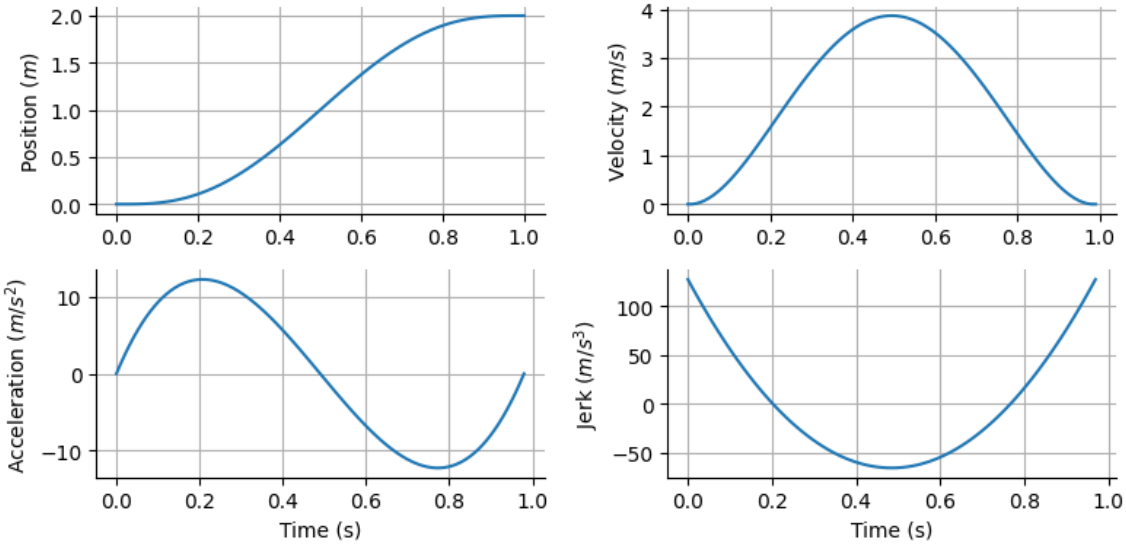

図 8.1 cell015.png

8.1.3 経由点を通る場合

経由点問題 (via-point problem) を考える.

```
via_point_pos = zeros(nt)'
via_point_pos[Int(nt/2)] = 1; # via point timing
```

```
Aeq2 = [init_pos; final_pos; via_point_pos; init_vel; final_vel; ↵
    init_accel; final_accel];

beq2 = zeros(7) # (init or final) or (pos, vel, acc) + via_point_pos = 2*3 ↵
    + 1 = 7
beq2[1] = 2     # inital position (m)
beq2[2] = 4     # final position (m)
beq2[3] = 6;    # via point position (m)
```

```
sol2_pos = solve_quad_prog(D_jerk' * D_jerk, zeros(nt), Aeq2, beq2);
sol2_vel = D_vel * sol2_pos;
sol2_accel = D_accel * sol2_pos;
sol2_jerk = (1/dt)^3 * D_jerk * sol2_pos;
```

```
figure(figsize=(8, 4))
subplot(2,2,1)
plot(trange, sol2_pos)
ylabel(L"Position ($m$)"); grid()

subplot(2,2,2)
plot(trange[1:nt-1], sol2_vel)
ylabel(L"Velocity ($m/s$)"); grid()

subplot(2,2,3)
plot(trange[1:nt-2], sol2_accel)
ylabel(L"Acceleration ($m/s^2$)"); xlabel("Time (s)"); grid()

subplot(2,2,4)
plot(trange[1:nt-3], sol2_jerk)
ylabel(L"Jerk ($m/s^3$)"); xlabel("Time (s)"); grid()

tight_layout()
```

8.2 終点誤差分散最小モデル

終点誤差分散最小モデル (minimum-variance model; [**?**]) を実装する．

$\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{B} \in \mathbb{R}^n$ とする．$\dot{x} = \mathbf{A}_c\mathbf{x} + \mathbf{B}_c(u + w)$ について，差分化すると

$$\mathbf{x}(t + dt) = \mathbf{x}(t) + \dot{\mathbf{x}}dt \tag{8.6}$$

$$\mathbf{x}_{t+1} = \mathbf{I}\mathbf{x}_t + (\mathbf{A}_c dt)\mathbf{x}_t + (\mathbf{B}_c dt)(u + w) \tag{8.7}$$

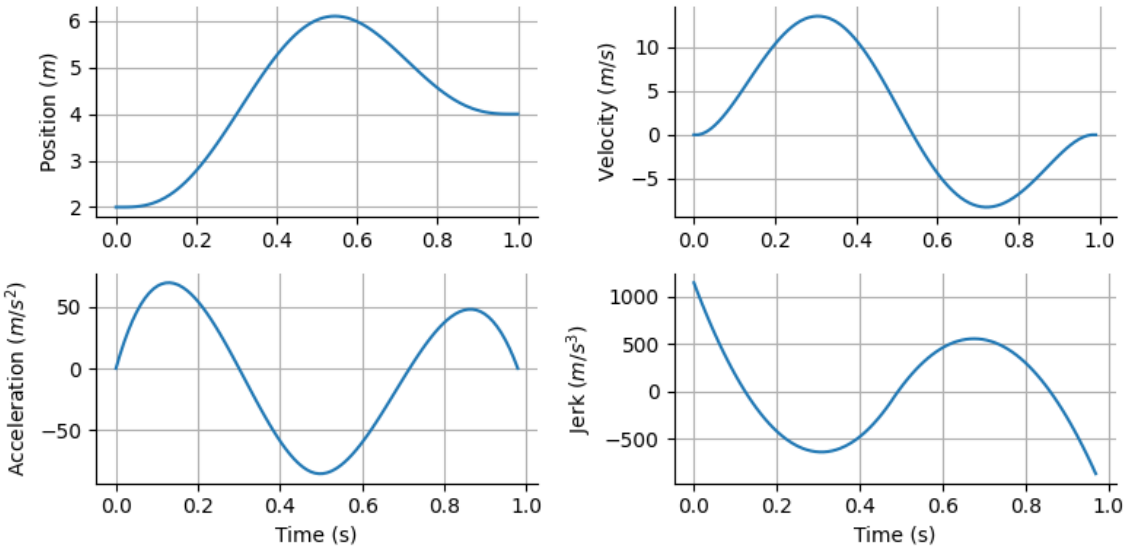

図 8.2 cell019.png

となる (ここで $\mathbf{I}$ は単位行列) ので，$\mathbf{A} = \mathbf{I} + \mathbf{A}_c dt, \mathbf{B} = \mathbf{B}_c dt$ として

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}(u_t + w_t) \tag{8.8}$$

と表せる．$\mathbf{x}_t$ の平均は

$$\mathbb{E}\left[\mathbf{x}_t\right] = \mathbf{A}^t \mathbf{x}_0 + \sum_{i=0}^{t-1} \mathbf{A}^{t-1-i} \mathbf{B} u_i \tag{8.9}$$

$\mathbf{x}_t$ の分散は

$$\mathrm{Cov}\left[\mathbf{x}_t\right] = k \sum_{i=0}^{t-1} \left(\mathbf{A}^{t-1-i}\mathbf{B}\right) \left(\mathbf{A}^{t-1-i}\mathbf{B}\right)^\top u_i^2 \tag{8.10}$$

となる．

8.2.1 終点誤差分散最小モデルの実装

以下では田中先生の https://www.motorcontrol.jp/archives/?MC13 のコードを参考に作成した．

```
using LinearAlgebra, Random, PyPlot
```

```
rc("axes.spines", top=false, right=false)
```

```
# Equality Constrained Quadratic Programming
function solve_quad_prog(P, q, A, b)
    """
    minimize   : 1/2 * x'*P*x + q'*x
    subject to : A*x = b
    """
    K = [P A'; A zeros(size(A)[1], size(A)[1])] # KKT matrix
    sol = K \ [-q; b]
    return sol[1:size(A)[2]]
end;
```

```
function minimum_variance_model(Ac, Bc, x0, xf, tf, tp, dt)
    dims = size(x0)[1]
    ntf = round(Int, tf/dt)
    ntp = round(Int, tp/dt)
    nt = ntf + ntp # total time steps

    A = I(dims) + Ac * dt
    B = Bc*dt
    #A = exp(Ac*dt);
    #B = Ac^-1 * (I(dims) - A) *Bc;

    # calculation of V
    diagV = zeros(nt);
    for t=0:nt-1
        if t < ntf
            diagV[t+1] = sum([(A^(k-t-1) * B * B' * A'^(k-t-1))[1,1] for ↵
                k=ntf:nt-1])
        else
            diagV[t+1] = diagV[t] + (A^(nt-t-2) * B * B' * A'^(nt-t-2))[1,1]
        end
    end
    diagV /= maximum(diagV) # for numerical stability
    V = Diagonal(diagV);

    # 制約条件における行列Cとベクトルdの計算
    #calculation of C
    C = zeros(dims*(ntp+1), nt);
    for p=1:ntp+1
        for q=1:nt
            if ntf-1+(p-1)-(q-1) >= 0
                idx = dims*(p-1)+1:dims*p
```

```
                C[idx, q] = A^(ntf-1-(q-1)+(p-1)) * B # if ↵
                    ntf-1-(q-1)+(p-1) == 0; A^(ntf-1-(q-1)+(p-1))*B equal ↵
                    to B
            end
        end
    end

    # calculation of d
    d = vcat([xf - A^(ntf+t) * x0 for t=0:ntp]...);

    # 制御信号を二次計画法で計算 (solution by quadratic programming)
    u = solve_quad_prog(V, zeros(nt), C, d);

    # 制御信号を二次計画法で計算 (forward solution)
    x = zeros(dims, nt);
    x[:,1] = x0;
    Σ = zeros(dims, dims, nt);
    Σ[:, :, 1] = B * u[1]^2 * B'
    for t=1:nt-1
        x[:,t+1] = A*x[:, t] + B*u[t] # update
        Σ[:, :, t+1] = A * Σ[:, :, t] * A' + B * u[t]^2 * B' # variance
    end
    return x, u, Σ
end
```

```
t1 = 224*1e-3 # time const of eye dynamics (s)
t2 = 13*1e-3  # another time const of eye dynamics (s)
tm = 10*1e-3
dt = 1e-3     # simulation time step (s)
tf = 50*1e-3  # movement duration (s)
tp = 40*1e-3  # post-movement duration (s)
nt = round(Int, (tf+tp)/dt) # total time steps
trange = (1:nt) * dt * 1e3 # ms

# 2nd order
x0₂ = zeros(2) # initial state (pos=0, vel=0)
xf₂ = [10, 0]  # final state (pos=10, vel=0)
Ac₂ = [0 1; -1/(t1*t2) -1/t1-1/t2];
Bc₂ = [0, 1]

# 3rd order
x0₃ = zeros(3)   # initial state (pos=0, vel=0, acc=0)
xf₃ = [10, 0, 0] # final state (pos=10, vel=0, acc=0)
Ac₃ = [0 1 0; 0 0 1; -1/(t1*t2*tm) -1/(t1*t2)-1/(t1*tm)-1/(t2*tm) ↵
    -1/t1-1/t2-1/tm];
```

```
Bc₃ = [0, 0, 1/tm];
```

```
x₂, u₂, Σ₂ = minimum_variance_model(Ac₂, Bc₂, x0₂, xf₂, tf, tp, dt);
x₃, u₃, Σ₃ = minimum_variance_model(Ac₃, Bc₃, x0₃, xf₃, tf, tp, dt);
```

結果の描画.

```
figure(figsize=(6, 4))
subplot(2,2,1); plot(trange, x₂[1, :], label="2nd order"); plot(trange, ↵
    x₃[1, :], "--", label="3rd order");
ylabel("Eye position (deg)"); grid(); legend()
subplot(2,2,2); plot(trange, x₂[2, :]); plot(trange, x₃[2, :], "--");
ylabel("Eye velocity (deg/s)"); grid();
subplot(2,2,3); plot(trange, u₂); plot(trange, u₃, "--");
ylabel("Control signal"); xlabel("Time (ms)"); grid();
ax = gca(); ax[:ticklabel_format](style="sci",axis="y",scilimits=(0,0))
subplot(2,2,4); plot(trange, Σ₂[1,1,:]); plot(trange, Σ₃[1,1,:], "--");
ylabel("Positional Variance"); xlabel("Time (ms)"); grid()
tight_layout()
```

8.3 最適フィードバック制御モデル

ToDo: infiniteOFC と数式の統一を行う.

8.3.1 最適フィードバック制御モデルの構造

最適フィードバック制御モデル (optimal feedback control; OFC) の特徴として目標軌道を必要としないことが挙げられる. **Kalman フィルタ**による状態推定と**線形 2 次レギュレーター (LQR: linear-quadratic regurator)** により推定された状態に基づいて運動指令を生成という 2 つの流れが基本となる.

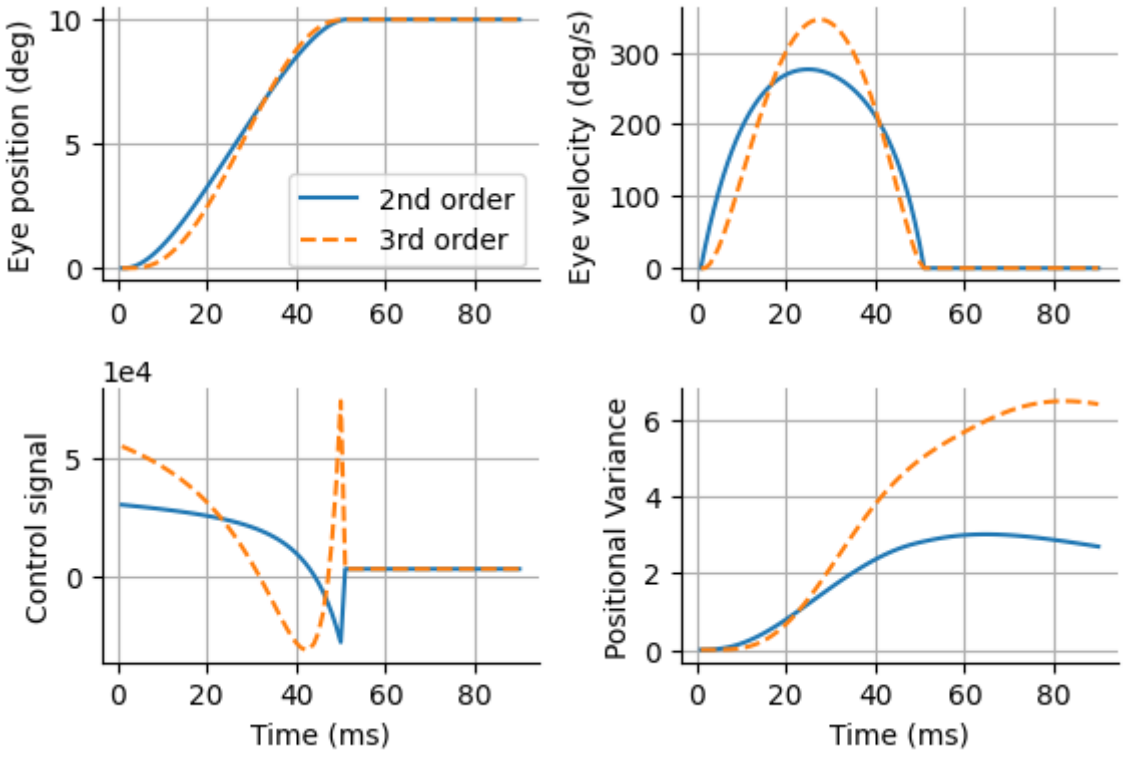

図 8.3 cell008.png

系の状態変化

$$\text{Dynamics} \quad \mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \boldsymbol{\xi}_t + \sum_{i=1}^{c} \varepsilon_t^i C_i \mathbf{u}_t \tag{8.11}$$

$$\text{Feedback} \quad \mathbf{y}_t = H\mathbf{x}_t + \omega_t + \sum_{i=1}^{d} \epsilon_t^i D_i \mathbf{x}_t \tag{8.12}$$

$$\text{Cost per step} \quad \mathbf{x}_t^\top Q_t \mathbf{x}_t + \mathbf{u}_t^\top R \mathbf{u}_t \tag{8.13}$$

LQG

加法ノイズしかない場合 ($C = D = 0$), 制御問題は**線形 2 次ガウシアン (LQG: linear-quadratic-Gaussian) 制御**と呼ばれる.

運動制御 (Linear-Quadratic Regulator)

$$\mathbf{u}_t = -L_t \widehat{\mathbf{x}}_t \tag{8.14}$$

$$L_t = \left(R + B^\top S_{t+1} B\right)^{-1} B^\top S_{t+1} A \tag{8.15}$$

$$S_t = Q_t + A^\top S_{t+1} \left(A - B L_t\right) \tag{8.16}$$

$$s_t = \mathrm{tr}(S_{t+1}\Omega^\xi) + s_{t+1}; s_T = 0 \tag{8.17}$$

$\boldsymbol{S}_T = Q$

状態推定 (Kalman Filter)

$$\widehat{\mathbf{x}}_{t+1} = A\widehat{\mathbf{x}}_t + B\mathbf{u}_t + K_t\left(\mathbf{y}_t - H\widehat{\mathbf{x}}_t\right) + \boldsymbol{\eta}_t \tag{8.18}$$

$$K_t = A\Sigma_t H^\top \left(H\Sigma_t H^\top + \Omega^\omega\right)^{-1} \tag{8.19}$$

$$\Sigma_{t+1} = \Omega^\xi + \left(A - K_t H\right)\Sigma_t A^\top \tag{8.20}$$

この場合に限り，運動制御と状態推定を独立させることができる．

一般化 LQG

状態および制御依存ノイズがある場合，

8.3.2 実装

ライブラリの読み込みと関数の定義．

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

ToDo: struct 修正 (n が両方に入っている)

```
@kwdef struct Reaching1DModelParameter
    n = 4 # number of dims
    p = 3 #
    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(p) I(p); -[0, α1, α2, α3]']
    B = [zeros(p); bu]
    C = [I(p) zeros(p)]
    D = Diagonal([1e-3, 1e-2, 5e-2])

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

@kwdef struct Reaching1DModelCostParameter
    n = 4
    dt = 1e-2 # sec
    T = 0.5 # sec
    nt = round(Int, T/dt) # num time steps
    Q = [zeros(nt-1, n, n); reshape(Diagonal([1.0, 0.1, 1e-3, 1e-4]), (1, ↵
        n, n))]
    R = 1e-4 / nt

    init_pos = -0.5
    x₁ = [init_pos; zeros(n-1)]#zeros(n)
    Σ₁ = zeros(n, n)
end
```

Q の値は各時刻において一般座標 (位置，速度，加速度，躍度) のそれぞれを 0 にするコストに対する重みづけである．例えば，速度も 0 にすることを重視すれば 2 番目の係数を上げる．S と Σ は各時点での値を一時的にしか必要としないので更新する．

```
function LQG(param::Reaching1DModelParameter, ↵
    cost_param::Reaching1DModelCostParameter)
    @unpack n, p, A, B, C, D, G = param
    @unpack Q, R, x₁, Σ₁, dt, nt = cost_param

    A = I + A * dt
    B = B * dt
    C = C * dt
    D = sqrt(dt) * D
    G = sqrt(dt) * G

    L = zeros(nt-1, n) # Feedback gains
    K = zeros(nt-1, n, p) # Kalman gains
```

```
    S = copy(Q[end, :, :]) # S_T = Q
    Σ = copy(Σ₁);

    for t in 1:nt-1
        K[t, :, :] = A * Σ * C' / (C * Σ * C' + D) # update K
        Σ = G + (A - K[t, :, :] * C) * Σ * A'      # update Σ
    end

    cost = 0
    for t in nt-1:-1:1
        cost += tr(S * G)
        L[t, :] = (R + B' * S * B) \ B' * S * A # update L
        S = Q[t, :, :] + A' * S * (A - B * L[t, :]')     # update S
    end

    # adjust cost
    cost += x₁' * S * x₁
    return L, K, cost
end
```

シミュレーション

信号依存ノイズ Y が入っている場合は LQG とは異なってくる.

$$\mathbf{u}_t = -L_t \hat{\mathbf{x}}_t \tag{8.21}$$

$$L_t = \left(B^\top S^{\mathrm{x}}_{t+1} B + R + \sum_n C_n^\top \left(S^{\mathrm{x}}_{t+1} + S^{\mathrm{e}}_{t+1} \right) C_n \right)^{-1} B^\top S^{\mathrm{x}}_{t+1} A \tag{8.22}$$

$$S^{\mathrm{x}}_t = Q_t + A^\top S^{\mathrm{x}}_{t+1} \left(A - B L_t \right); \quad S^{\mathrm{x}}_T = Q_T \tag{8.23}$$

$$S^{\mathrm{e}}_t = A^\top S^{\mathrm{x}}_{t+1} B L_t + \left(A - K_t H \right)^\top S^{\mathrm{e}}_{t+1} \left(A - K_t H \right); \quad S^{\mathrm{e}}_T = 0 \tag{8.24}$$

$$s_t = \mathrm{tr} \left(S^{\mathrm{x}}_{t+1} \Omega^\xi + S^{\mathrm{e}}_{t+1} \left(\Omega^\xi + \Omega^\eta + K_t \Omega^\omega K_t^\top \right) \right) + s_{t+1}; \quad s_n = 0. \tag{8.25}$$

$$\hat{\mathbf{x}}_{t+1} = A \hat{\mathbf{x}}_t + B \mathbf{u}_t + K_t \left(\mathbf{y}_t - H \hat{\mathbf{x}}_t \right) \tag{8.26}$$

$$K_t = A \Sigma^{\mathrm{e}}_t H^\top \left(H \Sigma^{\mathrm{e}}_t H^\top + \Omega^\omega \right)^{-1} \tag{8.27}$$

$$\Sigma^{\mathrm{e}}_{t+1} = \left(A - K_t H \right) \Sigma^{\mathrm{e}}_t A^\top + \sum_n C_n L_t \Sigma^{\hat{x}}_t L_t^\top C_n^\top; \quad \Sigma^{\mathrm{e}}_1 = \Sigma_1 \tag{8.28}$$

$$\Sigma^{\hat{\mathbf{x}}}_{t+1} = K_t H \Sigma^{\mathrm{e}}_t A^\top + \left(A - B L_t \right) \Sigma^{\hat{\mathbf{x}}}_t \left(A - B L_t \right)^\top; \quad \Sigma^{\hat{\mathbf{x}}}_1 = \hat{\mathbf{x}}_1 \hat{\mathbf{x}}_1^\top \tag{8.29}$$

```
function gLQG(param::Reaching1DModelParameter, ↵
    cost_param::Reaching1DModelCostParameter, maxiter=200, ϵ=1e-8)
    @unpack n, p, A, B, C, D, Y, G = param
    @unpack Q, R, x₁, Σ₁, dt, nt = cost_param

    A = I + A * dt
    B = B * dt
    C = C * dt
    D = sqrt(dt) * D
    G = sqrt(dt) * G
    Y = sqrt(dt) * Y

    L = zeros(nt-1, n) # Feedback gains
    K = zeros(nt-1, n, p) # Kalman gains

    cost = zeros(maxiter)
    for i in 1:maxiter
        Sˣ = copy(Q[end, :, :])
        Sᵉ = zeros(n, n)
        Σˣ^ = x₁ * x₁' # \Sigma TAB \^x TAB \hat TAB
        Σᵉ = copy(Σ₁)

        for t in 1:nt-1
            K[t, :, :] = A * Σᵉ * C' / (C * Σᵉ * C' + D)

            AmBL = A - B * L[t, :]'
            LΣˣ^L = L[t, :]' * Σˣ^ * L[t, :]

            Σˣ^ = K[t, :, :] * C * Σᵉ * A' + AmBL * Σˣ^ * AmBL'
            Σᵉ = G + (A - K[t, :, :] * C) * Σᵉ * A' + Y * LΣˣ^L * Y'
        end

        for t in nt-1:-1:1
            cost[i] += tr(Sˣ * G + Sᵉ * (G + K[t, :, :] * D * K[t, :, :]'))

            L[t, :] = (R + B' * Sˣ * B + Y' * (Sˣ + Sᵉ) * Y) \ B' * Sˣ * A

            AmKC = A - K[t, :, :] * C
            Sᵉ = A' * Sˣ * B * L[t, :]' + AmKC' * Sᵉ * AmKC
            Sˣ = Q[t, :, :] + A' * Sˣ * (A - B * L[t, :]')
        end

        # adjust cost
        cost[i] += x₁' * Sˣ * x₁ + tr((Sˣ + Sᵉ) * Σ₁)
        if i > 1 && abs(cost[i] - cost[i-1]) < ϵ
            cost = cost[1:i]
```

```
            break
        end
    end
    return L, K, cost
end
```

状態ノイズがある場合に関しては Todorov の MATLAB コード https://homes.cs.washington.edu/~todorov/software/gLQG.zip を参照.

位置は目標位置を基準とする座標で表現し，位置が 0 になるように運動を行う．状態の中に標的位置を含めコストパラメータを修正することで初期位置を基準とする座標系での運動を記述できる．モデルに関しては Todorov2005 を参照.

```
function simulation(param::Reaching1DModelParameter, ↵
    cost_param::Reaching1DModelCostParameter,
                    L, K; noisy=false)
    @unpack n, p, A, B, C, D, Y, G = param
    @unpack Q, R, x₁, dt, nt = cost_param

    X = zeros(n, nt)
    u = zeros(nt)
    X[:, 1] = x₁ # m; initial position (target position is zero)

    if noisy
        sqrtdt = √dt
        X^ = zeros(n, nt)
        X^[1, 1] = X[1, 1]
        for t in 1:nt-1
            u[t] = -L[t, :]' * X^[:, t]
            X[:, t+1] = X[:,t] + (A * X[:,t] + B * u[t]) * dt + sqrtdt * (Y ↵
                * u[t] * randn() + G * randn(n))
            dy = C * X[:,t] * dt + D * sqrtdt * randn(n-1)
            X^[:, t+1] = X^[:,t] + (A * X^[:,t] + B * u[t]) * dt + K[t, :, ↵
                :] * (dy - C * X^[:,t] * dt)
        end
    else
        for t in 1:nt-1
            u[t] = -L[t, :]' * X[:, t]
            X[:, t+1] = X[:, t] + (A * X[:, t] + B * u[t]) * dt
        end
    end
    return X, u
end
```

```
function simulation_all(param, cost_param, L, K)
    Xa, ua = simulation(param, cost_param, L, K, noisy=false);

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

    # visualization
    @unpack dt, T = cost_param
    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()
end
```

```
param = Reaching1DModelParameter()
cost_param = Reaching1DModelCostParameter();
```

```
L, K, cost = LQG(param, cost_param);
simulation_all(param, cost_param, L, K)
```

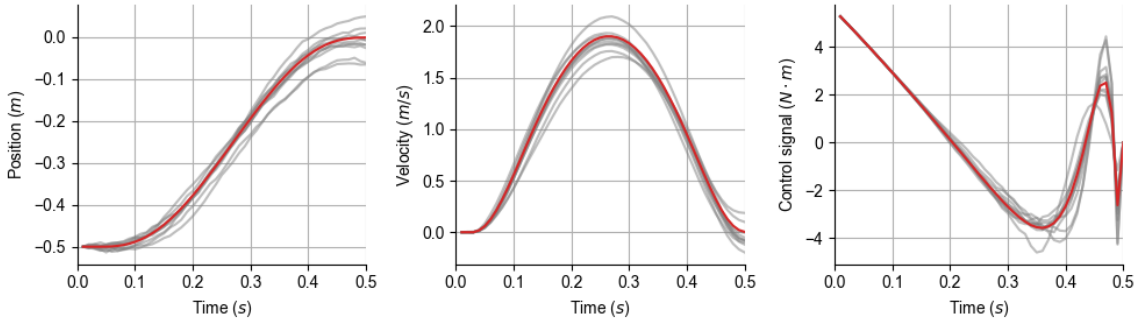

図 8.4 cell016.png

```
L, K, cost = gLQG(param, cost_param);
simulation_all(param, cost_param, L, K)
```

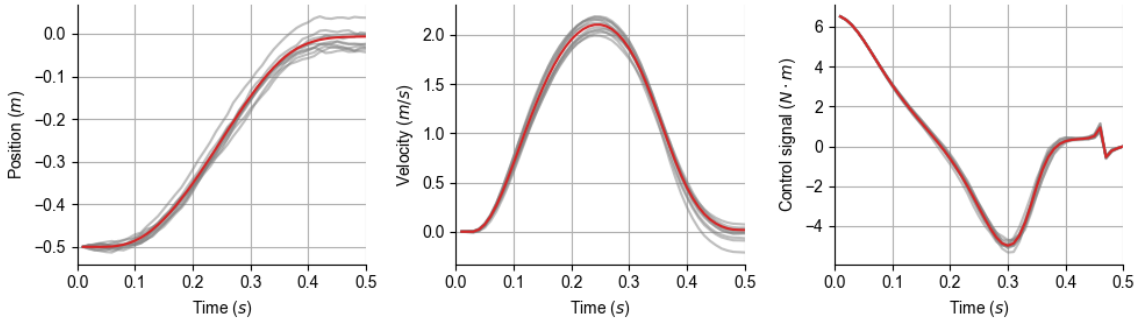

図 8.5 cell017.png

8.4 無限時間最適フィードバック制御モデル

8.4.1 モデルの構造

無限時間最適フィードバック制御モデル (infinite-horizon optimal feedback control model) [?]

$$dx = (\mathbf{A}x + \mathbf{B}u)dt + \mathbf{Y}ud\gamma + \mathbf{G}d\omega \tag{8.30}$$
$$dy = \mathbf{C}xdt + \mathbf{D}d\xi \tag{8.31}$$
$$d\hat{x} = (\mathbf{A}\hat{x} + \mathbf{B}u)dt + \mathbf{K}(dy - \mathbf{C}\hat{x}dt) \tag{8.32}$$

8.4.2 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

$$\alpha_1 = \frac{b}{t_a t_e I}, \quad \alpha_2 = \frac{1}{t_a t_e} + \left(\frac{1}{t_a} + \frac{1}{t_e}\right)\frac{b}{I} \tag{8.33}$$
$$\alpha_3 = \frac{b}{I} + \frac{1}{t_a} + \frac{1}{t_e}, \quad b_u = \frac{1}{t_a t_e I} \tag{8.34}$$

```
@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
```

$$\mathbf{X} \triangleq \begin{bmatrix} x \\ \tilde{x} \end{bmatrix}, d\bar{\omega} \triangleq \begin{bmatrix} d\omega \\ d\xi \end{bmatrix}, \bar{\mathbf{A}} \triangleq \begin{bmatrix} \mathbf{A} - \mathbf{BL} & \mathbf{BL} \\ \mathbf{0} & \mathbf{A} - \mathbf{KC} \end{bmatrix} \tag{8.35}$$

$$\bar{\mathbf{Y}} \triangleq \begin{bmatrix} -\mathbf{YL} & \mathbf{YL} \\ -\mathbf{YL} & \mathbf{YL} \end{bmatrix}, \bar{G} \triangleq \begin{bmatrix} \mathbf{G} & \mathbf{0} \\ \mathbf{G} & -\mathbf{KD} \end{bmatrix} \tag{8.36}$$

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．

$$\mathbf{P} \triangleq \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} \\ \mathbf{P}_{12} & \mathbf{P}_{22} \end{bmatrix} = \mathbb{E}\left[\mathbf{X}\mathbf{X}^\top\right] \tag{8.37}$$

$$\mathbf{V} \triangleq \begin{bmatrix} \mathbf{Q} + \mathbf{L}^\top R\mathbf{L} & -\mathbf{L}^\top R\mathbf{L} \\ -\mathbf{L}^\top R\mathbf{L} & \mathbf{L}^\top R\mathbf{L} + \mathbf{U} \end{bmatrix} \tag{8.38}$$

aaa

$$K = \mathbf{P}_{22}\mathbf{C}^\top \left(\mathbf{D}\mathbf{D}^\top\right)^{-1} \tag{8.39}$$

$$\mathbf{L} = \left(R + \mathbf{Y}^\top \left(\mathbf{S}_{11} + \mathbf{S}_{22}\right)\mathbf{Y}\right)^{-1} \mathbf{B}^\top \mathbf{S}_{11} \tag{8.40}$$

$$\bar{\mathbf{A}}^\top \mathbf{S} + \mathbf{S}\bar{\mathbf{A}} + \bar{\mathbf{Y}}^\top \mathbf{S}\bar{\mathbf{Y}} + \mathbf{V} = 0 \tag{8.41}$$

$$\bar{\mathbf{A}}\mathbf{P} + \mathbf{P}\bar{\mathbf{A}}^\top + \bar{\mathbf{Y}}\mathbf{P}\bar{\mathbf{Y}}^\top + \bar{\mathbf{G}}\bar{\mathbf{G}}^\top = 0 \tag{8.42}$$

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

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix} \tag{8.43}$$

で与えられる $mp \times nq$ 区分行列である．

Roth's column lemma (vec-trick)

$$(\mathbf{B}^\top \otimes \mathbf{A})\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{AXB}) = \mathrm{vec}(\mathbf{C}) \tag{8.44}$$

によりこれを解くと，

$$\mathbf{S} = -\mathrm{vec}^{-1}\left(\left(\mathbf{I}\otimes\bar{\mathbf{A}}^\top + \bar{\mathbf{A}}^\top\otimes\mathbf{I} + \bar{\mathbf{Y}}^\top\otimes\bar{\mathbf{Y}}^\top\right)^{-1}\mathrm{vec}(\mathbf{V})\right) \tag{8.45}$$

$$\mathbf{P} = -\mathrm{vec}^{-1}\left(\left(\mathbf{I}\otimes\bar{\mathbf{A}} + \bar{\mathbf{A}}\otimes\mathbf{I} + \bar{\mathbf{Y}}\otimes\bar{\mathbf{Y}}\right)^{-1}\mathrm{vec}(\bar{\mathbf{G}}\bar{\mathbf{G}}^\top)\right) \tag{8.46}$$

となる．ここで $\mathbf{I} = \mathbf{I}^\top$ 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)
```

```
        L_{t-1} = copy(L)

        K = P_{22} * C' / (D * D')
        L = (R + Y' * (S_{11} + S_{22}) * Y) \ B' * S_{11}
        if sum(abs.(K - K_{t-1})) < ϵ && sum(abs.(L - L_{t-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, ↵
    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
        sqrtdt = √dt
        X^ = zeros(n, nt)
        X^[1, 1] = X[1, 1]
        for t in 1:nt-1
            u[t] = -L * X^[:, t]
            X[:, t+1] = X[:,t] + (A * X[:,t] + B * u[t]) * dt + sqrtdt * (Y ↵
                * u[t] * randn() + G * randn(n))
            dy = C * X[:,t] * dt + D * sqrtdt * 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
            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)
    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()
```

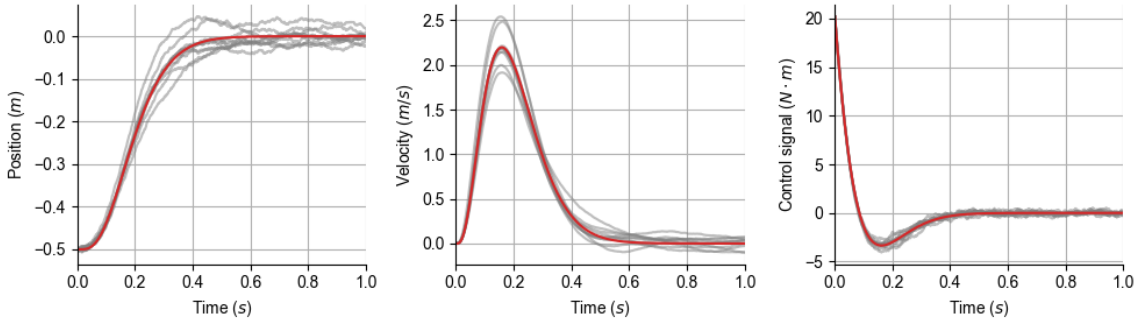

図 8.6 cell017.png

8.4.3 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
        sqrtdt = √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 + sqrtdt * (Y ↵
                * u[t] * randn() + G * randn(n))
            dy = C * X[:,t] * dt + D * sqrtdt * 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
    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()
```

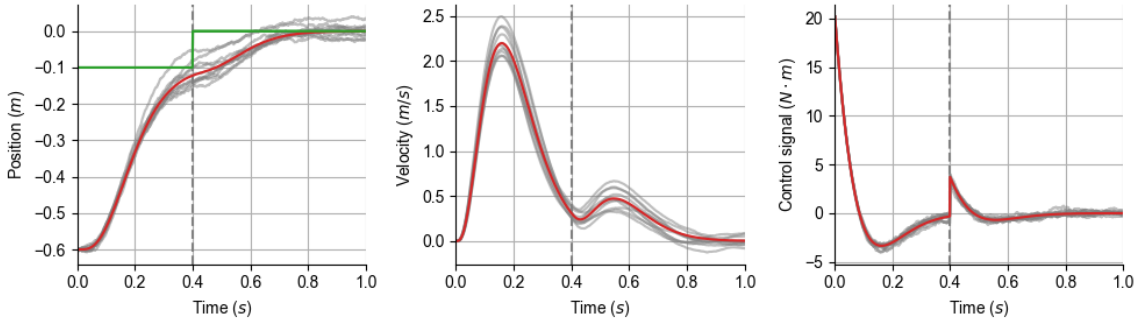

図 8.7 cell023.png

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