

# Julia で学ぶ計算論的神経科学

山本 拓都

2023 年 5 月 5 日



# 目次

- 0.1 神経サンプリング . . . . . 4
  - 0.1.1 ガウス尺度混合モデル . . . . . 4
  - 0.1.2 興奮性・抑制性神経回路によるサンプリング . . . . . 9
  - 0.1.3 Spiking ニューラルネットワークにおけるサンプリング . . . . . 11
  - 0.1.4 シナプスサンプリング . . . . . 11
  - 0.1.5 シナプスサンプリング . . . . . 11

## 0.1 神経サンプリング

サンプリングに基づく符号化 (sampling-based coding; SBC or neural sampling model) をガウス尺度混合モデルを例にとり実装する。

### 0.1.1 ガウス尺度混合モデル

**ガウス尺度混合 (Gaussian scale mixture; GSM) モデル**は確率的生成モデルの一種である [?][?]. GSM モデルでは入力を次式で予測する：

$$\text{入力} = z \left( \sum \text{神経活動} \times \text{基底} \right) + \text{ノイズ}$$

前節までのスパース符号化モデル等と同様に、入力が基底の線形和で表されるとしている。ただし、尺度 (scale) パラメータ  $z$  が基底の線形和に乘じられている点が異なる。<sup>\*1</sup>

#### 事前分布

$\mathbf{x} \in \mathbb{R}^{N_x}$ ,  $\mathbf{A} \in \mathbb{R}^{N_x \times N_y}$ ,  $\mathbf{y} \in \mathbb{R}^{N_y}$ ,  $\mathbf{z} \in \mathbb{R}$  とする。

$$p(\mathbf{x} | \mathbf{y}, z) = \mathcal{N}(\mathbf{z}\mathbf{A}\mathbf{y}, \sigma_x^2 \mathbf{I})$$

事前分布を

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{0}, \mathbf{C})$$

$$p(z) = \Gamma(k, \vartheta)$$

とする。  $\Gamma(k, \vartheta)$  はガンマ分布であり、 $k$  は形状 (shape) パラメータ、 $\vartheta$  は尺度 (scale) パラメータである。  $p(\mathbf{y})$  は  $\mathbf{y}$  の事前分布であり、刺激がない場合の自発活動の分布を表していると仮定する。

#### 重み行列 $\mathbf{A}$ の作成

```
using PyPlot, LinearAlgebra, Random, Distributions, KernelDensity, StatsBase
using PyPlot: matplotlib
Random.seed!(2)
rc("axes.spines", top=false, right=false)
```

```
function gabor(x, y, θ, σ=1, λ=2, ψ=0)
    xθ = x * cos(θ) + y * sin(θ)
    yθ = -x * sin(θ) + y * cos(θ)
    return exp(-.5(xθ^2 + yθ^2)/σ^2) * cos(2π/λ * xθ + ψ)
end;
```

```
function get_A(WH, Ny)
    Nx = WH^2
    A = zeros(Nx, Ny) # weight matrix
    p = range(-3, 3, length=WH) # position
    θ = (1:Ny) / Ny * π # theta for gabor
    for i in 1:Ny
        gb = gabor(p', p, θ[i])
        gb /= norm(gb) + 1e-8 # normalization
        A[:, i] = gb[:] # flatten and save
    end
    return A
end;
```

<sup>\*1</sup> コードは [?] <https://github.com/gergoorban/sampling-in-gsm>, および [?] [https://bitbucket.org/RSE\\_1987/ssn\\_inference\\_numerical\\_experiments/src/master/](https://bitbucket.org/RSE_1987/ssn_inference_numerical_experiments/src/master/) を参考に作成した。

```

WH = 16 # width/height of input image
Nx = WH^2 # dimension of the observed variable x
Ny = 50 # dimension of the hidden variable y

A = get_A(WH, Ny);

```

重み行列 **A** の一部を描画してみよう.

```

figure(figsize=(2,2))
plot_idx = [2,4,6,8]
weight_idx = [37,25,50,13]
titles = ["", "0°", "±90°", ""]
for i in 1:4
    subplot(3,3,plot_idx[i])
    title(titles[i])
    imshow(reshape(A[:, weight_idx[i]], WH, WH), cmap="gray")
    axis("off")
end
subplots_adjust(wspace=0.01, hspace=0.01)

```

### 分散共分散行列 **C** の作成

**C** は  $y$  の事前分布の分散共分散行列である. [?] では自然画像を用いて作成しているが, ここでは簡単のため **A** と同様に [?] に従って作成する. 前項で作成した通り, **A** の各基底には周期性があるため, 類似した基底を持つニューロン同士は類似した出力をすると考えられる. Echeveste らは  $\theta \in [-\pi/2, \pi/2]$  の範囲において Fourier 基底を複数作成し, そのグラム行列 (Gram matrix) を係数倍したものを **C** と設定している. ここではガウス過程 (Gaussian process) モデルとの類似性から, 周期カーネル (periodic kernel)

$$K(\theta, \theta') = \exp \left[ \phi_1 \cos \left( \frac{|\theta - \theta'|}{\phi_2} \right) \right]$$

を用いる. ここでは  $|\theta - \theta'| = m\pi$  ( $m = 0, 1, \dots$ ) の際に類似度が最大になればよいので,  $\phi_2 = 0.5$  とする. これが正定値行列になるように単位行列の係数倍  $\epsilon \mathbf{I}$  を加算し, スケーリングした上で, **Symmetric(C)** や **Matrix(Hermitian(C))** により実対象行列としたものを **C** とする. **C** を正定値行列にする理由は Julia の **MvNormal** が Cholesky 分解を用いて多変量正規分布の乱数を生成するためである. 事前に **cholesky(C)** が実行できるか確認するのもよい.

```

function get_C(Ny, C_range=[-0.5, 4.0], eps=0.1, ψ₁=2.0, ψ₂=0.5)
    K(x₁, x₂, ψ₁, ψ₂) = exp(ψ₁ * cos(abs(x₁-x₂) / ψ₂)) # periodic kernel
    θ = (1:Ny) / Ny * π # theta for gabor
    C = K.(θ', θ, ψ₁, ψ₂) # create covariance matrix
    C += eps * I # regularization to make C positive definite
    C_min, C_max = minimum(C), maximum(C)
    C = C_range[1] .+ (C_range[2]-C_range[1]) * (C .- C_min) / (C_max - C_min)
    return Symmetric(C); # make symmetric matrix using upper triangular matrix
end;

```

```

C = get_C(Ny)

figure(figsize=(3,2))
title(L"\mathbf{C}")
ims = imshow(C, origin="lower", cmap="bwr", vmin=-4, vmax=4, extent=(-90, 90, -90, 90))
xticks([-90,0,90]); yticks([-90,0,90]);
xlabel(L"\theta$ (Pref. ori)"); ylabel(L"\theta$ (Pref. ori)")
colorbar(ims);
tight_layout()

```

ここで Pref. ori は最適方位 (preferred orientation) を意味する.

## 事後分布の計算

事後分布は  $z$  と  $\mathbf{y}$  のそれぞれについて次のように求められる。

$$p(z | \mathbf{x}) \propto p(z) \mathcal{N}(0, z^2 \mathbf{A} \mathbf{C} \mathbf{A}^\top + \sigma_x^2 \mathbf{I})$$

$$p(\mathbf{y} | z, \mathbf{x}) = \mathcal{N}(\mu(z, \mathbf{x}), \Sigma(z))$$

ただし,

$$\Sigma(z) = \left( \mathbf{C}^{-1} + \frac{z^2}{\sigma_x^2} \mathbf{A}^\top \mathbf{A} \right)^{-1}$$

$$\mu(z, \mathbf{x}) = \frac{z}{\sigma_x^2} \Sigma(z) \mathbf{A}^\top \mathbf{x}$$

である。最終的な予測において  $z$  の事後分布は必要でないため、 $p(\mathbf{y} | z, \mathbf{x})$  から  $z$  を消去することを考えよう。厳密に行う場合、次式のように周辺化 (marginalization) により、 $z$  を (積分) 消去する必要がある。

$$p(\mathbf{y} | \mathbf{x}) = \int dz p(z | \mathbf{x}) \cdot p(\mathbf{y} | z, \mathbf{x})$$

周辺化においては、まず  $z$  の MAP 推定 (最大事後確率推定) 値  $z_{\text{MAP}}$  を求める。

$$z_{\text{MAP}} = \underset{z}{\operatorname{argmax}} p(z | \mathbf{x})$$

次に  $z_{\text{MAP}}$  の周辺で  $p(z | \mathbf{x})$  を積分し、積分値が一定の閾値を超える  $z$  の範囲を求め、この範囲で  $z$  を積分消去してやればよい。しかし、 $z$  は単一のスカラー値であり、この手法で推定するのは煩雑であるために近似手法が [?] において提案されている。Echeveste らは第一の近似として、 $z$  の分布を  $z_{\text{MAP}}$  でのデルタ関数に置き換える、すなわち、 $p(z | \mathbf{x}) \simeq \delta(z - z_{\text{MAP}})$  とすることを提案している。この場合、 $z$  は定数とみなせ、 $p(\mathbf{y} | \mathbf{x}) \simeq p(\mathbf{y} | \mathbf{x}, z = z_{\text{MAP}})$  となる。第二の近似として、 $z_{\text{MAP}}$  を真のコントラスト  $z^*$  で置き換えることが提案されている。GSM への入力  $\mathbf{x}$  は元の画像を  $\mathbf{x}$  とすると、 $\mathbf{x} = z^* \mathbf{x}$  としてスケールリングされる。この入力の前処理の際に用いる  $z^*$  を用いてしまおうということである。この場合、 $p(\mathbf{y} | \mathbf{x}) \simeq p(\mathbf{y} | \mathbf{x}, z = z^*)$  となる。しかし、入力を任意の画像とする場合、 $z^*$  は未知である。簡便さと精度のバランスを取り、ここでは第一の近似、 $z = z_{\text{MAP}}$  とする手法を用いることにする。

```
# log pdf of p(z)
log_Pz(z, k, theta) = logpdf.(Gamma(k, theta), z)

# pdf of p(z|x)
function Pz_x(z_range, x, ACA^T, sigma_x^2, k, theta)
    n_contrasts = length(z_range)
    log_p = zeros(n_contrasts)
    mu_xz = zeros(size(x))
    dz = z_range[2] - z_range[1]
    for i in 1:n_contrasts
        Cxz = z_range[i]^2 * ACA^T + sigma_x^2 * I
        log_p[i] = log_Pz(z_range[i], k, theta) + logpdf(MvNormal(mu_xz, Symmetric(Cxz)), x)
    end
    p = exp.(log_p .- maximum(log_p)) # for numerical stability
    p /= sum(p) * dz
    return p
end;
```

```
# mean and covariance matrix of p(y|x, z)
function post_moments(x, z, sigma_x^2, A, A^T A, C^-1)
    Sigma_z = inv(C^-1 + (z^2 / sigma_x^2) * A^T A)
    mu_xz = (z / sigma_x^2) * Sigma_z * A^T * x
    return mu_xz, Sigma_z
end;
```

## シミュレーション

```

ATA = A' * A
ACAT = A * C * A'

σx = 1.0 # Noise of the x process
σx2 = σx2
k, θ = 2.0, 2.0 # Parameter of the gamma dist. for z (Shape, Scale)

C-1 = inv(C); # inverse of C

```

## 入力データの作成

```

Z = [0.0, 0.25, 0.5, 1.0, 2.0] # true contrasts z^*
n_samples = size(Z)[1]
y = rand(MvNormal(zeros(Ny), C), 1) # sampling from P(y)=N(0, C)
X = hcat([rand(MvNormal(vec(z*A*y), σx*I)) for z in Z]...)'

x_min, x_max = minimum(X), maximum(X)

figure(figsize=(4,2))
for s in 1:n_samples
    subplot(1, n_samples, s)
    title(L"$z$: " * string(Z[s]))
    imshow(reshape(X[s, :], WH, WH), vmin=x_min, vmax=x_max, cmap="gray")
    axis("off")
end
tight_layout()

```

## 事後分布の計算をする.

```

μ_post = zeros(n_samples, Ny)
σ_post = zeros(n_samples, Ny)
Σ_post = zeros(n_samples, Ny, Ny)

z_range = range(0, 5.0, length=100) # range of z for MAP estimation
Z_MAP = zeros(n_samples)

for s in 1:n_samples
    p_z = Pz_x(z_range, X[s, :], ACAT, σx2, k, θ)
    Z_MAP[s] = z_range[argmax(p_z)] # MAP estimated z
    μ_post[s, :], Σ_post[s, :, :] = post_moments(X[s, :], Z_MAP[s], σx2, A, ATA, C-1)
    σ_post[s, :] = sqrt.(diag(Σ_post[s, :, :]))
end

```

## 結果

```

θs = range(-90, 90, length=Ny)
cm = get_cmap(:Greens) # get color map
cms = cm.((1:n_samples)/n_samples) # color list

fig, ax = subplots(1, 3, figsize=(7.5, 2))
ax[1].scatter(Z, Z_MAP, c=cms)
ax[1].plot(Z, Z_MAP, color="tab:gray", zorder=0)
ax[1].set_xlabel(L"$z$"); ax[1].set_ylabel(L"$z_{MAP}$");
for s in 1:n_samples
    ax[2].plot(θs, μ_post[s, :], color=cms[s])

```

```

        ax[3].plot(θs, σ_post[s, :], color=cms[s], label=L"$z$ : "*string(Z[s]))
    end
    ax[2].set_ylabel(L"$\mu$"); ax[3].set_ylabel(L"$\sigma$")
    for i in 2:3
        ax[i].set_xticks([-90,0,90])
        ax[i].set_xlabel(L"$\theta$ (Pref. ori)")
    end
    ax[3].legend(bbox_to_anchor=(1.05, 1), loc="upper left", borderaxespad=0)
    tight_layout()

```

```

fig, ax = subplots(1, n_samples, figsize=(7.5, 1), sharex="all", sharey="all")
for s in 1:n_samples
    ax[s].set_title(L"$z$ : "*string(Z[s]))
    ims = ax[s].imshow(Σ_post[s, :, :], origin="lower", cmap="bwr", extent=(-90, 90, -90, 90), vmin=-1, vmax=1)
    ax[s].set_xticks([-90,0,90]); ax[s].set_yticks([-90,0,90]);
    if s == 1
        ax[s].set_ylabel(L"$\theta$ (Pref. ori)")
    elseif s == ceil(Int, n_samples/2)
        ax[s].set_xlabel(L"$\theta$ (Pref. ori)");
    end
end
fig.colorbar(ims, ax=ax[n_samples]);

```

### 出力のサンプリング

```

membrane_potential(y, α=2.4, β=1.9, γ=0.6) = α * max(0, y+β)^γ

```

事後分布から応答をサンプリングする.

```

nt = 1000
h_gsm = zeros(n_samples, Ny, nt)
for s in 1:n_samples
    μ = μ_post[s, :]
    Σ = Σ_post[s, :, :]
    sample = rand(MvNormal(μ, Symmetric(Σ)), nt)
    h_gsm[s, :, :] = membrane_potential.(sample)
end

```

```

# modified from https://matplotlib.org/stable/gallery/statistics/confidence_ellipse.html
function confidence_ellipse(x, y, ax, n_std=3, alpha=1, facecolor="none", edgecolor="tab:gray")
    pearson = cor(x,y)
    rx, ry = sqrt(1 + pearson), sqrt(1 - pearson)
    ellipse = matplotlib.patches.Ellipse((0, 0), width=2*rx, height=2*ry, alpha=alpha, fc=facecolor, ec=edgecolor, lw=2, zorder=0)
    scales = [std(x), std(y)] * n_std
    means = [mean(x), mean(y)]
    transf = matplotlib.transforms.Affine2D().rotate_deg(45).scale(scales...).translate(means...)
    ellipse.set_transform(transf + ax.transData)
    return ax.add_patch(ellipse)
end;

```

```

fig, ax = subplots(figsize=(4, 3))
unit_idx = [1, 25]
for s in 1:n_samples

```



```

    h1, h2 = h_gsm[s, unit_idx[1], :], h_gsm[s, unit_idx[2], :]
    ax.plot(h1[1:15], h2[1:15], marker="o", markersize=5, alpha=0.5, color=cms[s],
            label=L"$z$ : " + string(Z[s]))
    confidence_ellipse(h1, h2, ax, 3, 1, "none", cms[s])
end
ax.set_xlabel("Neuron #" + string(unit_idx[1])); ax.set_ylabel("Neuron #" + string(unit_idx[2]))
axins = [ax.inset_axes([0.85, -0.25, 0.15, 0.15]), ax.inset_axes([-0.3, 0.85, 0.15, 0.15])]
for i in 1:2
    axins[i].imshow(reshape(A[:, unit_idx[i]], WH, WH), cmap="gray")
    axins[i].axis("off")
end
ax.set_aspect("equal", "box")
ax.legend(bbox_to_anchor=(1.05, 1), loc="upper left", borderaxespad=0)
tight_layout()

```

### 0.1.2 興奮性・抑制性神経回路によるサンプリング

前節で実装した MCMC を興奮性・抑制性神経回路 (excitatory-inhibitory [E-I] network) で実装する。HMC と LMC の両方を神経回路で実装する。Hamiltonian を用いる場合、一般化座標  $\mathbf{q}$  を興奮性神経細胞の活動  $\mathbf{u}$ 、一般化運動量  $\mathbf{p}$  を抑制性神経細胞の活動  $\mathbf{v}$  に対応させる。\*ToDo: 詳しい説明\* 簡単のため、前項で用いた入力刺激のうち、最も  $z$  が大きいサンプルのみを使用する。

```

dt = 1e-2 # ms
tau, tau_l = 10.0, 150.0 # ms
alpha_in = [1/tau - 1/tau_l, 1/tau + 1/tau_l]
alpha_ext = [1/tau_l, -1/tau]
rho = sqrt(2*dt/tau_l);

nt = 50000
M = cat(ones(1,1), C; dims=(1,2));
x_idx = n_samples # get last x
x = X[x_idx, :]
u_init = [1; zeros(Ny)];

```

```

function nabla_u_logP(u, x, sigma_x^2, A, C^-1)
    z, y = abs(u[1]), u[2:end]
    pred_error = A' * (x - z*A*y) / sigma_x^2 # prediction error signal
    du = zeros(size(u))
    du[1] = sign(u[1]) * (y' * pred_error - z)
    du[2:end] = z * pred_error - C^-1*y
    return du
end

```

```

nabla_log_p(u) = nabla_u_logP(u, x, sigma_x^2, A, C^-1);

```

```

function NeuralLMC(nabla_log_p::Function, u_init::Vector{Float64}, alpha::Float64, rho::Float64,
    dt::Float64, nt::Int)
    d = length(u_init)
    u = zeros(nt, d)
    u[1, :] = u_init

    for t in 1:nt-1
        I_ext = nabla_log_p(u[t, :]) # external input
        u[t+1, :] = u[t, :] + dt * (alpha * I_ext) + rho * randn(d)
    end
    return u
end

```

```

end

function NeuralHMC(∇log_p::Function, u_init::Vector{Float64}, M::Matrix{Float64},
    α_in::Vector{Float64}, α_ext::Vector{Float64}, ρ::Float64, dt::Float64, nt::Int)
    d = length(u_init)
    u, v = zeros(nt, d), zeros(nt, d)
    u[1, :] = u_init

    for t in 1:nt-1
        I_ext = ∇log_p(u[t, :]) # external input
        I_in = M * (u[t, :] - v[t, :]) # internal input
        u[t+1, :] = u[t, :] + dt * (α_in[1] * I_in + α_ext[1] * I_ext) + ρ * randn(d)
        v[t+1, :] = v[t, :] + dt * (α_in[2] * I_in + α_ext[2] * I_ext) + ρ * randn(d)
    end
    return u, v
end;

```

```
@time u_nlmc = NeuralLMC(∇log_p, u_init, α_ext[1], ρ, dt, nt);
```

```
@time u_nhmc, v_nhmc = NeuralHMC(∇log_p, u_init, M, α_in, α_ext, ρ, dt, nt);
```

初めの 100ms は burn-in 期間として除く。またダウンサンプリングする。

```

L = 100
burn_in = 10000
mcmc_time = (burn_in*dt):(L*dt):(nt*dt); # time for plot

```

```

mean_nlmc = mean(u_nlmc[burn_in:L:end, 2:end], dims=2); # pseudo-LFP
mean_nhmc = mean(u_nhmc[burn_in:L:end, 2:end], dims=2);

autocorr_nlmc = autocor(mean_nlmc, 1:length(mean_nlmc)-1);
autocorr_nhmc = autocor(mean_nhmc, 1:length(mean_nhmc)-1);

```

$z$  の推定過程を描画する。また、 $z$  を除いた  $u$  を平均化し、自己相関の度合いを確認する。

```

fig, ax = subplots(1,2,figsize=(6,2.5),sharex="all")
ax[1].plot(mcmc_time, u_nhmc[burn_in:L:end, 1], color="tab:red")
ax[1].plot(mcmc_time, u_nlmc[burn_in:L:end, 1], color="tab:blue")
ax[1].axhline(Z[x_idx], linestyle="dashed", color="tab:gray", alpha=0.5)
ax[1].set_xlabel("Time (ms)"); ax[1].set_ylabel(L"Estimated $z$")
ax[1].set_xlim(mcmc_time[1], mcmc_time[end])

ax[2].plot(mcmc_time[1:end-1], autocorr_nhmc, color="tab:red", label="Hamiltonian")
ax[2].plot(mcmc_time[1:end-1], autocorr_nlmc, color="tab:blue", label="Langevin")
ax[2].set_xlabel("Time (ms)"); ax[2].set_ylabel("Autocorrelation")
ax[2].set_xlim(mcmc_time[1], mcmc_time[end])
ax[2].axhline(0, linestyle="dashed", color="tab:gray", alpha=0.5)
ax[2].legend()
fig.tight_layout()

```

Hamiltonian ネットワークは自己相関を振動により低下させることで、効率の良いサンプリングを実現している。\*ToDo: 普通に MCMC やる場合も自己相関は確認したほうがいいという話をどこかに書く。\* 推定された事後分布を特定の神経細胞のペアについて確認する。

```

h_nhmc = membrane_potential.(u_nhmc[burn_in:L:end, :])
h_nlmc = membrane_potential.(u_nlmc[burn_in:L:end, :])

```

```
kde_bound = ((3,7),(3,7)) # ((xlo,xhi),(ylo,yhi))
U_gsm = kde((h_gsm[x_idx, unit_idx[1], :], h_gsm[x_idx, unit_idx[2], :]), μ
            boundary=kde_bound)
U_nhmc = kde((h_nhmc[:, unit_idx[1]+1], h_nhmc[:, unit_idx[2]+1]), boundary=kde_bound)
U_nlmc = kde((h_nlmc[:, unit_idx[1]+1], h_nlmc[:, unit_idx[2]+1]), boundary=kde_bound);
```

```
fig, ax = plt.subplots(1,3, figsize=(6, 2.5), sharey="all", sharex="all")
ax[1].contourf(U_gsm.x, U_gsm.x, U_gsm.density)
ax[1].set_title("Actual")
ax[2].contourf(U_nhmc.x, U_nhmc.x, U_nhmc.density)
ax[2].set_title("Hamiltonian")
ax[3].contourf(U_nlmc.x, U_nlmc.x, U_nlmc.density)
ax[3].set_title("Langevin")
ax[1].set_ylabel("Neuron #" + string(unit_idx[2]))
ax[2].set_xlabel("Neuron #" + string(unit_idx[1]))
fig.tight_layout()
```

Hamiltonian ネットワークの方が安定して事後分布を推定することができている。\*ToDo: 以下の記述\* - ここでは重みを設定したが, [?] では RNN に BPTT で重みを学習させている。 - 動的な入力に対するサンプリング [?]. burn-in がなくなり効率良くサンプリングできる。

### 0.1.3 Spiking ニューラルネットワークにおけるサンプリング

前項で挙げた例は発火率モデルであったが, SNN においてサンプリングを実行する機構自体は考案されている。\*ToDo: 以下の記述\* - [?] - [?] - [?]

### 0.1.4 シナプスサンプリング

ここまでシナプス結合強度は変化せず, 神経活動の変動によりサンプリングを行うというモデルについて考えてきた。一方で, シナプス結合強度自体が短時間で変動することによりベイズ推論を実行するというモデルがあり, シナプスサンプリング (synaptic sampling) と呼ばれる。\*ToDo: 以下の記述\* - [?] - [?]

### 0.1.5 シナプスサンプリング

ここまでシナプス結合強度は変化せず, 神経活動の変動によりサンプリングを行うというモデルについて考えてきた。一方で, シナプス結合強度自体が短時間で変動することによりベイズ推論を実行するというモデルがあり, シナプスサンプリング (synaptic sampling) と呼ばれる。\*ToDo: 以下の記述\* - [?] - [?]