1. 指数分布族

对于指数分布族

$$\rho(\theta;a) = h(\theta)\exp\{T(\theta)\cdot a - A(a)\}$$
$$0 = \int \nabla_a \rho(\theta;a)d\theta = \mathbb{E}_\rho[T(\theta) - \nabla_a A(a)]$$

Here we introduce another parameter vector λ. Secondly, the Fisher information matrix FIM is

$$\begin{aligned}FIM(a) &= \mathbb{E}_\rho[\nabla_a \log\rho(\theta;a)^T \nabla_a \log\rho(\theta;a)] \\ &= \int \nabla_a\rho(\theta;a)\big[T(\theta) - \nabla_a A(a)\big]d\theta \\ &= \int \rho(\theta;a)\nabla_a\nabla_a A(a)d\theta \quad \text{Using } \nabla_a \int \rho(\theta;a)\big[T(\theta)-\nabla_a A(a)\big]d\theta = 0 \\ &= \nabla_a\nabla_a A(a)\end{aligned}$$

2. 高斯变分推理

对于高斯分布

$$\rho_a = \frac{1}{(2\pi)^{N_\theta/2}\sqrt{|C|}}\exp\{-\frac{1}{2}(\theta-m)^T C^{-1}(\theta-m)\}$$

我们有

$$\begin{aligned}\nabla_m\rho_a &= \rho_a C^{-1}(\theta-m) = -\nabla_\theta\rho_a \\ \nabla_C\rho_a &= \rho_a\Big(-\frac{1}{2}C^{-1} + \frac{1}{2}C^{-1}(\theta-m)(\theta-m)^T C^{-1}\Big) = \frac{1}{2}\nabla_\theta\nabla_\theta\rho_a\end{aligned}$$

我们用了

$$\nabla_C|C| = |C|C^{-T} \qquad \nabla_C x^T C^{-1}x = -C^{-1}xx^TC^{-1}$$

对于高斯变分推理，我们有

$$\begin{aligned}\nabla_m KL[\rho_a\|\rho^*] &= \int \nabla_m\rho_a(\log(\rho_a) + \Phi_R)d\theta \\ &= -\int \nabla_\theta\rho_a(\log(\rho_a)+\Phi_R)d\theta \\ &= \int \rho_a\nabla_\theta(\log(\rho_a)+\Phi_R)d\theta \\ &= \int \nabla_\theta\rho_a + \rho_a\nabla_\theta\Phi_R d\theta \\ &= \int \rho_a\nabla_\theta\Phi_R d\theta \\ &= \mathbb{E}_{\rho_a}[\nabla_\theta\Phi_R] \\ \nabla_C KL[\rho_a\|\rho^*] &= \int \nabla_C\rho_a(\log(\rho_a)+\Phi_R)d\theta \\ &= \int \frac{1}{2}\nabla_\theta\nabla_\theta\rho_a(\log(\rho_a)+\Phi_R)d\theta \\ &= \int \frac{1}{2}\rho_a\nabla_\theta\nabla_\theta(\log(\rho_a)+\Phi_R)d\theta \\ &= \int \frac{1}{2}\rho_a(-C) + \frac{1}{2}\rho_a\nabla_\theta\nabla_\theta\Phi_R d\theta \\ &= -\frac{1}{2}C + \frac{1}{2}\mathbb{E}_{\rho_a}[\nabla_\theta\nabla_\theta\Phi_R]\end{aligned}$$

3. 自然梯度下降方法

我们首先推导高斯的Fisher信息矩阵 我们有

$$\begin{aligned}\int \frac{\nabla_m\rho_a\nabla_m\rho_a^T}{\rho_a}d\theta &= \int \rho_a C^{-1}(\theta-m)(\theta-m)^T C^{-1}d\theta d\theta = C^{-1} \\ \int \frac{\nabla_m\rho_a\nabla_C\rho_a^T}{\rho_a}d\theta &= \int \rho_a C^{-1}(\theta-m)\otimes\Big(-\frac{1}{2}C^{-1} - \frac{1}{2}C^{-1}(\theta-m)(\theta-m)^TC^{-1}\Big)d\theta = 0\end{aligned}$$

对于协方差项，我们定义

$$\begin{aligned}X :=& \frac{1}{4}\int \rho_a\Big(C^{-1}(\theta-m)(\theta-m)^TC^{-1} - C^{-1}\Big)\otimes\Big(C^{-1}(\theta-m)(\theta-m)^TC^{-1} - C^{-1}\Big)d\theta \\ =& \frac{1}{4}\int \mathcal{N}(y;0,I)C^{-1/2}\Big(yy^T - I\Big)C^{-1/2}\otimes C^{-1/2}\Big(yy^T-I\Big)C^{-1/2}dy, \text{ where } y = C^{-1/2}(\theta-m).\end{aligned}$$

它满足

$$\begin{aligned}X[ij,lm] =& \frac{1}{4}\sum_{r,s,p,q} C^{-1/2}[i,r]C^{-1/2}[j,s]C^{-1/2}[l,p]C^{-1/2}[m,q]\int(y_ry_s - \delta_{r,s})(y_py_q - \delta_{p,q})\mathcal{N}(y;0,I)dy \\ =& \frac{1}{4}\sum_{r,s,p,q} C^{-1/2}[i,r]C^{-1/2}[j,s]C^{-1/2}[l,p]C^{-1/2}[m,q]\big(\delta_{r,p}\delta_{s,q} + \delta_{r,q}\delta_{s,p}\big) \\ =& \frac{1}{4}\Big(C^{-1}[i,l]C^{-1}[j,m] + C^{-1}[i,m]C^{-1}[j,l]\Big).\end{aligned}$$

因此

$$\begin{aligned}(XY)_{ij} &= \sum_{l,m} X[ij,lm]Y[l,m] \\ &= \frac{1}{4}\sum_{l,m}\Big(C^{-1}[i,l]Y[l,m]C^{-1}[j,m] + C^{-1}[i,m]Y[l,m]C^{-1}[j,l]\Big) \\ &= \frac{1}{4}(C^{-1}YC^{-1} + C^{-1}Y^TC^{-1})_{ij}.\end{aligned}$$

我们用了

由于

$$\frac{\mathrm{d}C_t^{-1}}{\mathrm{d}t} = -C_t^{-1}\frac{\mathrm{d}C_t}{\mathrm{d}t}C_t^{-1}$$

自然梯度下降法可以写成

$$\begin{aligned}\frac{\mathrm{d}m_t}{\mathrm{d}t} &= C_tC^{*^{-1}}(m^* - m_t), \\ \frac{\mathrm{d}C_t^{-1}}{\mathrm{d}t} &= -C_t^{-1} + C^{*^{-1}}.\end{aligned}$$

对于协方差，我们有

$$C_t^{-1} = (1-e^{-t})C^{*^{-1}} + e^{-t}C_0^{-1}.$$

对于期望，我们有

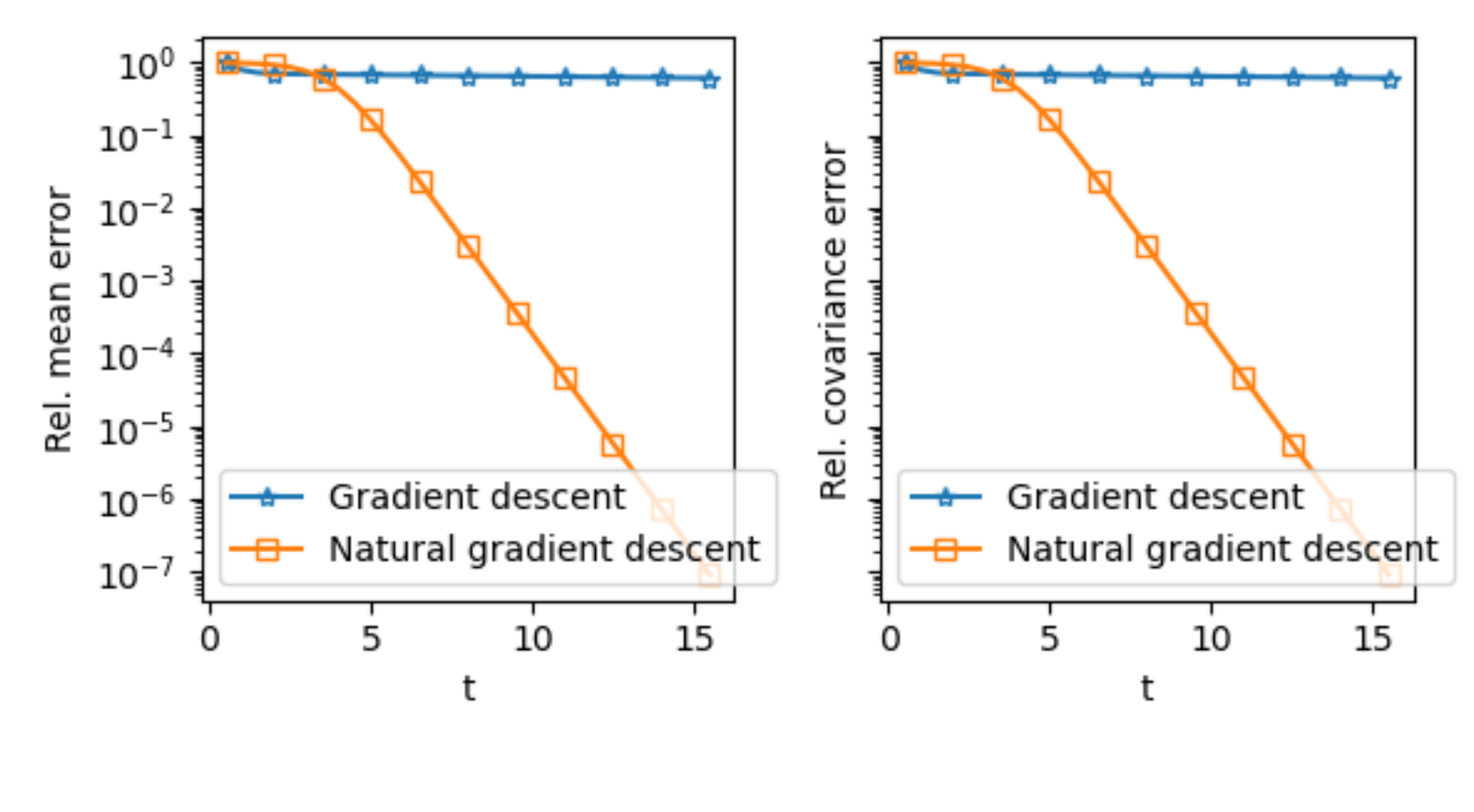
$$m^* - m_t = e^{-t}C_tC_0^{-1}(m^* - m_0).$$

计算它的导数，我们能得到

$$\begin{aligned}\frac{\mathrm{d}m_t}{\mathrm{d}t} &= -e^{-t}C_tC_0^{-1}(m_0 - m^*) + e^{-t}\frac{\mathrm{d}C_t}{\mathrm{d}t}C_0^{-1}(m_0 - m^*) \\ &= -e^{-t}C_tC_0^{-1}(m_0-m^*) + e^{-t}(C_t - C_tC^{*^{-1}}C_t)C_0^{-1}(m_0 - m^*) \\ &= C_tC^{*^{-1}}e^{-t}C_tC_0^{-1}(m^* - m_0) \\ &= C_tC^{*^{-1}}(m^* - m_t).\end{aligned}$$

In [1]:

```
using LinearAlgebra
using PyPlot
function expectation(method_type::String, m_oo, C_oo, m, C)
    return C_oo\(m - m_oo), inv(C_oo)
    if method_type == "gradient_descent"
        # under-determined case
        return -C_oo\(m - m_oo), 1/2.0*(inv(C) - inv(C_oo))

    elseif method_type == "natural_gradient_descent"
        # over-determined case
        return -(m - m_oo), C - C*(C_oo\C)

    elseif method_type == "natural_gradient_descent_Cinv"
        # over-determined case
        return -(m - m_oo), -inv(C) + C_oo

    elseif method_type == "wasserstein_gradient_descent"
        # over-determined case
        return -C_oo\(m - m_oo), 2I - C_oo\C - C/C_oo

    else
        error("Problem type : ", problem_type, " has not implemented!")
    end
end

function Continuous_Dynamics(method_type::String, m_oo, C_oo, m_0, C_0, Δt, N_t)

    N_θ = length(m_0)
    m = zeros(N_t+1, N_θ)
    C = zeros(N_t+1, N_θ, N_θ)

    m[1, :] = m_0
    C[1, :, :] = C_0

    for i = 1:N_t
        E∇Φ, E∇∇Φ= expectation(method_type, m_oo, C_oo, m[i, :], C[i, :, :])
        if method_type == "gradient_descent"
            m[i+1, :]    = m[i, :] - E∇Φ * Δt
            C[i+1, :, :] = C[i, :, :] + 1/2.0*(inv(C[i, :, :]) - E∇∇Φ) * Δt
        elseif method_type == "natural_gradient_descent"
            m[i+1, :]    = m[i, :] - C[i, :, :]*E∇Φ * Δt
            C[i+1, :, :] = inv( inv(C[i, :, :]) + (E∇∇Φ - inv(C[i, :, :])) * Δt)
        else
            error("Problem type : ", problem_type, " has not implemented!")
        end
    end

    return m, C
end


Δt, N_t = 5e-1, 30
fig, ax = PyPlot.subplots(ncols=2, sharex=true, sharey=true, figsize=(6,3))


m_oo = [1; 1]
C_oo = [100.0 0; 0 1.0]

m_0 = [0.0; 0.0]
C_0 = [1.0 0; 0 1.0]

m_gd, C_gd   = Continuous_Dynamics("gradient_descent", m_oo, C_oo, m_0, C_0, Δt, N_t)
m_ngd, C_ngd = Continuous_Dynamics("natural_gradient_descent", m_oo, C_oo, m_0, C_0, Δt, N_t)

e_gd, e_ngd  = zeros(N_t+1, 2),  zeros(N_t+1, 2)


for i = 1:N_t+1

    e_gd[i, 1]  = ( norm(m_gd[i,:] - m_oo) / norm(m_oo) )
    e_gd[i, 2]  = ( norm(C_gd[i,:,:] - C_oo) / norm(C_oo) )

    e_ngd[i, 1] = ( norm(m_ngd[i,:] - m_oo) / norm(m_oo) )
    e_ngd[i, 2] = ( norm(C_ngd[i,:,:] - C_oo) / norm(C_oo) )
end

ts = Array(1:N_t+1)*Δt
ax[1].semilogy(ts, e_gd[:, 1],  "-*", fillstyle="none", markevery=div(N_t, 10), label="Gradient descent")
ax[1].semilogy(ts, e_ngd[:, 2], "-s", fillstyle="none", markevery=div(N_t, 10), label="Natural gradient descent")
ax[2].semilogy(ts, e_gd[:, 1],  "-*", fillstyle="none", markevery=div(N_t, 10), label="Gradient descent")
ax[2].semilogy(ts, e_ngd[:, 2], "-s", fillstyle="none", markevery=div(N_t, 10), label="Natural gradient descent")

ax[1].legend()
ax[2].legend()
ax[1].set_ylabel("Rel. mean error")
ax[2].set_ylabel("Rel. covariance error")
ax[1].set_xlabel("t")
ax[2].set_xlabel("t")
fig.tight_layout()
fig.savefig("Gaussian-VI.pdf")
```



4. Langevin 动力系统

考虑随机动力系统

$$d\theta_t = f(\theta_t, t)dt + \sqrt{g}dW_t$$

它对应的 描述 $\theta_t \sim \rho_t$ 的 Fokker-Planck 方程是

$$\partial_t\rho_t = -\nabla\big(f(x,t)\rho_t\big) + \frac{1}{2}\nabla\cdot\nabla\cdot(g\rho_t)$$

使用 Itô's 公式，我们有

$$\begin{aligned}dh(\theta_t) &= \nabla h(\theta_t)^T(f(\theta_t,t)dt + \sqrt{g}dW_t) + \frac{1}{2}g\nabla^2h(\theta_t)dt \\ \partial_t\mathbb{E}[h(\theta_t)] &= \mathbb{E}\Big[f(\theta_t,t)\nabla h(\theta_t) + \frac{g}{2}\nabla^2h(\theta_t)\Big] \\ \int h(\theta)\partial_t\rho_t &= \int \rho_tf(\theta,t)\nabla h(\theta) + \frac{g}{2}\rho_t\nabla^2h(\theta)d\theta \\ &= \int h(\theta)\nabla(-\rho_tf(\theta,t)) + h(\theta)\frac{1}{2}\nabla\cdot\nabla\cdot(g\rho_t)d\theta\end{aligned}$$

我们可以取

$$f = -\nabla_\theta\Phi_R \qquad g = 2$$

5. 仿射不变性

给定可逆仿射变换 $\tilde{\theta} = \mathcal{T}\theta = A\theta + b$,

$$\begin{aligned}&\tilde{\rho}(\tilde{\theta}) = \rho(\theta)|A^{-1}| \qquad \tilde{\rho}_t(\tilde{\theta}) = \rho_t(\theta)|A^{-1}| \qquad \tilde{C}_t = AC_tA^T \\ &\nabla_{\tilde{\theta}}f(\tilde{\theta}) = A^{-T}\nabla_\theta f(\theta) \qquad \nabla_{\tilde{\theta}}\cdot v(\tilde{\theta}) = \nabla_\theta\cdot(A^{-1}v(\theta))\end{aligned}$$

带入Kalman-Waserstein梯度流

$$\begin{aligned}&\frac{\partial\tilde{\rho}_t}{\partial t} = \nabla_{\tilde{\theta}}\cdot\Big[\tilde{\rho}_t\tilde{C}_t(\nabla_{\tilde{\theta}}\log\tilde{\rho}^* - \nabla_{\tilde{\theta}}\log\tilde{\rho}_t)\Big] \\ \Longleftrightarrow &\frac{\partial\rho_t}{\partial t}|A^{-1}| = \nabla_{\tilde{\theta}}\cdot\Big[\rho_tAC_tA^T(\nabla_{\tilde{\theta}}\log\tilde{\rho}^* - \nabla_{\tilde{\theta}}\log\tilde{\rho}_t)\Big]|A^{-1}| \\ \Longleftrightarrow &\frac{\partial\rho_t}{\partial t}|A^{-1}| = \nabla_\theta\cdot\Big[\rho_tC_t(\nabla_\theta\log\tilde{\rho}^* - \nabla_\theta\log\tilde{\rho}_t)\Big]|A^{-1}| \\ \Longleftrightarrow &\frac{\partial\rho_t}{\partial t} = \nabla_\theta\cdot\Big[\rho_tC_t(\nabla_\theta\log\rho^* - \nabla_\theta\log\rho_t)\Big]\end{aligned}$$

6. 练习

In [2]:

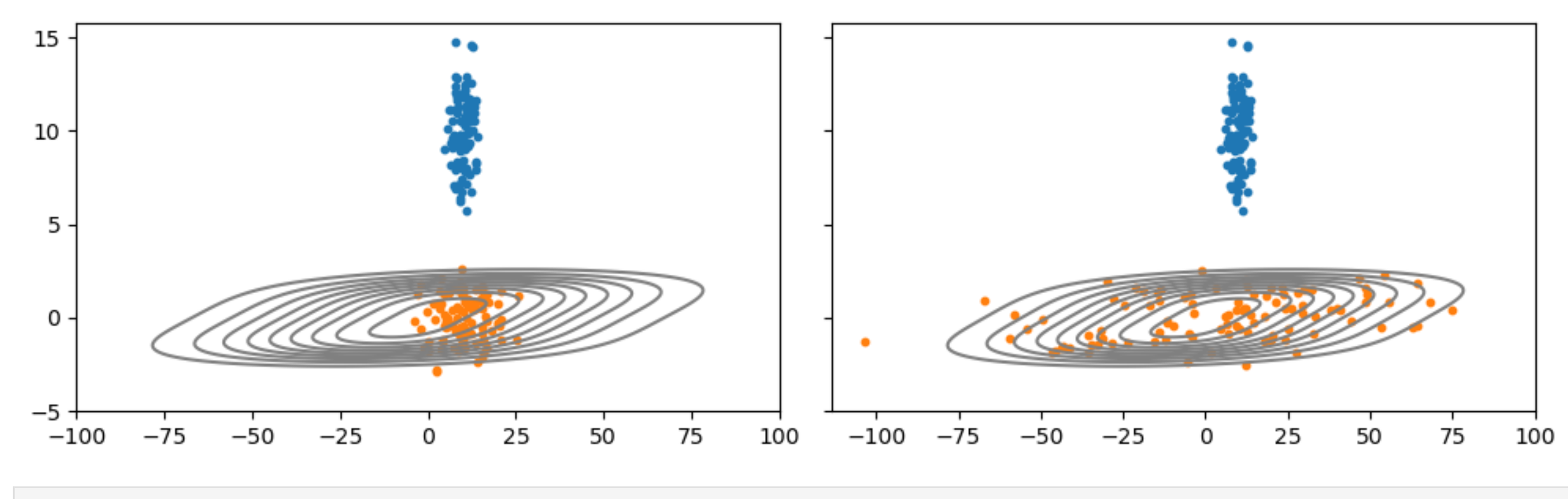
```
using PyPlot, Random, Statistics, Distributions, ForwardDiff, LinearAlgebra

Random.seed!(42)

function Phi_R_Convex(θ)
    ϵ = 0.1
    θ₁, θ₂ = θ
    return ( (ϵ*θ₁ - θ₂)^2 + θ₂^4)/20
end


ΦR(θ) = Phi_R_Convex(θ)
function ∇ΦR(θ)
    return ΦR(θ), ForwardDiff.gradient(ΦR, θ)
end


function update_ensemble(θ, Δt, func_neg_log_rho::Function,  preconditioner::Bool)
    N_θ, N_ens = size(θ)

    Φ_R, ∇Φ_R = zeros(N_ens), zeros(N_θ, N_ens)
    for i = 1:N_ens
        Φ_R[i], ∇Φ_R[:,i] = func_neg_log_rho(θ[:, i])
    end

    θ_mean = mean(θ, dims=2)
    θθ_cov = ((θ .- θ_mean) * (θ .- θ_mean)') / N_ens
    Prec = (preconditioner ? θθ_cov : 1)
    σ = (preconditioner ? cholesky(Hermitian(θθ_cov)).L : 1)

    noise = rand(Normal(0, 1), (N_θ, N_ens))
    θ = θ - Δt*Prec*∇Φ_R + sqrt(2*Δt)*(σ*noise)

    return θ
end
```

Out[2]:

```
update_ensemble (generic function with 1 method)
```

In [3]:

```
N_θ = 2
m_0 = [10.0; 10.0]
σ0 = 2
C_0 = [σ0^2  0.0; 0.0  σ0^2]



Lx , Ux = -100.0,  100.0
Ly , Uy = -5.0,  5.0
N = 500
X = zeros(N, N)
Y = zeros(N, N)
ρ = zeros(N, N)
for i = 1:N
    for j = 1:N
        X[i, j], Y[i, j] = Lx + (Ux - Lx) * (i-1)/(N-1), Ly + (Uy - Ly) * (j-1)/(N-1)
        ρ[i, j] = Phi_R_Convex([X[i, j]; Y[i, j]])
    end
end
Z = sum(ρ)
ρ .= exp.(-ρ)/Z

fig, ax = PyPlot.subplots(ncols=2, nrows=1, sharex="col", sharey=true, figsize=(10,3))
ax[1].contour(X, Y, ρ, 10, colors="grey")
ax[2].contour(X, Y, ρ, 10, colors="grey")


N_ens = 100
θ0 = Array(rand(MvNormal(m_0, C_0), N_ens))
θ = zeros(N_θ, N_ens)
Δt = 0.01
N_t = 2000



for preconditioner = 1:2
    θ .= θ0
    for i = 1:N_t
        θ = update_ensemble(θ, Δt, ∇ΦR,  preconditioner == 2)
    end

    ax[preconditioner].scatter(θ0[1, :], θ0[2, :], s = 10)
    ax[preconditioner].scatter(θ[1, :], θ[2, :], s = 10)


end

fig.tight_layout()
fig.savefig("GF.png")
```



In [ ]: