1. 集合卡尔曼滤波

卡尔曼方法满足

$$
\begin{aligned}
m_{n+1} &= \hat{m}_{n+1} + \hat{C}^{xy}_{n+1}(\hat{C}^{yy}_{n+1})^{-1}(y_{n+1} - \hat{y}_{n+1}) \\
C_{n+1} &= \hat{C}_{n+1} - \hat{C}^{xy}_{n+1}(\hat{C}^{yy}_{n+1})^{-1}\hat{C}^{xy^T}_{n+1}
\end{aligned}
$$

基于卡尔曼方法，我们设计了集合卡尔曼滤波的分析步：

$$
x^j_{n+1} = \hat{x}^j_{n+1} + \hat{C}^{xy}_{n+1}(\hat{C}^{yy}_{n+1})^{-1}(y_{n+1} - y^j_{n+1} - \eta^j_{n+1})
$$

当 $\mathcal{H}(x) = Hx$ 时，我们有$\hat{C}^{xy}_{n+1} = \hat{C}_{n+1}H^T$， $\hat{C}^{yy}_{n+1} = H\hat{C}_{n+1}H^T + \Sigma_\eta$ 和 $\hat{y}^j_{n+1} = H\hat{x}_{n+1}$， 滤波分析步可以写为

$$
\begin{aligned}
x^j_{n+1} &= \hat{x}^j_{n+1} + \hat{C}_{n+1}H^T(H\hat{C}_{n+1}H^T + \Sigma_\eta)^{-1}(y_{n+1} - H\hat{x}^j_{n+1} - \eta^j_{n+1}) \\
&= \hat{x}^j_{n+1} + (\hat{C}^{-1}_{n+1} + H\Sigma^{-1}_\eta H^T)^{-1}H^T\Sigma^{-1}_\eta(y_{n+1} - H\hat{x}^j_{n+1} - \eta^j_{n+1}) \\
&= (\hat{C}^{-1}_{n+1} + H\Sigma^{-1}_\eta H^T)^{-1}(\hat{C}^{-1}_{n+1}\hat{x}^j_{n+1} + H^T\Sigma^{-1}_\eta(y_{n+1} - \eta^j_{n+1}))
\end{aligned}
$$

因而对应了最优化问题

$$
\operatorname{argmin}_{x_{n+1}} \frac{1}{2}\|\hat{C}^{-1/2}_{n+1}(x_{n+1} - \hat{x}_{n+1})\|^2 + \frac{1}{2}\|\hat{\Sigma}^{-1/2}_\eta(y_{n+1} - Hx_{n+1} - \eta^j_{n+1})\|^2
$$

集合卡尔曼滤波的误差估计

我们定义

$$\hat{m}_{n+1} = \frac{1}{J}\sum_{j=1}^{J}\hat{x}_{n+1}^{j} \qquad m_{n+1} = \frac{1}{J}\sum_{j=1}^{J}x_{n+1}^{j} \qquad \hat{y}_{n+1}^{j} = \mathcal{H}(\hat{x}_{n+1}^{j}) \qquad \hat{y}_{n+1} = \frac{1}{J}\sum_{j=1}^{J}\hat{y}_{n+1}^{j}$$

$$\hat{X}_{n+1} = \frac{1}{\sqrt{J-1}}\begin{pmatrix}\hat{x}_{n+1}^{1}-\hat{m}_{n+1} & \hat{x}_{n+1}^{2}-\hat{m}_{n+1} & \dots & \hat{x}_{n+1}^{J}-\hat{m}_{n+1}\end{pmatrix}$$

$$X_{n+1} = \frac{1}{\sqrt{J-1}}\begin{pmatrix}x_{n+1}^{1}-m_{n+1} & x_{n+1}^{2}-m_{n+1} & \dots & x_{n+1}^{J}-m_{n+1}\end{pmatrix}$$

$$\hat{Y}_{n+1} = \frac{1}{\sqrt{J-1}}\begin{pmatrix}\hat{y}_{n+1}^{1}-\hat{y}_{n+1} & \hat{y}_{n+1}^{2}-\hat{y}_{n+1} & \dots & \hat{y}_{n+1}^{J}-\hat{y}_{n+1}\end{pmatrix}$$

和

$$\hat{C}_{n+1}^{xy} = \hat{X}_{n+1}\hat{Y}_{n+1}^{T} \qquad \hat{C}_{n+1}^{yy} = \hat{Y}_{n+1}\hat{Y}_{n+1}^{T} + \Sigma_{\eta} \qquad \hat{C}_{n+1} = \hat{X}_{n+1}\hat{X}_{n+1}^{T} \qquad C_{n+1} = X_{n+1}X_{n+1}^{T}$$

对于期望，我们有

$$\begin{aligned} m_{n+1} &= \hat{m}_{n+1} + \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}(y_{n+1}-\hat{y}_{n+1}) - \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}\frac{\sum_{j=1}^{J}\eta_{n+1}^{j}}{J} \\ &= \hat{m}_{n+1} + \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}(y_{n+1}-\hat{y}_{n+1}) + \mathcal{O}(\frac{1}{\sqrt{J}}) \\ x_{n+1}^{j} - m_{n+1} &= \hat{x}_{n+1}^{j} - \hat{m}_{n+1} + \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}\Big(\frac{\sum_{j=1}^{J}\eta_{n+1}^{j}}{J} - \eta_{n+1}^{j} + \hat{y}_{n+1} - \hat{y}_{n+1}^{j}\Big) \end{aligned}$$

对于协方差，我们定义

$$E_{n+1} = \frac{1}{\sqrt{J-1}}\begin{pmatrix}\eta_{n+1}^{1}-\frac{\sum_{j=1}^{J}\eta_{n+1}^{j}}{J} & \eta_{n+1}^{2}-\frac{\sum_{j=1}^{J}\eta_{n+1}^{j}}{J} & \dots & \eta_{n+1}^{J}-\frac{\sum_{j=1}^{J}\eta_{n+1}^{j}}{J}\end{pmatrix}$$

我们有

$$
\begin{aligned}
C_{n+1} =& Z_{n+1}Z_{n+1}^T \\
=& \hat{C}_{n+1} - \hat{Z}_{n+1}(E_{n+1}+\hat{Y}_{n+1})^T(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} - \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}(E_{n+1}+\hat{Y}_{n+1})\hat{Z}_{n+1}^T \\
&+ \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}(E_{n+1}+\hat{Y}_{n+1})(E_{n+1}+\hat{Y}_{n+1})^T(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} \\
=& \hat{C}_{n+1} - 2\hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} \\
&- \hat{Z}_{n+1}E_{n+1}^T(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} - \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}E_{n+1}\hat{Z}_{n+1}^T \\
&+ \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}\Big(\hat{Y}_{n+1}\hat{Y}_{n+1}^T + E_{n+1}E_{n+1}^T + E_{n+1}\hat{Y}_{n+1}^T + \hat{Y}_{n+1}E_{n+1}^T\Big)(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} \\
=& \hat{C}_{n+1} - \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} \\
&- \hat{Z}_{n+1}E_{n+1}^T(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} - \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}E_{n+1}\hat{X}_{n+1}^T \\
&+ \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}\Big(E_{n+1}E_{n+1}^T - \Sigma_\eta + E_{n+1}\hat{Y}_{n+1}^T + \hat{Y}_{n+1}E_{n+1}^T\Big)(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} \\
=& \hat{C}_{n+1} - \hat{C}_{n+1}^{xy}(\hat{C}_{n+1}^{yy})^{-1}\hat{C}_{n+1}^{xy^T} + \mathcal{O}(\frac{1}{\sqrt{J}})
\end{aligned}
$$

这里我们使用 $\mathbb{E}E_{n+1} = 0$, E_{n+1}每一列方差是 $\mathcal{O}(\frac{1}{J})$。

2. Lorenz 63

考虑洛伦兹63系统

$$
\begin{aligned}
\frac{dx_1}{dt} &= \sigma(x_2 - x_1), \\
\frac{dx_2}{dt} &= x_1(r - x_3) - x_2, \\
\frac{dx_3}{dt} &= x_1x_2 - \beta x_3;
\end{aligned}
$$

其中 $\sigma = 10, \beta = \frac{8}{3}, r = 28$。

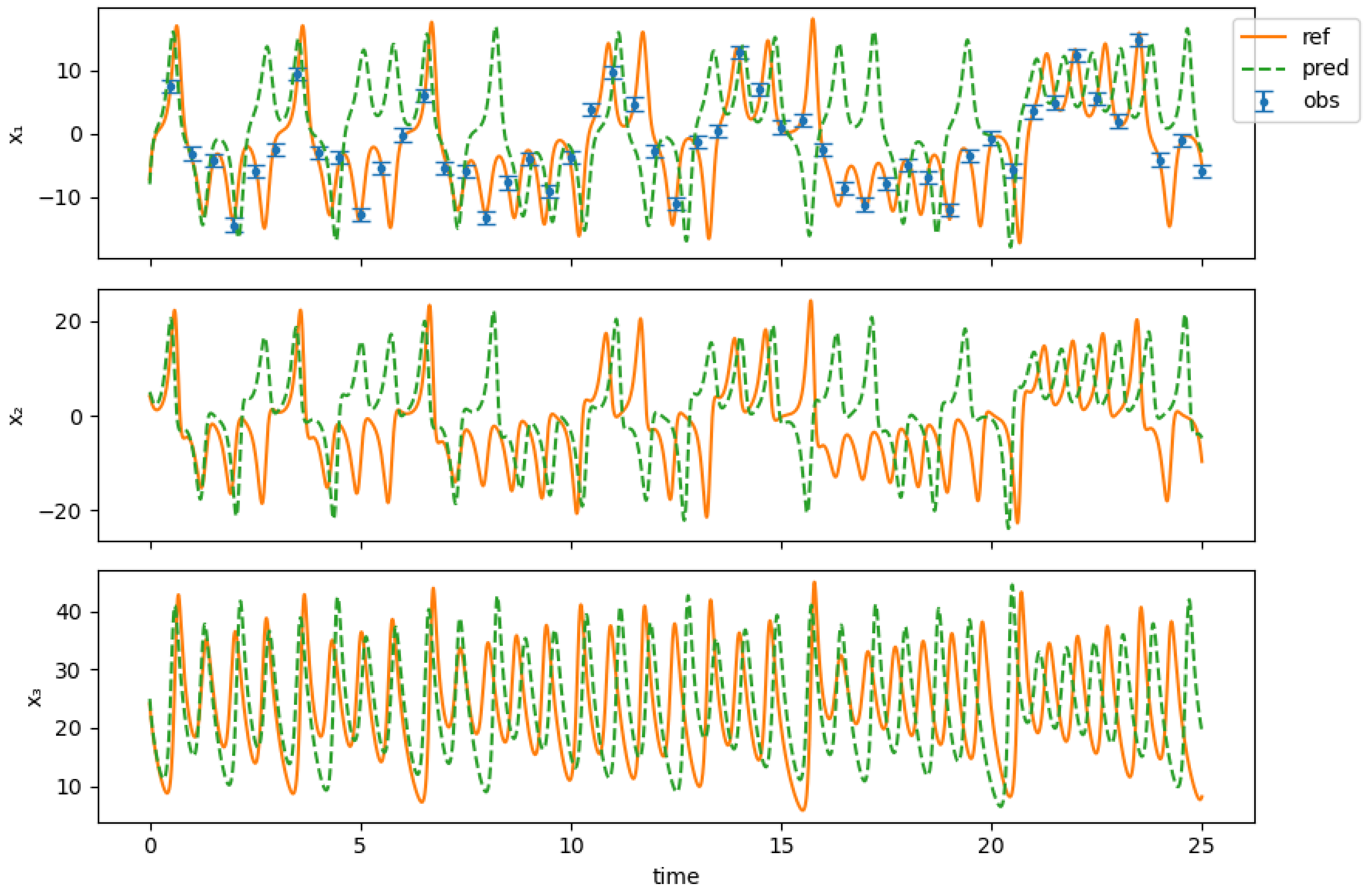
生成参考数据

In [14]:
```
using PyPlot
using Statistics
using LinearAlgebra
using Distributions, Random

function f(x::Array{FT,1}, σ::FT, r::FT, β::FT) where {FT<:AbstractFloat}
    return [σ*(x[2]-x[1]); x[1]*(r-x[3])-x[2]; x[1]*x[2]-β*x[3]]
end

function obs_f(x::Array{FT,1}) where {FT<:AbstractFloat}
   return [x[1];]
end

# integrate Lorenz63 with forward Euler method
function compute_Lorenz63_FE(x0::Array{FT,1}, θ::Array{FT,1}, Δt::FT, N_t::IT) where {FT<:AbstractFloat, IT<:Int}
    N_x = length(x0)

    σ, r, β = θ

    xs = zeros(N_x, N_t+1)
    xs[:, 1] = x0
    for i = 1:N_t
        xs[:, i+1] = xs[:, i] + Δt*f(xs[:, i], σ, r, β)
    end

    return xs
end
```

Out[14]:
```
compute_Lorenz63_FE (generic function with 1 method)
```

In [15]:
```
Random.seed!(42)

T = 25
Δt = 0.001
N_t = Int64(T/Δt)
time = LinRange(0, T, N_t+1)

dt_obs = 0.5
N_obs = Int64( T / dt_obs )
```

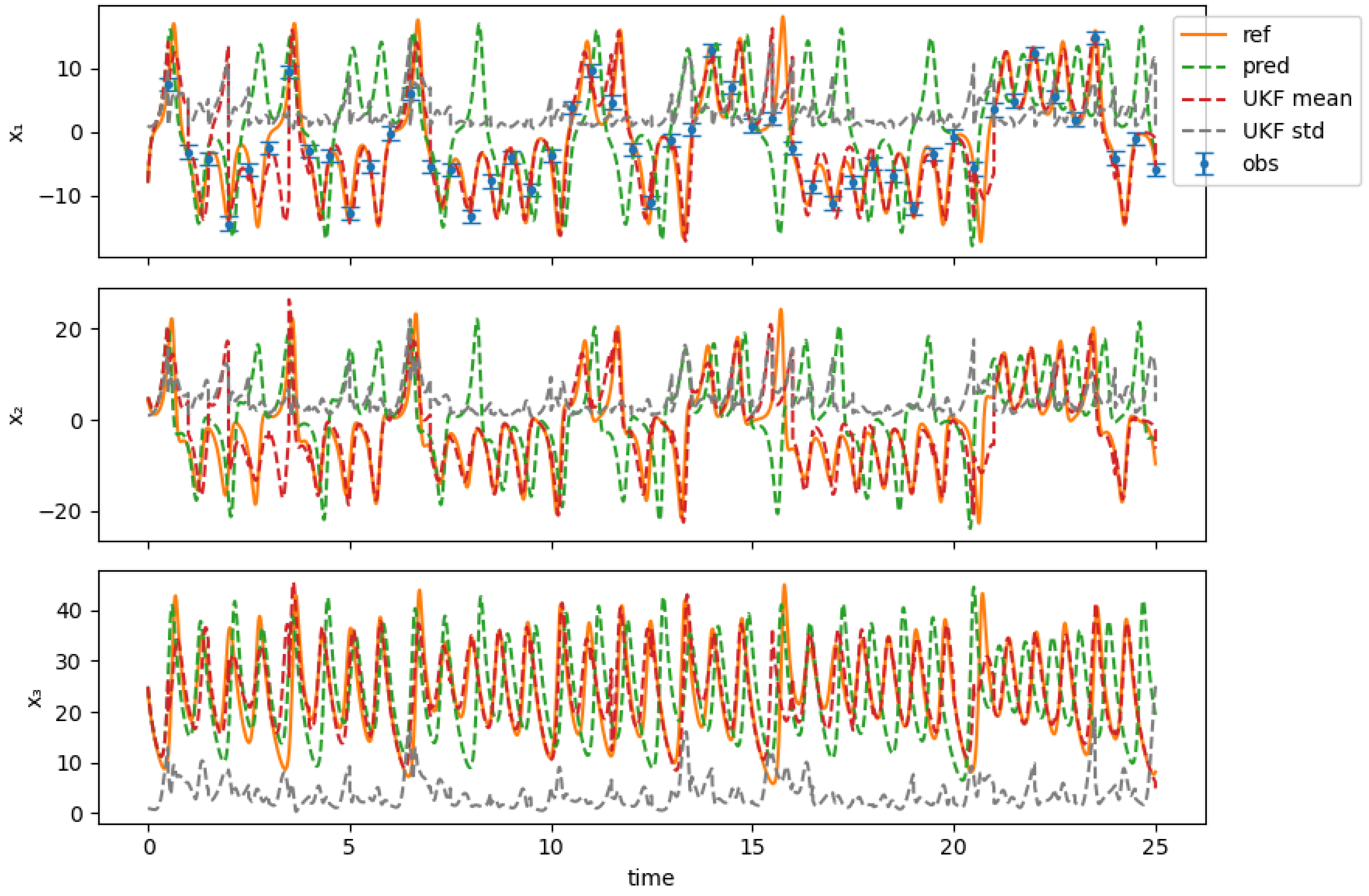
```julia
gap_obs  = Int64(dt_obs/Δt)
ind_obs = Array(gap_obs+1:gap_obs:length(time))
time_obs = time[ind_obs]
sigobs = 1.  # standard deviation of the observation noise
x0_mean = [-8.0; 5.0; 25]
x0_cov = sigobs^2 * Array(diagm(ones(length(x0_mean))))
obs0_mean = obs_f(x0_mean)
obs_err_cov = sigobs^2 * Array(diagm(ones(length(obs0_mean))))

# reference trajectory and observation
x0_ref = x0_mean + sigobs .* rand(Normal(0, 1), size(x0_mean))
θ = [10.0; 28.0; 8.0/3.0]
xs_ref = compute_Lorenz63_FE(x0_ref, θ, Δt, N_t)
obs_ref_noiseless = zeros(length(obs0_mean), length(ind_obs))
for i = 1:length(ind_obs)
   obs_ref_noiseless[:, i] = obs_f(xs_ref[:,ind_obs[i]])
end
noise = sigobs .* rand(Normal(0, 1), size(obs_ref_noiseless))
obs_ref = obs_ref_noiseless + noise

# prediction initialized at the mean
x0_pred = x0_mean
xs_pred = compute_Lorenz63_FE(x0_pred, θ, Δt, N_t)

# plot
fig, ax = PyPlot.subplots(ncols=1, nrows=3, sharex=true, sharey=false, figsize=(9,6))
ax[1].plot(time, xs_ref[1,:], color="C1",  label="ref")
ax[1].errorbar(time_obs, obs_ref[1,:], yerr=sigobs.+zeros(length(time_obs)), fmt="o", markersize=3, capsize=4, label='
ax[1].plot(time, xs_pred[1,:], color="C2", "--",label="pred")

ax[1].set_ylabel("x₁")
ax[1].legend(bbox_to_anchor=(1.1, 1), loc="upper right")
ax[2].plot(time, xs_ref[2,:], color="C1")
ax[2].plot(time, xs_pred[2,:], "--", color="C2")

ax[2].set_ylabel("x₂")
ax[3].plot(time, xs_ref[3,:], color="C1")
ax[3].plot(time, xs_pred[3,:], "--", color="C2")
ax[3].set_ylabel("x₃")
ax[3].set_xlabel("time")
fig.tight_layout()
fig.savefig("Lorenz63-DA-traj.pdf")
```



使用无迹卡尔曼滤波

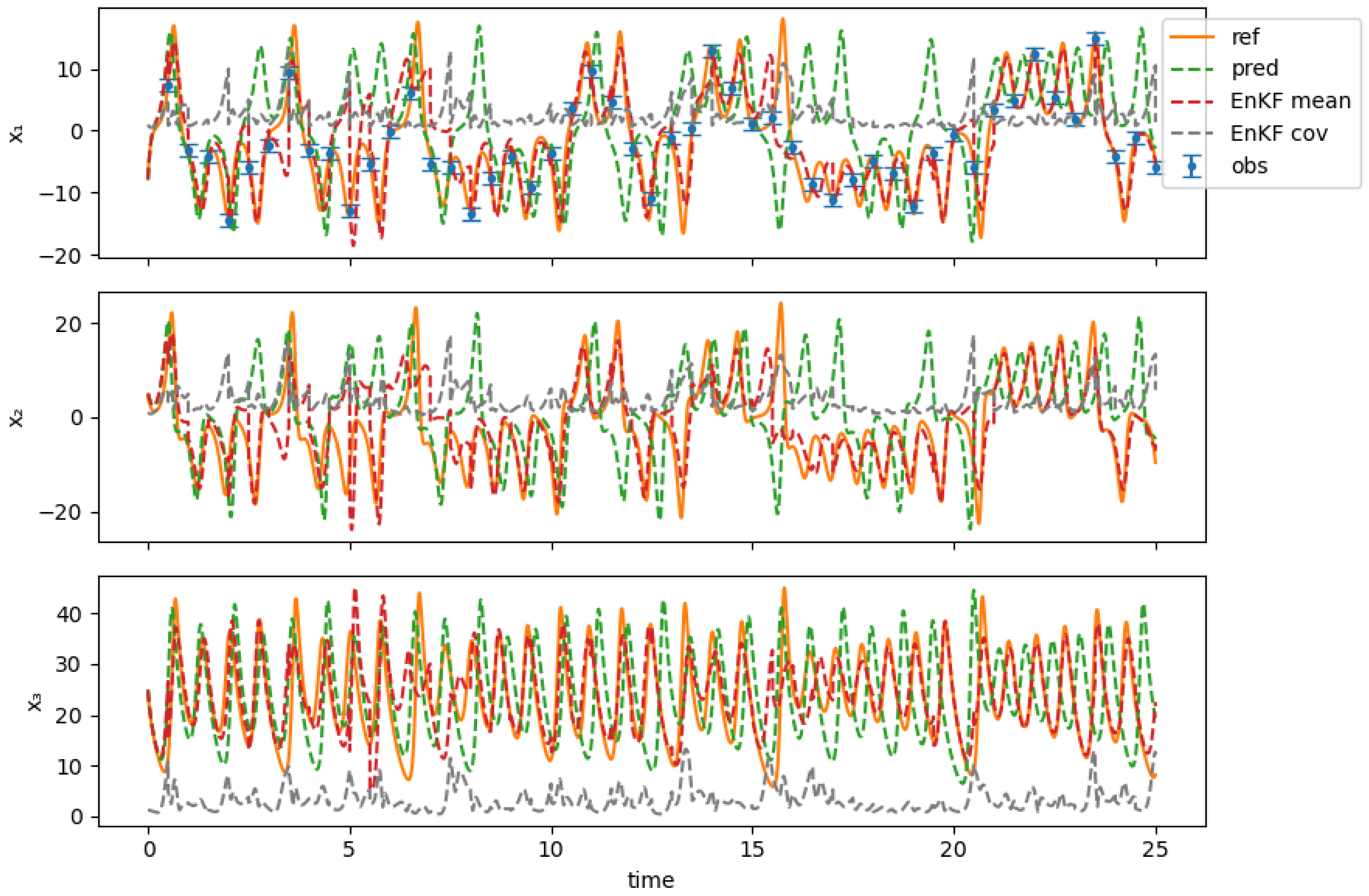
In [16]:
```julia
function generate_sigma_points(m::Array{Float64,1}, C::Array{Float64,2}, sigma_weight::Array{Float64,1})
    # generate sigma points
    dim = length(m)
    N_ens = 2dim + 1
    chol_C = cholesky(Hermitian(C)).L
    θ = zeros(Float64, dim, N_ens)
    θ[:, 1] = m
    for i = 1: dim
        θ[:, i+1] = m + sigma_weight[i]*chol_C[:,i]
        θ[:, i+1+dim] = m - sigma_weight[i]*chol_C[:,i]
    end

    return θ
end


function unscented_transform(g::Array{Float64,2}, mean_weight::Array{Float64,1}, cov_weight::Array{Float64,1})
    # compute mean
    dim, N_ens = size(g)
    mg = g * mean_weight
    Cg = zeros(Float64, dim, dim)
    for i = 1: N_ens
        Cg .+= cov_weight[i]*(g[:, i] - mg)*(g[:, i] - mg)'
    end
    return mg, Cg
end


function unscented_transform_cov(g₁::Array{Float64,2}, g₂::Array{Float64,2}, mean_weight::Array{Float64,1}, cov_weight
    # compute mean
    dim₁, N_ens = size(g₁)
    dim₂, N_ens = size(g₂)
    mg₁ = g₁ * mean_weight
    mg₂ = g₂ * mean_weight
    Cg₁g₂ = zeros(Float64, dim₁, dim₂)
    for i = 1: N_ens
        Cg₁g₂ += cov_weight[i]*(g₁[:, i] - mg₁)*(g₂[:, i] - mg₂)'
    end
    return Cg₁g₂
end

function UKF_init(dim)
    N_ens = 2dim + 1


    a = min(sqrt(4/(dim)), 1.0)
    λ = a^2*(dim) - dim
    # weights
    sigma_weight = zeros(dim)
    sigma_weight .= sqrt(dim + λ)

    mean_weight = zeros(N_ens)
    mean_weight[1] = 1.0

    cov_weight = zeros(N_ens)
    cov_weight .= 1/(2(dim + λ))

    return sigma_weight, mean_weight, cov_weight
end



# integrate Lorenz63 with forward Euler method
function compute_Lorenz63_FE_UKF(x0::Array{FT,1}, θ::Array{FT,1}, Δt::FT, N_t::IT,
        cov0::Array{FT,2}, obs::Array{FT,2}, sigobs::FT, gap_obs::IT
        ) where {FT<:AbstractFloat, IT<:Int}
    N_x = length(x0)

    σ, r, β = θ
    N_ens = 2*N_x + 1

    sigma_weight, mean_weight, cov_weight = UKF_init(N_x)

    xs = zeros(N_x, N_t+1, N_ens)
    xs_mean = zeros(N_x, N_t+1)
    xs_cov = zeros(N_x, N_x, N_t+1)

    xs[:, 1, :] = generate_sigma_points(x0, cov0, sigma_weight)
    xs_mean[:, 1] .= x0
    xs_cov[:, :, 1] .= cov0
    Hxs = zeros(size(obs,1), N_ens)

    for n = 1:N_t
        # prediction
        for j = 1:N_ens
            xs[:, n+1, j] = xs[:, n, j] + Δt*f(xs[:, n, j], σ, r, β)
        end
        xs_mean[:, n+1], xs_cov[:, :, n+1] = unscented_transform(xs[:, n+1, :], mean_weight, cov_weight)

        # analysis
        if (n+1 - 1) % gap_obs == 0
            # compute observation
            for j = 1:N_ens
                Hxs[:, j] = obs_f(xs[:, n+1, j])
            end

            ŷ, Ĉʸʸ = unscented_transform(Hxs, mean_weight, cov_weight)
            Ĉʸʸ += sigobs^2 * I
            Ĉˣʸ = unscented_transform_cov(xs[:, n+1, :], Hxs, mean_weight, cov_weight)
            xs_mean[:, n+1] = xs_mean[:, n+1] + Ĉˣʸ*(Ĉʸʸ\(obs[:, div(n,gap_obs)] - ŷ))
            xs_cov[:, :, n+1] = xs_cov[:, :, n+1] - Ĉˣʸ*(Ĉʸʸ\(Ĉˣʸ'))
            xs[:, n+1, :] = generate_sigma_points(xs_mean[:, n+1], xs_cov[:, :, n+1], sigma_weight)
        end
    end

    return xs, xs_mean, xs_cov
end



xs_ukf, xs_mean_ukf, xs_cov_ukf = compute_Lorenz63_FE_UKF(x0_mean, θ, Δt, N_t, x0_cov, obs_ref, sigobs, gap_obs)


fig, ax = PyPlot.subplots(ncols=1, nrows=3, sharex=true, sharey=false, figsize=(9,6))
ax[1].plot(time, xs_ref[1,:], color="C1", label="ref")
ax[1].errorbar(time_obs, obs_ref[1,:], yerr=sigobs.+zeros(length(time_obs)), fmt="o", markersize=3, capsize=4, label="
ax[1].plot(time, xs_pred[1,:], "--", color="C2", label="pred")
ax[1].plot(time, xs_mean_ukf[1,:], "--", color="C3", label="UKF mean")
ax[1].plot(time, sqrt.(xs_cov_ukf[1,1,:]), "--", color="grey", label="UKF std")
ax[1].set_ylabel("x₁")
ax[1].legend(bbox_to_anchor=(1.15, 1), loc="upper right")

ax[2].plot(time, xs_ref[2,:], color="C1")
ax[2].plot(time, xs_pred[2,:], "--", color="C2", label="pred")
ax[2].plot(time, xs_mean_ukf[2,:], "--", color="C3", label="UKF mean")
ax[2].plot(time, sqrt.(xs_cov_ukf[2,2,:]), "--", color="grey", label="UKF std")
ax[2].set_ylabel("x₂")

ax[3].plot(time, xs_ref[3,:], color="C1")
ax[3].plot(time, xs_pred[3,:], "--", color="C2", label="pred")
ax[3].plot(time, xs_mean_ukf[3,:], "--", color="C3", label="UKF")
ax[3].plot(time, sqrt.(xs_cov_ukf[3,3,:]), "--", color="grey", label="UKF std")
ax[3].set_ylabel("x₃")
ax[3].set_xlabel("time")
```

```
fig.tight_layout()
fig.savefig("Lorenz63-DA-traj-UKF.pdf")
```



使用集合卡尔曼滤波

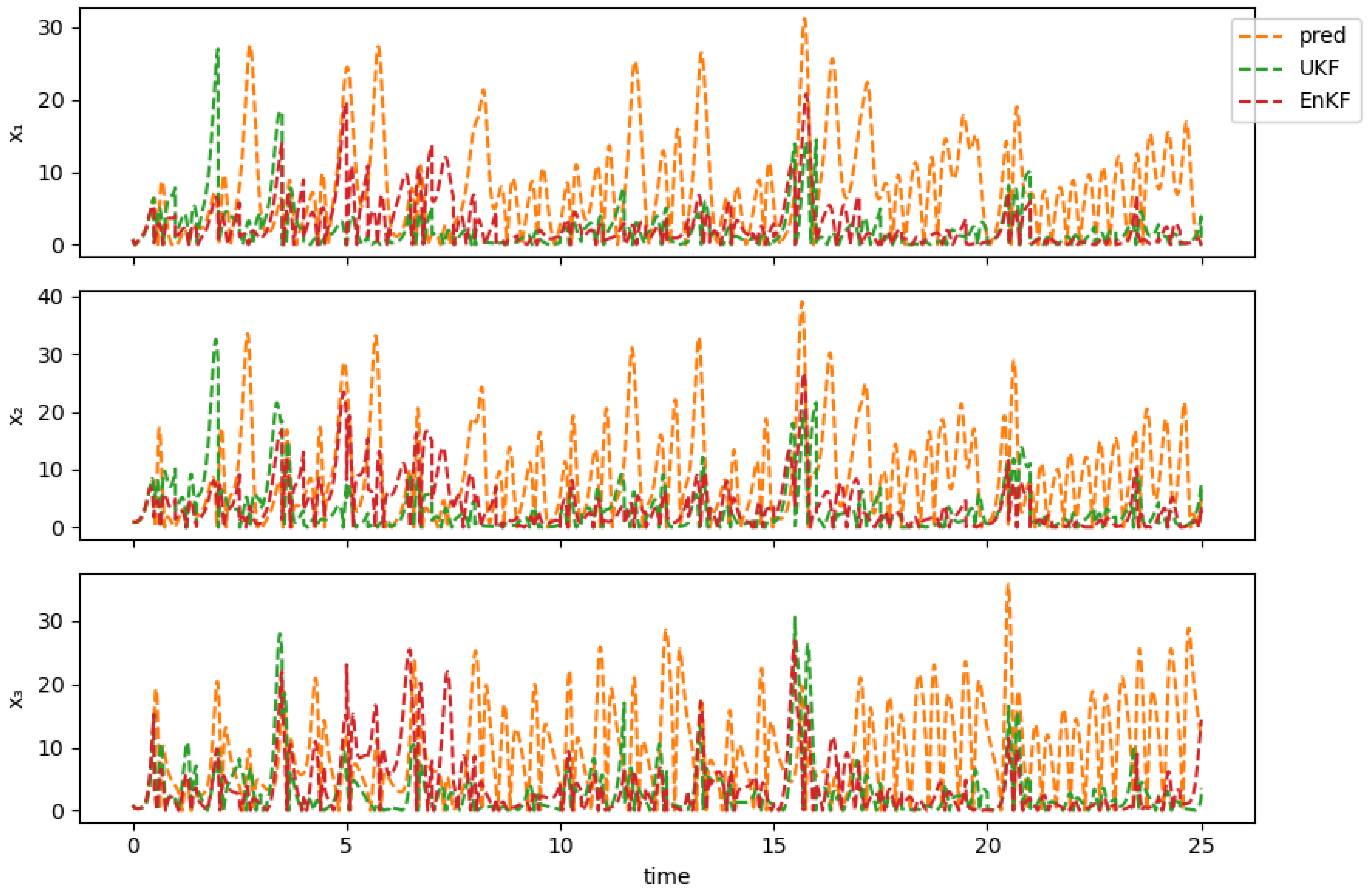
In [17]:
```julia
# integrate Lorenz63 with forward Euler method
function compute_Lorenz63_FE_EnKF(x0::Array{FT,1}, θ::Array{FT,1}, Δt::FT, N_t::IT,
        cov0::Array{FT,2}, obs::Array{FT,2}, sigobs::FT, gap_obs::IT,  N_ens::IT
        ) where {FT<:AbstractFloat, IT<:Int}
    N_x = length(x0)

    σ, r, β = θ

    xs = zeros(N_x, N_t+1, N_ens)

    xs[:, 1, :] = rand(MvNormal(x0, cov0), N_ens)

    Hxs = zeros(size(obs,1), N_ens)

    for n = 1:N_t
        # prediction
        for j = 1:N_ens
            xs[:, n+1, j] = xs[:, n, j] + Δt*f(xs[:, n, j], σ, r, β)
        end

        # analysis
        if (n+1 - 1) % gap_obs == 0
            # compute observation
            for j = 1:N_ens
                Hxs[:, j] = obs_f(xs[:, n+1, j])
            end
            x̂ = mean(xs[:,n+1,:], dims=2)
            ŷ = mean(Hxs, dims=2)

            Ĉʸʸ = ((Hxs .- ŷ) * (Hxs .- ŷ)')/(N_ens - 1) + sigobs^2 * I
            Ĉˣʸ = ((xs[:, n+1, :] .- x̂) * (Hxs .- ŷ)')/(N_ens - 1)
            η = sigobs .* rand(Normal(0, 1), size(Hxs))
            for j = 1:N_ens
                xs[:, n+1, j] += Ĉˣʸ*(Ĉʸʸ\(obs[:, div(n,gap_obs)] - Hxs[:, j] - η[:,j]))
            end
        end
    end

    xs_mean = zeros(N_x, N_t+1)
    xs_cov = zeros(N_x, N_x, N_t+1)
    for n = 1:N_t+1
        xs_mean[:,n] = mean(xs[:,n,:], dims=2)
        xs_cov[:,:,n] = ((xs[:,n,:] .- xs_mean[:,n]) * (xs[:,n,:] .- xs_mean[:,n])')/(N_ens - 1)
    end

    return xs, xs_mean, xs_cov
end


N_ens = 10
xs_enkf, xs_mean_enkf, xs_cov_enkf = compute_Lorenz63_FE_EnKF(x0_mean, θ, Δt, N_t, x0_cov, obs_ref, sigobs, gap_obs, N


fig, ax = PyPlot.subplots(ncols=1, nrows=3, sharex=true, sharey=false, figsize=(9,6))
ax[1].plot(time, xs_ref[1,:], color="C1", label="ref")
ax[1].errorbar(time_obs, obs_ref[1,:], yerr=sigobs.+zeros(length(time_obs)), fmt="o", markersize=3, capsize=4, label='
ax[1].plot(time, xs_pred[1,:], "--", color="C2", label="pred")
ax[1].plot(time, xs_mean_enkf[1,:], "--", color="C3", label="EnKF mean")
ax[1].plot(time, sqrt.(xs_cov_enkf[1,1,:]), "--", color="grey", label="EnKF cov")
ax[1].set_ylabel("x₁")
ax[1].legend(bbox_to_anchor=(1.15, 1), loc="upper right")


ax[2].plot(time, xs_ref[2,:], color="C1")
ax[2].plot(time, xs_pred[2,:], "--", color="C2", label="pred")
ax[2].plot(time, xs_mean_enkf[2,:], "--", color="C3", label="EnKF")
ax[2].plot(time, sqrt.(xs_cov_enkf[2,2,:]), "--", color="grey", label="EnKF cov")
ax[2].set_ylabel("x₂")


ax[3].plot(time, xs_ref[3,:], color="C1")
ax[3].plot(time, xs_pred[3,:], "--", color="C2", label="pred")
ax[3].plot(time, xs_mean_enkf[3,:], "--", color="C3", label="EnKF")
ax[3].plot(time, sqrt.(xs_cov_enkf[3,3,:]), "--", color="grey", label="EnKF cov")
ax[3].set_ylabel("x₃")
ax[3].set_xlabel("time")
fig.tight_layout()
fig.savefig("Lorenz63-DA-traj-EnKF.pdf")
```



```
In [18]: fig, ax = PyPlot.subplots(ncols=1, nrows=3, sharex=true, sharey=false, figsize=(9,6))
         #ax[1].plot(time, xs_ref[1,:], color="C1", label="ref")
         ax[1].plot(time, abs.(xs_pred[1,:] - xs_ref[1,:]), "--", color="C1", label="pred")
         ax[1].plot(time, abs.(xs_mean_ukf[1,:] - xs_ref[1,:]), "--", color="C2", label="UKF")
         ax[1].plot(time, abs.(xs_mean_enkf[1,:] - xs_ref[1,:]), "--", color="C3", label="EnKF")
         ax[1].set_ylabel("x₁")
```

```julia
ax[1].legend(bbox_to_anchor=(1.1, 1), loc="upper right")
@info "Pred : ", mean(abs.(xs_pred[1,:] - xs_ref[1,:]))
@info "UKF : ", mean(abs.(xs_mean_ukf[1,:] - xs_ref[1,:]))
@info "EKF : ", mean(abs.(xs_mean_enkf[1,:] - xs_ref[1,:]))


ax[2].plot(time, abs.(xs_pred[2,:] - xs_ref[2,:]), "--", color="C1", label="pred")
ax[2].plot(time, abs.(xs_mean_ukf[2,:] - xs_ref[2,:]), "--", color="C2", label="UKF")
ax[2].plot(time, abs.(xs_mean_enkf[2,:] - xs_ref[2,:]), "--", color="C3", label="EnKF")
ax[2].set_ylabel("x₂")
@info "Pred : ", mean(abs.(xs_pred[2,:] - xs_ref[2,:]))
@info "UKF : ", mean(abs.(xs_mean_ukf[2,:] - xs_ref[2,:]))
@info "EKF : ", mean(abs.(xs_mean_enkf[2,:] - xs_ref[2,:]))


ax[3].plot(time, abs.(xs_pred[3,:] - xs_ref[3,:]), "--", color="C1", label="pred")
ax[3].plot(time, abs.(xs_mean_ukf[3,:] - xs_ref[3,:]), "--", color="C2", label="UKF")
ax[3].plot(time, abs.(xs_mean_enkf[3,:] - xs_ref[3,:]), "--", color="C3", label="EnKF")
ax[3].set_ylabel("x₃")
ax[3].set_xlabel("time")
@info "Pred : ", mean(abs.(xs_pred[3,:] - xs_ref[3,:]))
@info "UKF : ", mean(abs.(xs_mean_ukf[3,:] - xs_ref[3,:]))
@info "EKF : ", mean(abs.(xs_mean_enkf[3,:] - xs_ref[3,:]))


fig.tight_layout()
fig.savefig("Lorenz63-DA-traj-Error.pdf")
```

```
[ Info: ("Pred : ", 7.83536433270032‌35)
[ Info: ("UKF : ", 2.212368756713286)
[ Info: ("EKF : ", 2.7166280742607536)
[ Info: ("Pred : ", 9.178939088062657)
[ Info: ("UKF : ", 3.311514251367836)
[ Info: ("EKF : ", 3.9308310093805057)
[ Info: ("Pred : ", 9.688611194395497)
[ Info: ("UKF : ", 3.1049010436331006)
[ Info: ("EKF : ", 3.830685065795678)
```



考虑对流方程

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0 \qquad x \in [0, 1000].$$

In [122…

```
function generate_u0(N_mode=50 ;s=true, c=true)
    L, N = 1000, 1000
    xx = LinRange(0,L, N+1)[1:N]

    u0 = zeros(N)

    a, b = rand(Normal(0, 1), N_mode), rand(Normal(0, 1), N_mode)

    if s
        for i = 1:N_mode
            u0 += a[i]*sin.(2π*xx*i/L)
        end
    end
    if c
        for i = 1:N_mode
            u0 += b[i]*cos.(2π*xx*i/L)
        end
    end

    return u0
end


function advection(u0, t)
    #return ut
    N = length(u0)
    u = zeros(N)
    u[t+1:N] = u0[1:N-t]
    u[1:t] = u0[N-t+1:N]
    return u
end
function get_obs(u)
    return u[[125;375;625;875]]
end

function trunc_svd(X,  ϵ = 1.0e-6)
    n_row, n_col = size(X)
    svd_X = svd(X)
    rank_X = min(n_row, n_col)
    for i = 1:min(n_row, n_col)
        if svd_X.S[i] <= ϵ*svd_X.S[1]
            rank_X = i - 1
            break
        end
    end

    return svd_X.U[:, 1:rank_X], svd_X.S[1:rank_X], svd_X.Vt[1:rank_X, :]'
end
```

Out[122]:
```
trunc_svd (generic function with 2 methods)
```

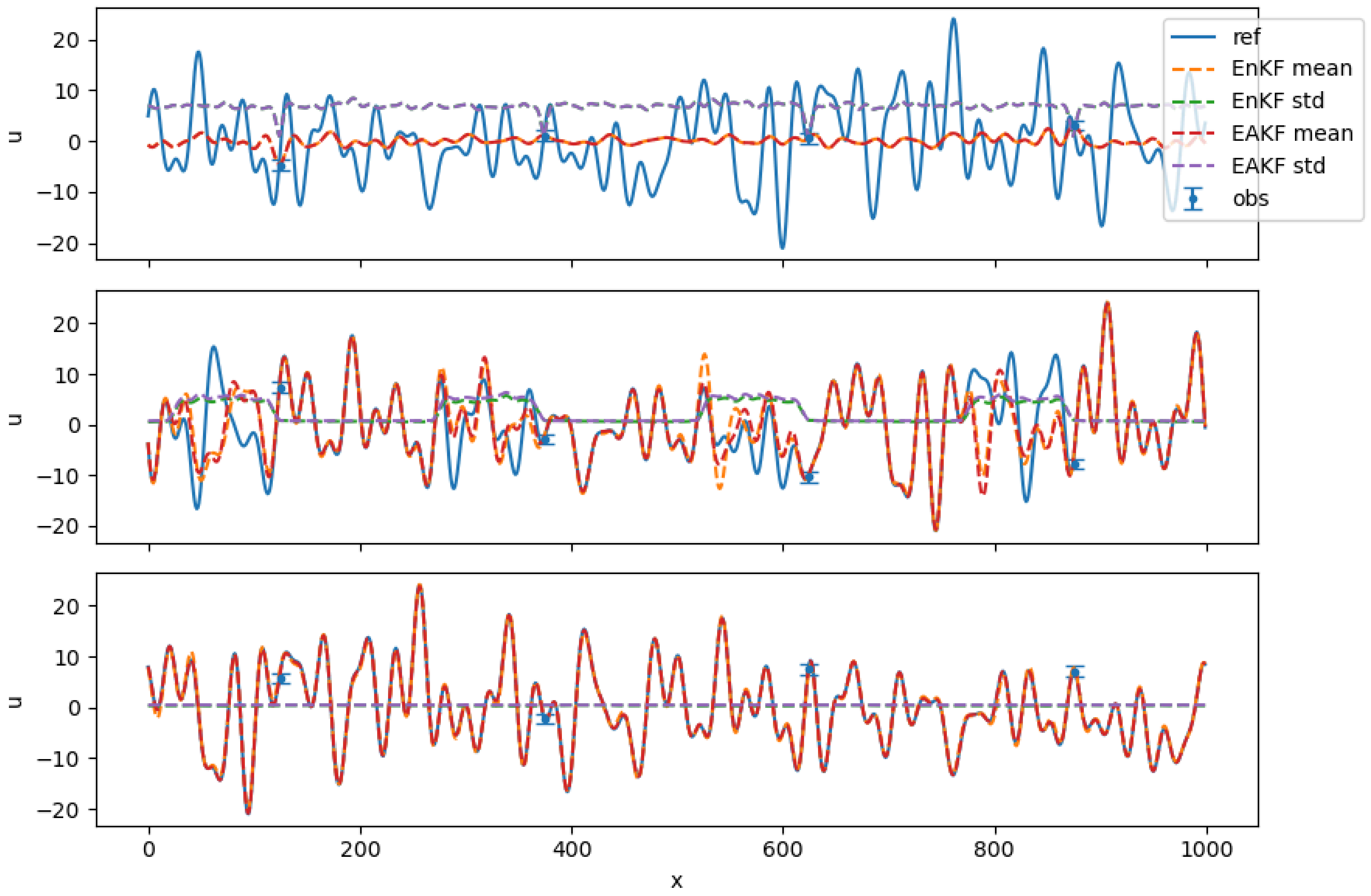
In [123…
```
# integrate Lorenz63 with forward Euler method
function compute_advection_EKF(u0_ref::Array{FT,1}, u0s::Array{FT,2}, obs_t::IT, N_t::IT, sigobs::FT, N_ens::IT; metho
        ) where {FT<:AbstractFloat, IT<:Int}

    N = length(u0_ref)
    us = zeros(N, N_t+1, N_ens)
    us_ref = zeros(N, N_t+1)
    us[:, 1, :] = u0s
    us_ref[:, 1] = u0_ref
    Hus = zeros(4, N_ens)

    for n = 1:N_t
        # prediction
        for j = 1:N_ens
            us[:, n+1, j] = advection(us[:, n, j], obs_t)
        end
        us_ref[:, n+1] = advection(us_ref[:, n], obs_t)

        # analysis

        # compute observation
        for j = 1:N_ens
            Hus[:, j] = get_obs(us[:, n+1, j])
        end
        obs = get_obs(us_ref[:, n+1]) + 0.1*rand(Normal(0, 1), 4)



        x̂ = mean(us[:,n+1,:], dims=2)
        ŷ = mean(Hus, dims=2)
        if method == "EnKF"
            Ĉʸʸ = ((Hus .- ŷ) * (Hus .- ŷ)')/(N_ens - 1) + sigobs^2 * I
            Ĉˣʸ = ((us[:, n+1, :] .- x̂) * (Hus .- ŷ)')/(N_ens - 1)
            η = sigobs .* rand(Normal(0, 1), size(Hus))
            for j = 1:N_ens
                us[:, n+1, j] += Ĉˣʸ*(Ĉʸʸ\(obs - Hus[:, j] - η[:,j]))
            end
        elseif method == "EAKF"
            Ẑ = (us[:, n+1, :] - x̂*ones(N_ens)')/ sqrt(N_ens - 1)
            Ŷ = (Hus - ŷ*ones(N_ens)')/ sqrt(N_ens - 1)
            Ĉʸʸ = Ŷ * Ŷ' + sigobs^2 * I
            Ĉˣʸ = Ẑ * Ŷ'
            us_mean = x̂ + Ĉˣʸ*(Ĉʸʸ\(obs - ŷ))

            P, sqrt_D̂ , V =  trunc_svd(Ẑ)
            svd_Y = svd(V' * ((Ŷ'/sigobs^2*Ŷ + I)\V))

            U, D = svd_Y.U, svd_Y.S
            A = (P .* sqrt_D̂' * U .* sqrt.(D)') * (sqrt_D̂ .\ P')



            us[:, n+1, :] = (A*(us[:, n+1, :] - x̂*ones(N_ens)') + us_mean*ones(N_ens)')
        end

    end

    us_mean = zeros(N, N_t+1)
    us_cov = zeros(N, N, N_t+1)
    for n = 1:N_t+1
        us_mean[:,n] = mean(us[:,n,:], dims=2)
        us_cov[:,:,n] = ((us[:,n,:] .- us_mean[:,n]) * (us[:,n,:] .- us_mean[:,n])')/(N_ens - 1)
    end

    return us, us_mean, us_cov, us_ref
end
```

Out[123]:
```
compute_advection_EKF (generic function with 2 methods)
```

In [124…
```
Random.seed!(111)
u0_ref = generate_u0(50);
```

In [132…
```
Random.seed!(42)
L, N = 1000, 1000
xx = LinRange(0,L, N+1)[1:N]
obs_t, N_t, sigobs = 5, 100, 1.0
N_ens = 200
```

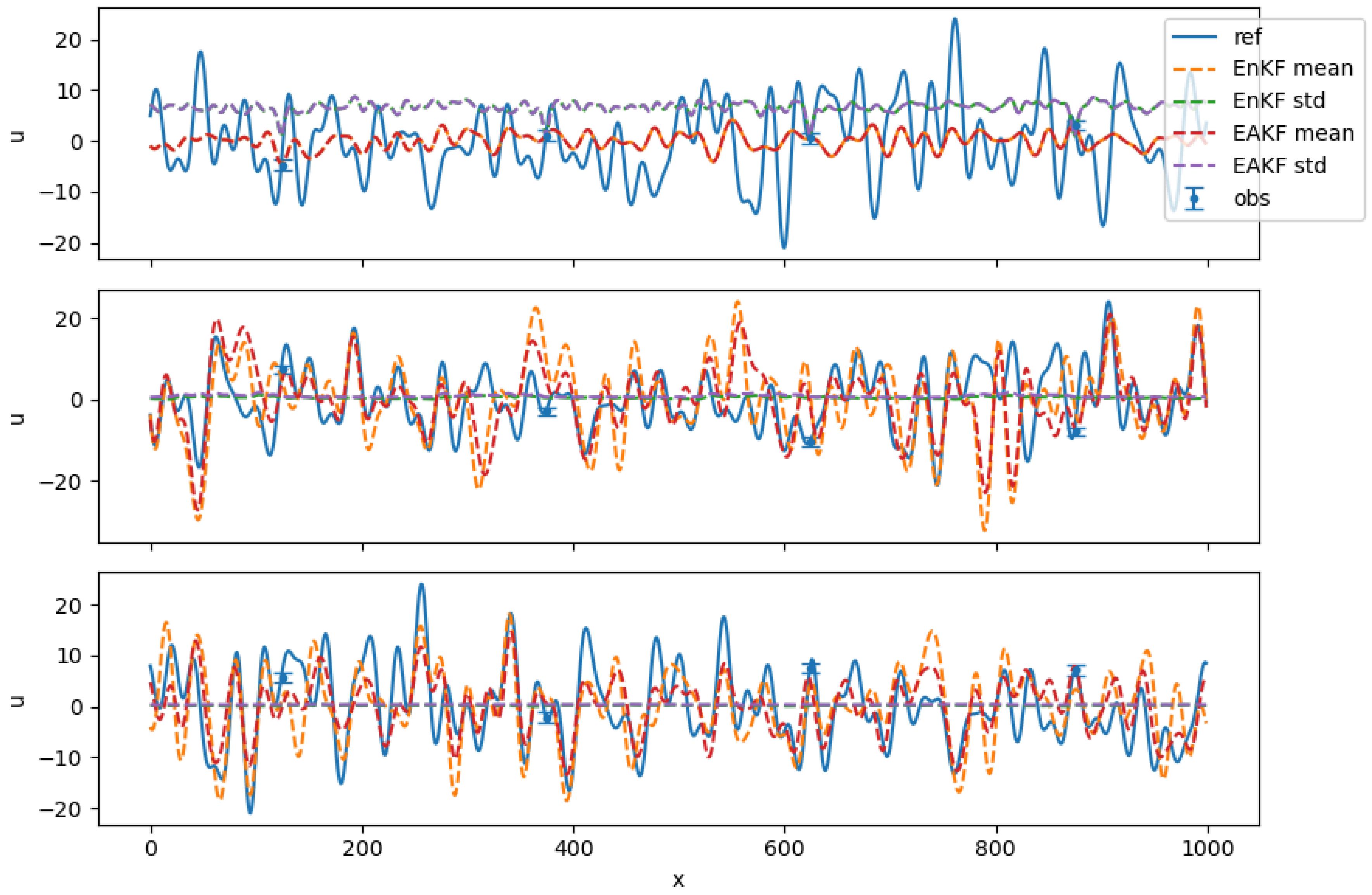
```
u0s = zeros(N, N_ens)
for j = 1:N_ens
    u0s[:, j] = generate_u0(50)
end
us_enkf, us_mean_enkf, us_cov_enkf, us_ref = compute_advection_EKF(u0_ref, u0s, obs_t, N_t, sigobs, N_ens; method="Enk
us_eakf, us_mean_eakf, us_cov_eakf, us_ref = compute_advection_EKF(u0_ref, u0s, obs_t, N_t, sigobs, N_ens; method="EAk

xx_obs = [125;375;625;875]

fig, ax = PyPlot.subplots(ncols=1, nrows=3, sharex=true, sharey=false, figsize=(9,6))

ax[1].plot(xx, us_ref[:,2], color="C0", label="ref")
ax[1].errorbar(xx_obs, get_obs(us_ref[:,2]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4, lab
ax[1].plot(xx, us_mean_enkf[:,2], "--", color="C1", label="EnKF mean")
ax[1].plot(xx, sqrt.(diag(us_cov_enkf[:,:,2])), "--", color="C2", label="EnKF std")
ax[1].plot(xx, us_mean_eakf[:,2], "--", color="C3", label="EAKF mean")
ax[1].plot(xx, sqrt.(diag(us_cov_eakf[:,:,2])), "--", color="C4", label="EAKF std")

ax[1].set_ylabel("u")
ax[1].legend(bbox_to_anchor=(1.1, 1), loc="upper right")

ax[2].plot(xx, us_ref[:, 31], color="C0")
ax[2].errorbar(xx_obs, get_obs(us_ref[:,31]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4, la
ax[2].plot(xx, us_mean_enkf[:, 31], "--", color="C1", label="EnKF")
ax[2].plot(xx, sqrt.(diag(us_cov_enkf[:,:,31])), "--", color="C2", label="EnKF std")
ax[2].plot(xx, us_mean_eakf[:,31], "--", color="C3", label="EAKF mean")
ax[2].plot(xx, sqrt.(diag(us_cov_eakf[:,:,31])), "--", color="C4", label="EAKF std")

ax[2].set_ylabel("u")

ax[3].plot(xx, us_ref[:, end], color="C0")
ax[3].errorbar(xx_obs, get_obs(us_ref[:,end]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4, 
ax[3].plot(xx, us_mean_enkf[:, end], "--", color="C1", label="EnKF")
ax[3].plot(xx, sqrt.(diag(us_cov_enkf[:,:,end])), "--", color="C2", label="EnKF std")
ax[3].plot(xx, us_mean_eakf[:, end], "--", color="C3", label="EAKF")
ax[3].plot(xx, sqrt.(diag(us_cov_eakf[:,:,end])), "--", color="C4", label="EAKF std")

ax[3].set_ylabel("u")
ax[3].set_xlabel("x")
fig.tight_layout()
fig.savefig("adv-DA-traj-1.pdf")
```



In [134…

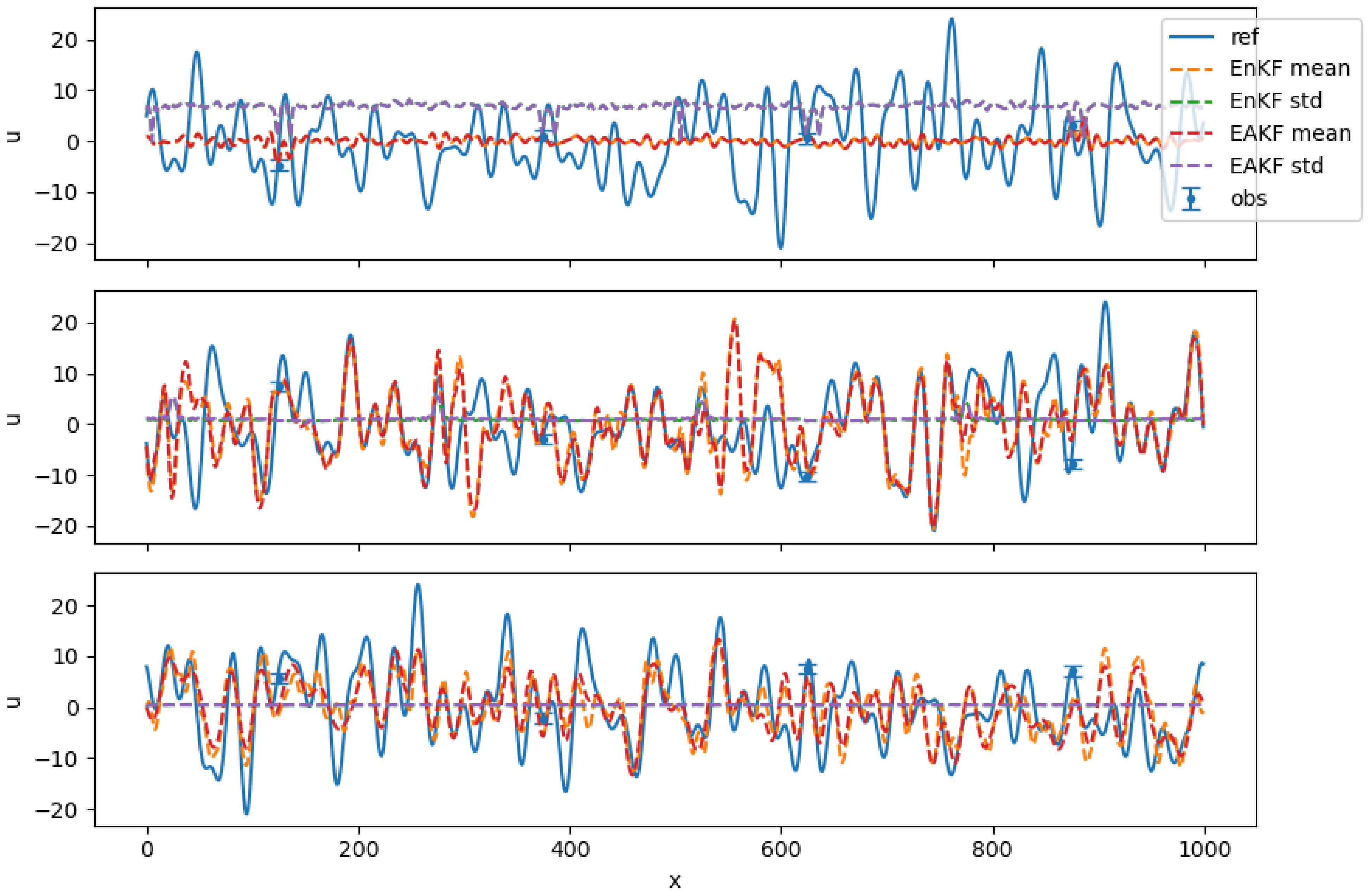
```
Random.seed!(42)
L, N = 1000, 1000
xx = LinRange(0,L, N+1)[1:N]
obs_t, N_t, sigobs = 5, 100, 1.0
N_ens = 50
u0s = zeros(N, N_ens)
```

```julia
for j = 1:N_ens
    u0s[:, j] = generate_u0(50)
end

us_enkf, us_mean_enkf, us_cov_enkf, us_ref = compute_advection_EKF(u0_ref, u0s, obs_t, N_t, sigobs, N_ens; method="En
us_eakf, us_mean_eakf, us_cov_eakf, us_ref = compute_advection_EKF(u0_ref, u0s, obs_t, N_t, sigobs, N_ens; method="EA

xx_obs = [125;375;625;875]

fig, ax = PyPlot.subplots(ncols=1, nrows=3, sharex=true, sharey=false, figsize=(9,6))

ax[1].plot(xx, us_ref[:,2], color="C0", label="ref")
ax[1].errorbar(xx_obs, get_obs(us_ref[:,2]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4, la
ax[1].plot(xx, us_mean_enkf[:,2], "--", color="C1", label="EnKF mean")
ax[1].plot(xx, sqrt.(diag(us_cov_enkf[:,:,2])), "--", color="C2", label="EnKF std")
ax[1].plot(xx, us_mean_eakf[:,2], "--", color="C3", label="EAKF mean")
ax[1].plot(xx, sqrt.(diag(us_cov_eakf[:,:,2])), "--", color="C4", label="EAKF std")

ax[1].set_ylabel("u")
ax[1].legend(bbox_to_anchor=(1.1, 1), loc="upper right")

ax[2].plot(xx, us_ref[:, 31], color="C0")
ax[2].errorbar(xx_obs, get_obs(us_ref[:,31]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4, l
ax[2].plot(xx, us_mean_enkf[:, 31], "--", color="C1", label="EnKF")
ax[2].plot(xx, sqrt.(diag(us_cov_enkf[:,:,31])), "--", color="C2", label="EnKF std")
ax[2].plot(xx, us_mean_eakf[:,31], "--", color="C3", label="EAKF mean")
ax[2].plot(xx, sqrt.(diag(us_cov_eakf[:,:,31])), "--", color="C4", label="EAKF std")

ax[2].set_ylabel("u")

ax[3].plot(xx, us_ref[:, end], color="C0")
ax[3].errorbar(xx_obs, get_obs(us_ref[:,end]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4,
ax[3].plot(xx, us_mean_enkf[:, end], "--", color="C1", label="EnKF")
ax[3].plot(xx, sqrt.(diag(us_cov_enkf[:,:,end])), "--", color="C2", label="EnKF std")
ax[3].plot(xx, us_mean_eakf[:, end], "--", color="C3", label="EAKF")
ax[3].plot(xx, sqrt.(diag(us_cov_eakf[:,:,end])), "--", color="C4", label="EAKF std")

ax[3].set_ylabel("u")
ax[3].set_xlabel("x")
fig.tight_layout()
fig.savefig("adv-DA-traj-2.pdf")
```



```
In [133…  Random.seed!(42)
          L, N = 1000, 1000
          xx = LinRange(0,L, N+1)[1:N]
          obs_t, N_t, sigobs = 5, 100, 1.0
          N_ens = 200
          u0s = zeros(N, N_ens)
```

```julia
for j = 1:N_ens
    u0s[:, j] = generate_u0(100; c=false)
end

us_enkf, us_mean_enkf, us_cov_enkf, us_ref = compute_advection_EKF(u0_ref, u0s, obs_t, N_t, sigobs, N_ens; method="Enk
us_eakf, us_mean_eakf, us_cov_eakf, us_ref = compute_advection_EKF(u0_ref, u0s, obs_t, N_t, sigobs, N_ens; method="EAK

xx_obs = [125;375;625;875]

fig, ax = PyPlot.subplots(ncols=1, nrows=3, sharex=true, sharey=false, figsize=(9,6))

ax[1].plot(xx, us_ref[:,2], color="C0", label="ref")
ax[1].errorbar(xx_obs, get_obs(us_ref[:,2]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4, lab
ax[1].plot(xx, us_mean_enkf[:,2], "--", color="C1", label="EnKF mean")
ax[1].plot(xx, sqrt.(diag(us_cov_enkf[:,:,2])), "--", color="C2", label="EnKF std")
ax[1].plot(xx, us_mean_eakf[:,2], "--", color="C3", label="EAKF mean")
ax[1].plot(xx, sqrt.(diag(us_cov_eakf[:,:,2])), "--", color="C4", label="EAKF std")

ax[1].set_ylabel("u")
ax[1].legend(bbox_to_anchor=(1.1, 1), loc="upper right")

ax[2].plot(xx, us_ref[:, 31], color="C0")
ax[2].errorbar(xx_obs, get_obs(us_ref[:,31]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4, la
ax[2].plot(xx, us_mean_enkf[:, 31], "--", color="C1", label="EnKF")
ax[2].plot(xx, sqrt.(diag(us_cov_enkf[:,:,31])), "--", color="C2", label="EnKF std")
ax[2].plot(xx, us_mean_eakf[:,31], "--", color="C3", label="EAKF mean")
ax[2].plot(xx, sqrt.(diag(us_cov_eakf[:,:,31])), "--", color="C4", label="EAKF std")

ax[2].set_ylabel("u")

ax[3].plot(xx, us_ref[:, end], color="C0")
ax[3].errorbar(xx_obs, get_obs(us_ref[:,end]), yerr=sigobs.+zeros(length(xx_obs)), fmt="o", markersize=3, capsize=4,
ax[3].plot(xx, us_mean_enkf[:, end], "--", color="C1", label="EnKF")
ax[3].plot(xx, sqrt.(diag(us_cov_enkf[:,:,end])), "--", color="C2", label="EnKF std")
ax[3].plot(xx, us_mean_eakf[:, end], "--", color="C3", label="EAKF")
ax[3].plot(xx, sqrt.(diag(us_cov_eakf[:,:,end])), "--", color="C4", label="EAKF std")

ax[3].set_ylabel("u")
ax[3].set_xlabel("x")
fig.tight_layout()
fig.savefig("adv-DA-traj-3.pdf")
```



ref
EnKF mean
EnKF std
EAKF mean
EAKF std
obs
u
x
20
10
0
−10
−20
0
200
400
600
800
1000

```
In [ ]:
```