Liu-West Filter for a building Grey-box model

Sang woo Ham

1/28/2020

Contents

1. Introduction

This is accompanies code of *Appendix C* in paper "xxx". This is a simple demonstaration of Liu-West filter for a simple building grey-box model. We generated a synthetic data and applied Liu-West filter to see if the filter can be applicable for this problem. All code is written in `R` language. The main purpose of this document is to provide reproducible example.

The pacakge depedency of this code is managed by `renv` package. You can look at `renv.lock` file to see the required package. However, for the simplicity, just run following script on this rproject.

After clone the repository and run `building_lw_filter.Rproj` file (you must have Rstudio) with `R>3.5.3`.

```
git clone https://github.com/ecosang/building_lw_filter.git
```

In R console, run following script.

```r
# Run this code in Rstudio
install.packages('renv',repos="https://cran.rstudio.com")
renv::equip() #install required software
renv::restore()
```

Technically, this installs all required packages, and you can reproduce all codes below. However, if this doesn't work, please report issues.

All functions used in this code is in `code/utility.R`. Also, all generated data and trained model are stored in `data` folder.

2. Model description

2.1 A simple grey-box model with two states.

The grey-model is composed of two states. Below figure shows R-C circuit diagram of this model. All variables are listed below.

Variables

- x_{i}: indoor temperature state [°C]
- x_{e}: envelope temperature state [°C]
- T_{a}: outdoor air temperature [°C]
- y_{x_{i},t_k}: measured temperature of state x_{i} [°C]
- R_{ie}: thermal resistance between x_{i} and x_{e} [K/W]
- R_{ea}: thermal resistance between x_{e} and T_{a} [K/W]
- C_{e}: thermal capacitance of x_{e} [J/K]
- C_{i}: thermal capacitance of x_{i} [J/K]
- $\dot{Q}_{\mathrm{hc}}$: heating or cooling flow rate [W]
- t and t_k are time in seconds and its discretized time, respectively.
- $d\omega/dt$ is standard weiner process.
- σ_{i}^2 and σ_{e}^2 are process noise variance of states x_{i} and x_{e}, repsectively.
- $\sigma_{\mathrm{d,y}}^2$ is observational noise variance of measurement y.

2.2 Governing differential equations

The thermal dynamics are expressed with below system of differntial equations. Here, R is thermal resistance and C is thermal capacitance.

$$\frac{dx_{\mathrm{i}}}{dt} = \left(\frac{1}{R_{\mathrm{ie}}C_{\mathrm{i}}}(x_{\mathrm{e}} - x_{\mathrm{i}}) + \frac{1}{C_{\mathrm{i}}}\dot{Q}_{\mathrm{hc}}\right) + \sigma_i \frac{d\omega}{dt}$$

$$\frac{dx_{\mathrm{e}}}{dt} = \left(\frac{1}{R_{\mathrm{ie}}C_{\mathrm{e}}}(x_{\mathrm{i}} - x_{\mathrm{e}}) + \frac{1}{R_{\mathrm{ea}}C_{\mathrm{e}}} + (T_{\mathrm{out}} - x_{\mathrm{e}})\right) + \sigma_e \frac{d\omega}{dt}$$

$$y_{x_{\mathrm{i}},t_k} = x_{\mathrm{i},t_k} + \varepsilon_{y,t_k} \text{ where } \varepsilon_{y,t_k} \sim \mathcal{N}(0, \sigma_{\mathrm{d},y})$$

2.3 System equations in a probablistic format

The system of matrix can be expressed in matrix form.

$$\begin{bmatrix} \dot{x}_{\mathrm{i},t} \\ \dot{x}_{\mathrm{e},t} \end{bmatrix} = \underbrace{\begin{bmatrix} -\frac{1}{R_{\mathrm{ie}}C_{\mathrm{i}}} & \frac{1}{R_{\mathrm{ie}}C_{\mathrm{i}}} \\ \frac{1}{R_{\mathrm{ie}}C_{\mathrm{e}}} & -\frac{1}{R_{\mathrm{ie}}C_{\mathrm{e}}} - \frac{1}{R_{\mathrm{ea}}C_{\mathrm{e}}} \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x_{\mathrm{i},t} \\ x_{\mathrm{e},t} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & \frac{1}{C_{\mathrm{i}}} \\ \frac{1}{R_{\mathrm{ea}}C_{\mathrm{e}}} & 0 \end{bmatrix}}_{\mathbf{B}} \begin{bmatrix} T_{\mathrm{out},t} \\ \dot{Q}_{\mathrm{hc},t} \end{bmatrix} + \begin{bmatrix} \sigma_i \dot{\boldsymbol{\omega}}_t \\ \sigma_e \dot{\boldsymbol{\omega}}_t \end{bmatrix}$$

$$y_{x_{\mathrm{i}},t_k} = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} x_{\mathrm{i},t_k} \\ x_{\mathrm{e},t_k} \end{bmatrix} + \varepsilon_{y,t_k}$$

Since this system is a linear gaussian model, it can be discretized without integration (see Appendix B of paper) (or this link). Here, subscript $_d$ indicates discretization. $\mathbf{x}_{t_k} = [x_{\mathrm{i},t_k},\, x_{\mathrm{e},t_k}]^\intercal$, $\mathbf{u}_{t_k} = [T_{\mathrm{a}}, \dot{Q}_{\mathrm{hc}}]^\intercal$, and $\boldsymbol{\theta} = \{R_{\mathrm{ie}}, R_{\mathrm{ea}}, C_{\mathrm{e}}, C_{\mathrm{i}}, \sigma_{\mathrm{i}}, \sigma_{\mathrm{e}}\}$

$$P(\mathbf{x}_{t_k+1}|\mathbf{x}_{t_k}) = \mathcal{N}(\mathbf{x}_{t_k+1}|f_d\left(\mathbf{x}_{t_k}, \mathbf{u}_{t_k}, \boldsymbol{\theta}\right), \boldsymbol{\sigma}_{\mathrm{d},x})$$

$$P(y_{x_{\mathrm{i}},t_k}|\mathbf{x}_{t_k}) = \mathcal{N}(y_{x_{\mathrm{i}},t_k}|g_d\left(\mathbf{x}_{t_k}, \mathbf{u}_{t_k}, \boldsymbol{\theta}\right), \sigma_{\mathrm{d},y})$$

$$f_d\left(\mathbf{x}_{t_k}, \mathbf{u}_{t_k}, \boldsymbol{\theta}\right) = \mathbf{A}_{\mathrm{d}}\mathbf{x}_{t_k} + \mathbf{B}_{\mathrm{d}}\mathbf{u}_{t_k} \text{ and } g_d\left(\mathbf{x}_{t_k}, \boldsymbol{\theta}\right) = \mathbf{C}_{\mathrm{d}}\mathbf{x}_{t_k}$$

where $\mathbf{A}_{\mathrm{d}}$, $\mathbf{B}_{\mathrm{d}}$, and $\mathbf{C}_{\mathrm{d}}$ are discretized parameters of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ matrix given above.

3. Synthetic Data

3.1 True parameter

Based on the above model, we assigned true parameter values to generate synthetica data. From the synthetic data, we apply Liu-West filter, and the final posterior distribution of parameters will be compared to those true parameter values.

	$T_{\mathrm{i},0}$	$T_{\mathrm{e},0}$	C_{i}	C_{e}	R_{ie}	R_{ea}
true	21.0	15.0	500000	6000000	1/55	1/55

To assign, true values, we assume following properties.

- Total surface are of exterior wall and window are 200m^2 and 50m^2, respectively.
- U-value of exterior wall and window are 0.2W/m^2-K and 1.4W/m^2-K, respectively.
- The total R-value is $\frac{1}{UA_{\text{ext-wall}}+UA_{\text{window}}} = \frac{1}{110}$. Thus, we split 1/110 by 1/55 and 1/55 because only the sum of $UA_{\text{ext-wall}} + UA_{\text{window}}$ is important.
- Assuming all envelope mass is concerete. Specific heat (C_p: 1000 J/kg-K) and density (ρ: 2400 kg/m^3). Volume with 0.1m thickness: $250\text{m}^2 \times 0.1\text{m} = 25\text{m}^3$.
- The exterior wall heat capacitance (C_{e}): $C_p \times \rho \times V \approx 6000000$ J/K
- Assuming air specific heat is 1J/K-m^3 and volume 1000 m^3, the indoor heat capacitance (C_{i}) is 500000 J/K.

Noise parameters

$\sigma_{\mathrm{d},y}$	σ_{x_i}	σ_{x_e}
0.25	$0.01/\sqrt{900}$	$0.01/\sqrt{900}$

Here, $\sigma_{\mathrm{d},y}$ is set to 0.25 becuase the sensor measurement accuracy is $\pm 0.5°$C. Specifically, $\mathcal{N}(0, 0.25)$ generates data about $[-0.5, 0.5]$ thinking 95% quantiles.

The continuous process noise σ_i and σ_e are set to 0.25/30, respectively. When we discretize the continuous system, the discretized standard deviation is approximately an order of $\sqrt{\Delta t}$. Therefore, assuming our process noise is 0.25 in a discrete time-scale, it is approximately $0.01/\sqrt{\Delta t}$. Our dataset has 900s time-scale. Therefore, they are set to $0.01/\sqrt{900}$

In the model, we will use fixed value of $\sigma_{\mathrm{d},y}$ because the measurement noise value can be obtained from the sensor information. Actually, this is helpful to stabilize Liu-West filter operation.

3.2 Read input data and generate synthetic data

Based on input data ($\mathbf{u}_{1:t_K}$), we will generate synthetic observation data by using a stochastic simulation.

```r
# load functions
source("code/utility.r")
# load input u data.
dat=readr::read_csv("data/syn_data.csv")
dt=dat$time[2]-dat$time[1] #discrete time
# define all true parameters
x0_true=c(21.0, 15.0) # initial states

#           Ci     Ce        Rie       Rea  #sigma_x_i #sigma_x_e
par_true=c(500000, 6000000,  1/55,     1/55, 0.01/sqrt(dt),      0.01/sqrt(dt) )
dims=c(2,2,1) # dimension of x, u, y, which is [nx, nu, ny]
nx=dims[1] # x dim
nu=dims[2] # u dim
ny=dims[3] # y dim
t_K=dim(dat)[1] #total time time is (1:t_K)

# create u_matrix [nu x t_K], (T_a, Qhc)
u_mat=rbind(t_a=dat$t_a,q_hc=dat$Q_hc) #input u matrix [nu x NT].
# weplit data into two parts to separate prior generation part, filter part.
index_split=ceiling(t_K/2)
```

Create data synthetic dataset. The data is created once and stored into `data/synthetic_data.rds` fiel. Then, it is loaded for next time because we generate the data with random noise from $\sigma_{x_{\mathrm{i}}}$, $\sigma_{x_{\mathrm{e}}}$, and $\sigma_{\mathrm{d},y}$. Here the dataset are splitted into two parts. The first part (`prior_data`) is used to create priors. The second part (`filter_data`) is used to used particle filtering process.

```r
## not run this code
# create discretized system matrix
dsys=discretize_system(par=par_true,dims=dims,dt=dt)
# create data set with stochastic process
simulated_data=create_data_set(dsys=dsys,u_mat=u_mat,x0=x0_true,dims=dims,dt=dt,seed_num=1234,stochastic
y_t_i=simulated_data$y_mat[1,] # observation data

# split data by train_data, test_data, all_data
prior_data=list(u_mat=u_mat[,1:index_split],
                y_mat=y_t_i[1:index_split])
filter_data=list(u_mat=u_mat[,(index_split+1):t_K],
                y_mat=y_t_i[(index_split+1):t_K])
all_data=list(u_mat=u_mat,
                y_mat=y_t_i)

# store synthetic data.
write_rds(list(prior_data=prior_data,filter_data=filter_data,all_data=all_data,index_split=index_split)
          paste0("data/synthetic_data.rds"))
```

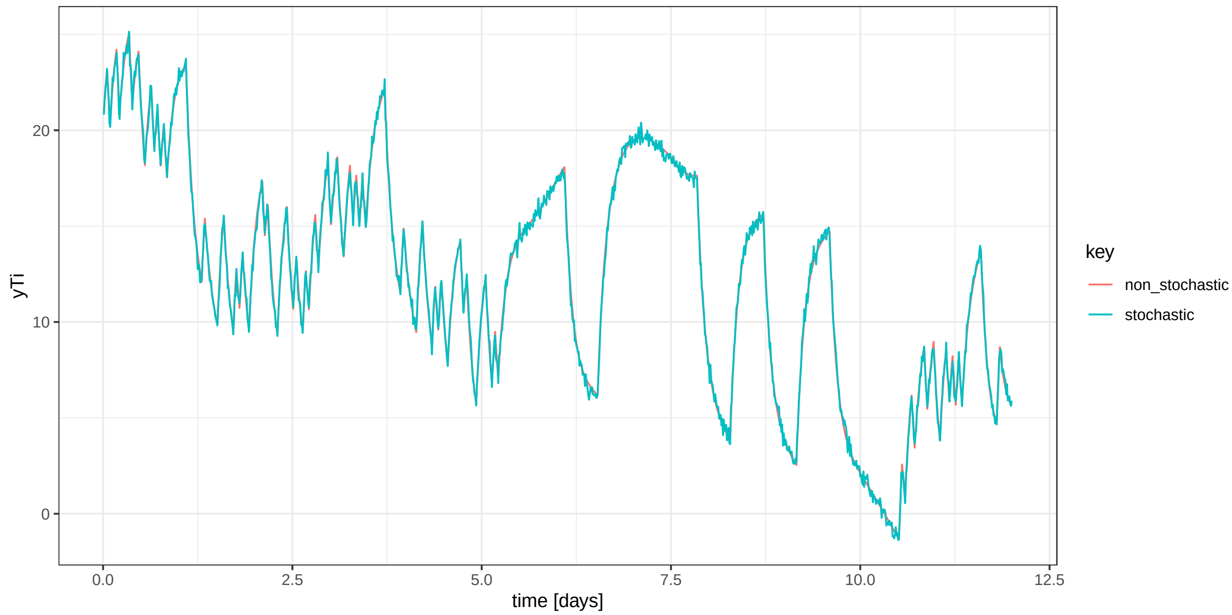
To see how it looks like, visualize the data with non-stochastic simulation.

```r
# load synthetic (stochastically simluated data)
synthetic_data<-readr::read_rds(paste0("data/synthetic_data.rds"))
# load data
prior_data=synthetic_data$prior_data
filter_data=synthetic_data$filter_data
all_data=synthetic_data$all_data
index_split=synthetic_data$index_split
```

```
dsys=discretize_system(par=par_true,dims=dims,dt=dt)

non_stochastic_data=create_data_set(dsys=dsys,u_mat=u_mat,x0=x0_true,dims=dims,dt=dt,seed_num=1234,stoch
```

```
## [1] "non-stochastic"
```

```
# Create dataframe for visualization.
plot_df=tibble(non_stochastic=as.numeric(non_stochastic_data$y_mat),stochastic=as.numeric(synthetic_data
  mutate(time=row_number()/4/24)

# plot
ggplot(plot_df%>%gather(key,yTi,-time),aes(time,yTi))+geom_line(aes(color=key))+xlab("time [days]")+ther
```



4. Create priors

As described in section 3.4 Model initialization (Prior generation) in the paper, we created priors to initialize Liu-West filter.

Define parameter lower and upper bounds that create priors.

4.1 Upper and lower bounds of states and parameter

	$T_{i,0}$	$T_{e,0}$	C_i	C_e	R_{ie}	R_{ea}
true	21.0	15.0	500000	6000000	1/55	1/55
min	0	0	100000	1000000	1e-9	1e-9
max	30.0	30.0	1000000	10000000	0.1	0.1

```
# define upper and lower bounds of x0 and theta for prior_data.
max_x_par=c(c(30,30,c(1000000,10000000,.1,0.1)))
min_x_par=c(10,0,100000,1000000,1e-9,1e-9)
```

4.2 Optimization to get set of parameters (Eq. 14)

d

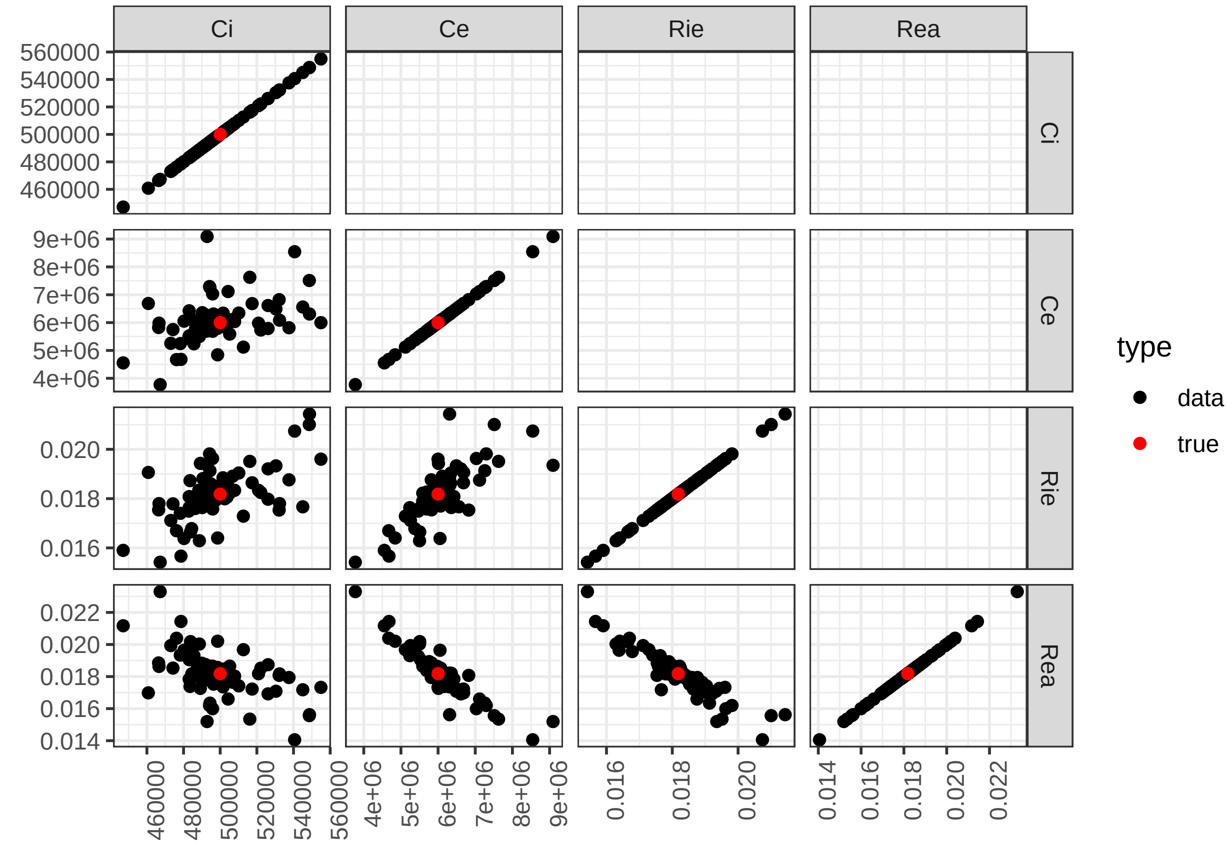
```
# sometimes optimization fails because cost is NaN. Thus, have this trycatch loop with while so that th
# check optimizer runs. Single run.
## library(kfsang) Don't use this library. This package is written by me to have cpp function of nstep
# ss=DEoptim::DEoptim(fn=get_rmse, lower=rep((-0.5+1e-9),length(L_val)),
#                       upper=rep(0.5,length(L_val)),control=list(NP=500, itermax=10000,trace=TRUE,relto
#                                                  parVar=list("discretize_system","nstep_cpp","expm")
#                       dt=dt,u_mat=train_data$u_mat,y_mat=train_data$y_mat,dims=dims,output="cost",norm

sol_list=list()
iii=1
while(length(sol_list)<=100){
  set.seed(iii+30000)
  # sometimes optimizer fails since it gives NaN during the nstep-ahead prediction. Thus, we use try-ca
  try_result=tryCatch(DEoptim::DEoptim(fn=get_rmse, lower=min_x_par/max_x_par,
                      upper=max_x_par/max_x_par,
                      control=list(NP=100, itermax=10000,trace=TRUE,reltol=1e-16,steptol=15,parallelType=
                                                 parVar=list("discretize_system","nstep","expm")),
                      dt=dt,u_mat=prior_data$u_mat,y_mat=prior_data$y_mat,dims=dims,output="cost",normali
                      ,
                        error=function(e){"error"} )

  if((try_result=="error")){

  }else{
    sol_list[[(length(sol_list)+1)]]=try_result
  }
  iii=iii+1
}
# store solution.
write_rds(sol_list,paste0("data/sol_list.rds"))
```

4.3 Visualize obtained set of parameters

To run Liu-West filter on the `filter_data`, we need both prior for initial states and parameters. In 3.3.2, initial states and parameters of `prior_data` are obtained from optimizer. The initial states of `filter_data` is actually the last states of `prior_data`. Therefore, we do `n-step` ahead prediction on `prior_data` with set of parameters from the obtimizer to have initial states of `filter_data`.

```
# load results
sol_list=readr::read_rds(paste0("data/sol_list.rds"))


# sol_list contains intial states and parameters of prior_data.
# we will have initial state and parameters for filter_data in sol_list_temp
sol_list_temp=list()

for (i in 1:length(sol_list)){
  sol_list_temp[[i]]=(sol_list[[i]]$optim$bestmem)*max_x_par #extract optimizer solution and unnormaliz
  # store n-step ahead prediction result
  # 1:index_split: prior data, index_split+1:index_split*2: filter_data
  tempp=get_rmse(par=sol_list_temp[[i]],dt=dt,
```

```r
                 u_mat=all_data$u_mat[,1:(index_split+1)],
                 y_mat=all_data$y_mat[1:(index_split+1)],
                 dims=dims,output="data",normalize=FALSE)
  # put indepx_split+1 states into initial states into sol_list.
  sol_list_temp[[i]][1:2]=tempp$x_pred[,index_split+1]
}


# sol_list_temp to dataframe
all_df=do.call(rbind,sol_list_temp)%>%as_tibble()

state_names<-c("Ti0","Te0")
par_name<-c("Ci","Ce","Rie","Rea")
# split data_frame into state/parameter
par_df=all_df[,-c(1:2)]%>%set_names(par_name)
state_df=all_df[,1:2]%>%set_names(state_names)
# data_frame for visualization
par_df_plot<-par_df%>%mutate(type="data")
# add true parameter into data frame for visualization
par_true_df=par_true[1:length(par_name)]

names(par_true_df)<-par_name
par_true_df=par_true_df%>%as.list()%>%as_tibble()%>%mutate(type="true")

par_df_plot=bind_rows(par_df_plot,par_true_df)


library(ggforce)

cols <- c("data" = "black", "true" = "red")
# Ti,0  Te,0,   Ce, Ci  Rea Rie
ggplot(par_df_plot, aes(x = .panel_x, y = .panel_y,color=type,fill=type)) +
  geom_point(alpha = 1.0, shape = 16, size = 2.0) +
  facet_matrix(vars(-type),layer.lower = T,layer.diag = T, layer.upper = F)+
  scale_colour_manual(values = cols)+theme_bw()+theme(axis.text.x = element_text(angle = 90))
```



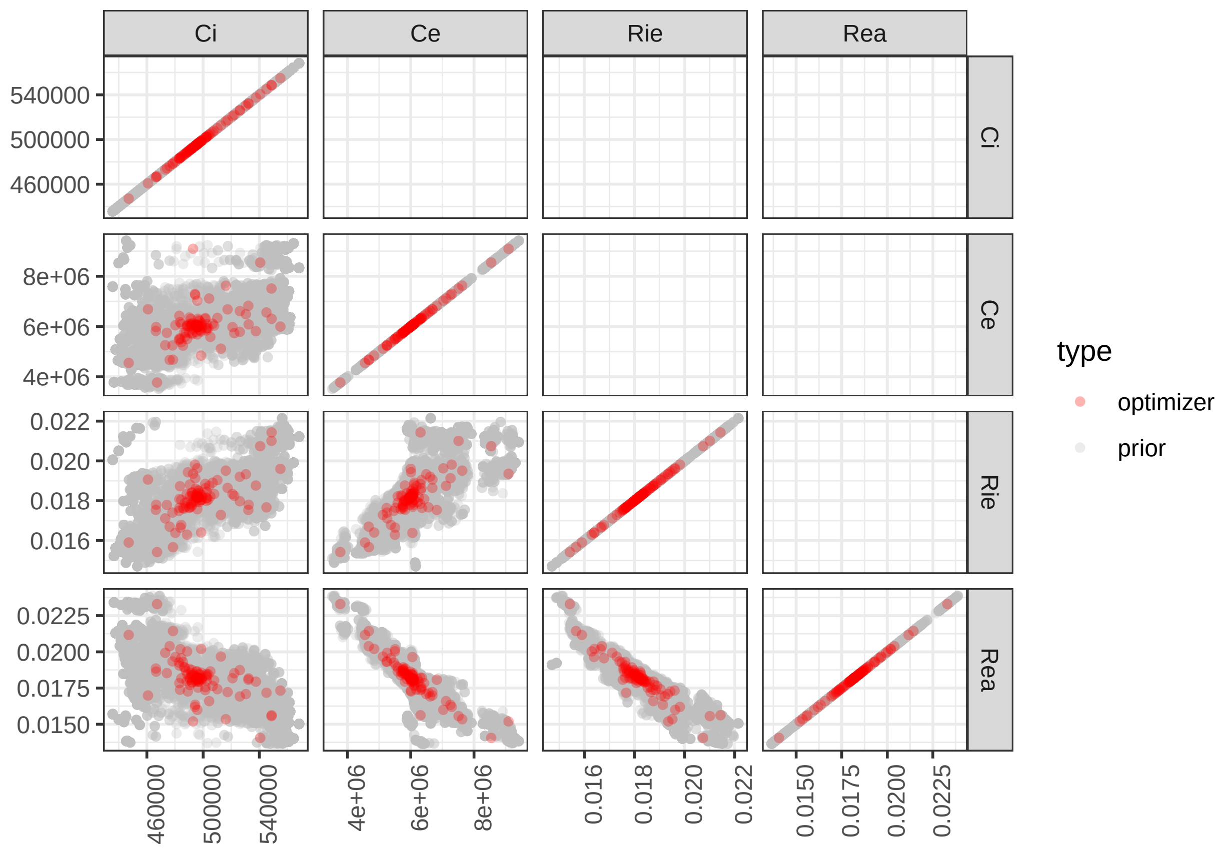
4.4 Kernel density approximation of set of pameters

```
suppressPackageStartupMessages(library(kdevine))
# multivariate kernel density approximation
kde_par <- kdevine::kdevine(as.matrix(par_df)) #kernel model K(x)
kde_state <- kdevine::kdevine(as.matrix(state_df)) #kernel model K(theta)

# Generate
set.seed=104
NP=10000
prior_par=Rfast::transpose(abs(kdevine::rkdevine(NP, kde_par)))
prior_state=Rfast::transpose(kdevine::rkdevine(NP, kde_state))


write_rds(kde_par,paste0("data/kde_par.rds"))
write_rds(kde_state,paste0("data/kde_state.rds"))
write_rds(prior_par,paste0("data/prior_par.rds"))
write_rds(prior_state,paste0("data/prior_state.rds"))
```

4. Kernel density approximation of set of pameters

```
# load generated prior data from the kernel density approximation
prior_par=readr::read_rds(paste0("data/prior_par.rds"))
prior_state=readr::read_rds(paste0("data/prior_state.rds"))
NP=dim(prior_par)[2]
```

```r
# permutate gerenated prior to have uniform weight in a wide range
min_val=floor(min(prior_par[1,])) # minimum bounds of single parameter elemenet
max_val=ceiling(max(prior_par[1,])) # maximum bounds of single parameter elemenet
min_max_grid=seq(min_val,max_val,length.out=NP) # parameter grids based on the min/max range.
permutation_grid=sapply(1:NP,function(x) which.min(abs(prior_par[1,]-min_max_grid[x])))
prior_par=prior_par[,permutation_grid]
prior_state=prior_state[,permutation_grid]

# create data_frame for visualization
prior_par_df=prior_par%>%Rfast::transpose()%>%
  as.data.frame()%>%as_tibble()%>%set_names(par_name[1:(dim(prior_par)[1])])

par_df_plot2=bind_rows(prior_par_df%>%mutate(type="prior"),par_df%>%mutate(type="optimizer"))

cols2 <- c("optimizer" = "red", "prior" = "grey75")
ggplot(par_df_plot2, aes(x = .panel_x, y = .panel_y,color=type,fill=type)) +
  geom_point(alpha = 0.3, shape = 16, size = 1.5) +
  facet_matrix(vars(-type),layer.lower = T,layer.diag = T, layer.upper = F)+
  scale_colour_manual(values = cols2)+
  scale_fill_manual(values = cols2)+
  theme_bw()+theme(axis.text.x = element_text(angle = 90))
```



5. Liu-West filter

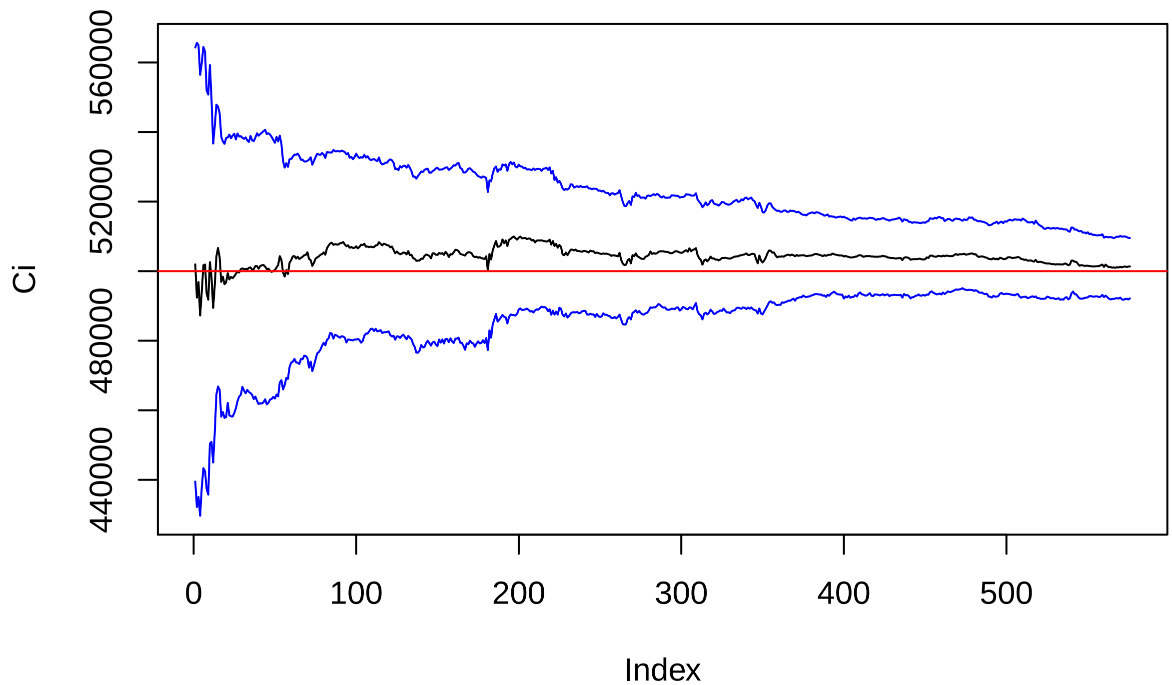
Since we have priors, we can run Liu-West filter. For complete algorithm of the filter, refer Table 2 in the paper.

```
# L_values of parameter to normalize parameters
L_val_par=apply(prior_par,MARGIN=c(1),max)
prior_par_n=(prior_par)/L_val_par-0.5 #normalized parameter
num_state=dim(prior_state)[1] # number of state


# define L_values of noise parameter (sigma_x_i,sigma_x_e)
L_val_sd=rep(.1/sqrt(dt),num_state)
L_val=c(L_val_par,L_val_sd) #L_val for parameter +noise parameter

# normalized priors for noise parameter by using Ratin-Hyper cube sampling
set.seed(1234)
prior_sd_n=(Rfast::transpose(lhs::randomLHS(NP,length(L_val)-(dim(prior_par_n)[1] ))))-0.5

# all normalized parameters.
prior_all_n=rbind(prior_par_n,prior_sd_n)
num_par=dim(prior_all_n)[1]

# initial x0
x=prior_state #x0
pii=w=rep(1/NP,NP) #initilal weight

# inputs for the model
inputs=list()
inputs$L_val=L_val #L_val for normalization/unnormalization.
inputs$dt=dt #time interval

inputs$y_mat=filter_data$y_mat # y_data
inputs$u_mat=filter_data$u_mat # input u_data

# parameter names
par_names=c("Ci","Ce","Rie","Rea","sdte","sdti")
inputs$par_names=par_names

# initial parameter prior particles
theta=prior_all_n
delta=0.9 # Filter tuning parameter
seed_num=13244 #seed number
```

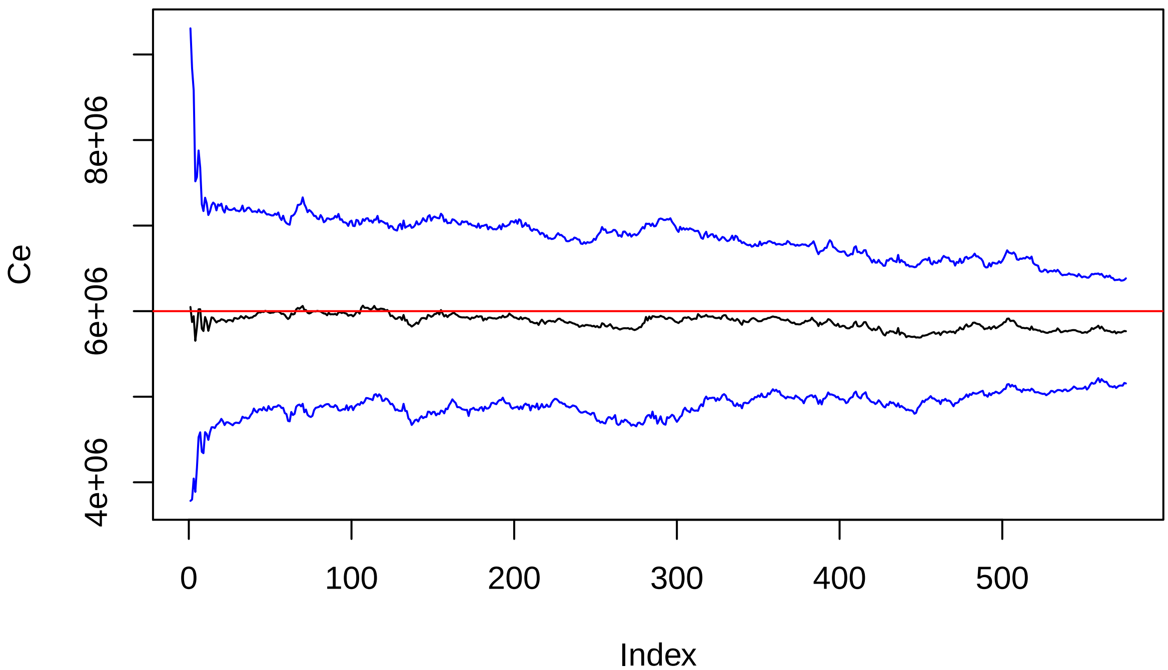
```
# seed=seed_num
#res=lw_ss_filter(inputs=inputs,NP=NP,pii=pii,x=x,theta=theta,wt=wt,seed=seed_num,training=TRUE)
res=lw_ss_filter(inputs=inputs,NP=NP,pii=pii,x=x,theta=theta,delta=delta,seed=seed_num,training=TRUE)



for(i in 1:dim(res$r_thetas)[1]){
  write_rds(res$r_thetas[i,,],paste0("data/res_thetas_",i,".rds"))
}
for(i in 1:dim(res$xs)[1]){
  write_rds(res$xs[i,,],paste0("data/res_xs_",i,".rds"))
```

```r
}

res$thetas<-NULL
res$r_thetas<-NULL
res$xs<-NULL
write_rds(res,"data/res.rds")
```

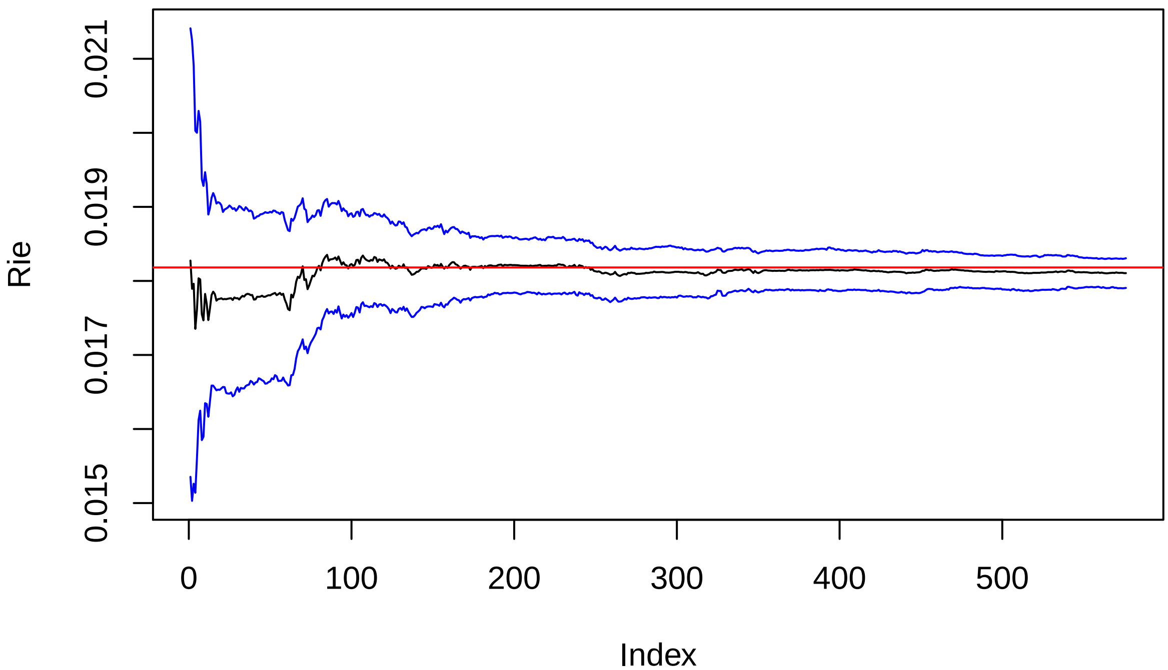
```r
res=read_rds("data/res.rds")

res$r_thetas=array(data=0,dim=c(length(par_names),NP,dim(filter_data$u_mat)[2]))
res$xs=array(data=0,dim=c(length(state_names),NP,dim(filter_data$u_mat)[2]))
for (i in 1:length(par_names)){
  res$r_thetas[i,,]=read_rds(paste0("data/res_thetas_",i,".rds"))
}

for (i in 1:length(state_names)){
  res$xs[i,,]=read_rds(paste0("data/res_xs_",i,".rds"))
}
# res$xs[1,,]%>%colMeans%>%plot(type='l')
# filter_data$y_mat%>%points(type='l',col="red")
# res$xs[2,,]%>%colMeans%>%plot()



#theta_quantile=res$thetas%>%apply(MARGIN=c(2,3),function (x) quantile(x,c(0.025,0.5,0.975)))
rtheta_quantile=res$r_thetas%>%apply(MARGIN=c(1,3),function (x) quantile(x,c(0.025,0.5,0.975)))
#par_names=names(par_recover_lw(1,res$lw_mean,res$lw_sd))
par_name=c("Ci","Ce","Rie","Rea")

for (i in 1:4){
  theta_idx=i
  plot(rtheta_quantile[2,theta_idx,],col="black",type="l",ylim=c(min(rtheta_quantile[,theta_idx,],par_t
       ylab=paste0(par_name[theta_idx]))
  lines(rtheta_quantile[1,theta_idx,],col="blue")
  lines(rtheta_quantile[3,theta_idx,],col="blue")
  abline(h=par_true[theta_idx],col="red")
}
```



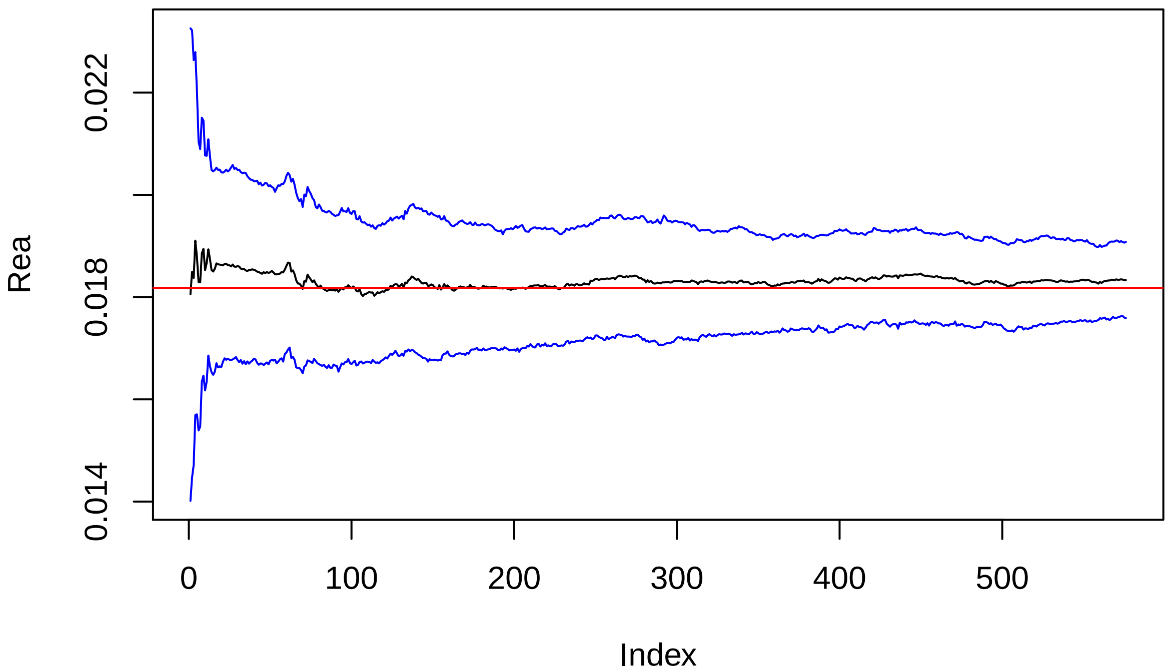
Ci
440000
480000
520000
560000
0
100
200
300
400
500
Index



Ce
4e+06
6e+06
8e+06
0
100
200
300
400
500
Index



Rie
0.015
0.017
0.019
0.021
0
100
200
300
400
500
Index


Rea
0.014
0.018
0.022
0
100
200
300
400
500
Index