Filteling and Smoothing

稗田尚弥

2017-06-01

去年やっていたことの復讐
POMP に逃げる前に、どこまで行っていたのか
どこでつまずいたのか確認
確認していたら、粒子フィルタリングじゃなくてモンテカルロフィルタになっていること判明。

というか、そもそも何やってたか忘れた
提案分布をシュミレーションしている AR モデルと同じ分布としている.

パッケージの読み込み

ところどころ、開始からの時間を出力しておく

```
library(rgl)
library(mvtnorm)
library(reshape2)
library(ggplot2)
library(doSNOW)
library(pforeach)
rm(list=ls())
start_time <- Sys.time()
```

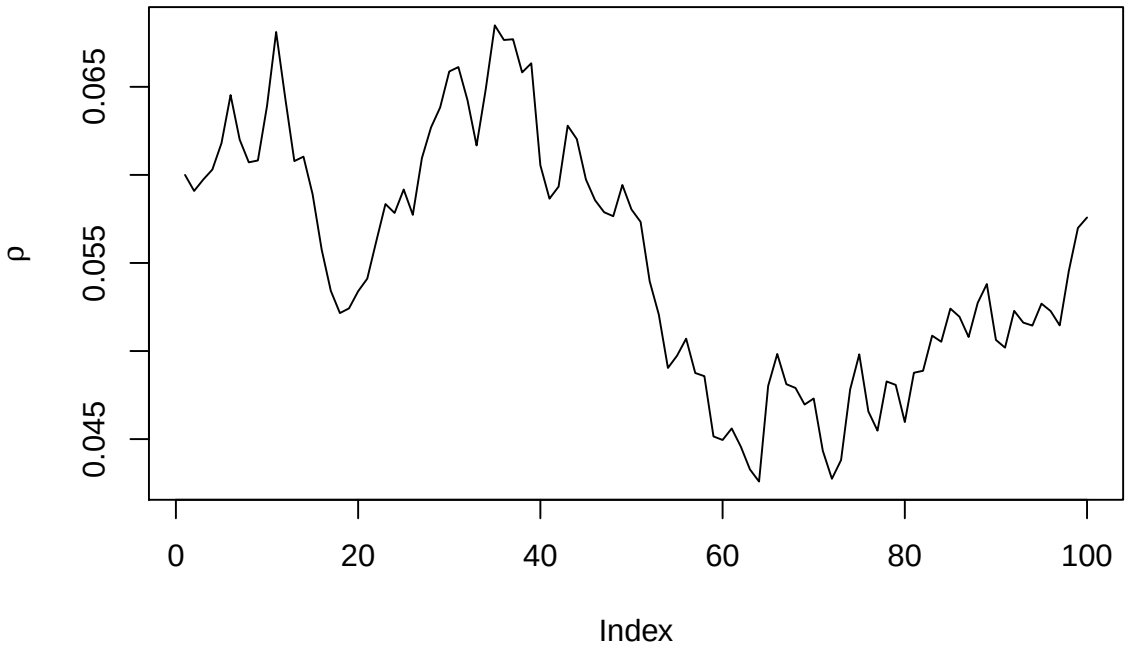
とりあえず、AR モデルにに従って、PDとρを生成して、DRを Hull の密度関数に従って、棄却法で発生

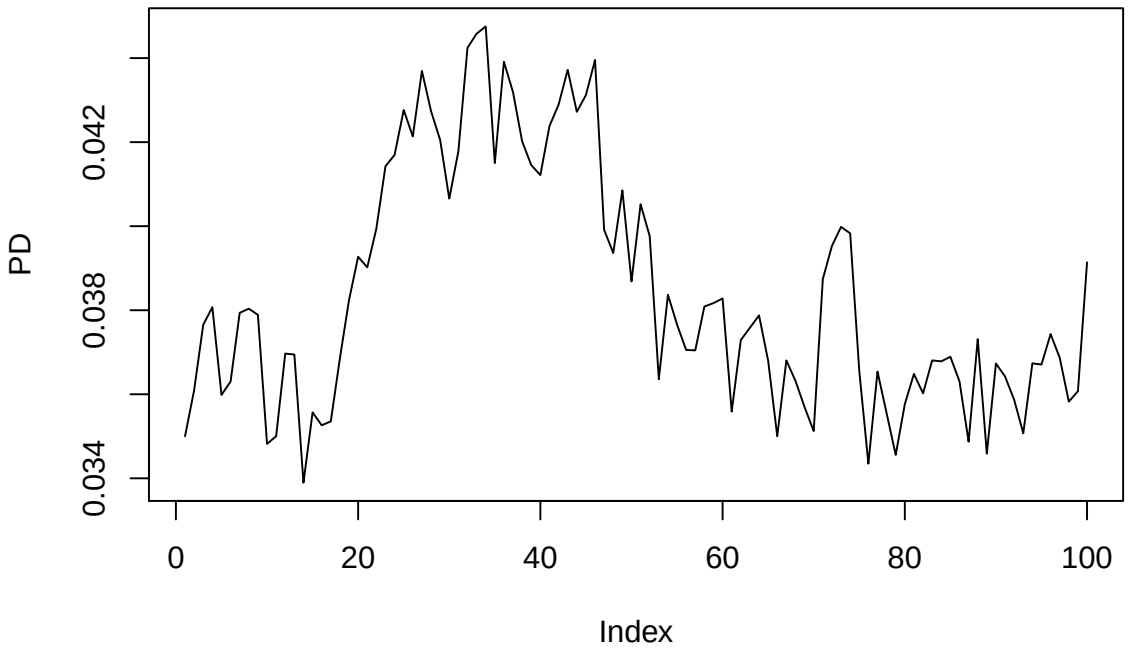
シミュレーション

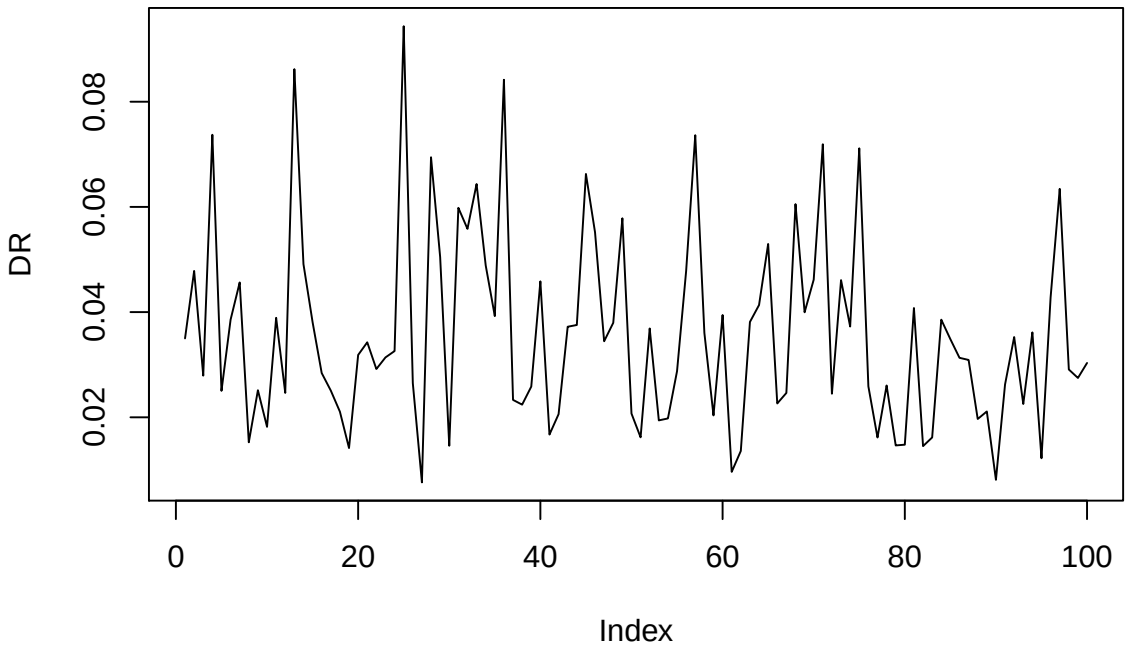
```
source("../script/データ編集.R", encoding = "UTF-8")
source('../script/DR_density.R', encoding = 'UTF-8')
source('../script/AR_sim.R', encoding = 'UTF-8')
set.seed(708)
answer<-AR_sim(time=100,rho = 0.06,PD=0.035)
```

```
##         rho         PD
## 1 0.06000000 0.03500000
## 2 0.05908945 0.03609709
## 3 0.05973462 0.03765537
## 4 0.06030230 0.03807128
## 5 0.06180245 0.03598501
```

6 0.06453122 0.03630226








```
head(answer)
```

```
##          rho         PD         DR
## 1 0.06000000 0.03500000 0.03500000
## 2 0.05908945 0.03609709 0.04783225
## 3 0.05973462 0.03765537 0.02792155
## 4 0.06030230 0.03807128 0.07372085
## 5 0.06180245 0.03598501 0.02503292
## 6 0.06453122 0.03630226 0.03855752
```

```
data_for_Kalman<-data.frame(dt=c(1:length(answer$DR)),DR=answer$DR)
colnames(data_for_Kalman)<-c("dt","Default_Rate")
str(data_for_Kalman)
```

```
## 'data.frame':    100 obs. of  2 variables:
##  $ dt          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Default_Rate: num  0.035 0.0478 0.0279 0.0737 0.025 ...
```

```
Sys.time()-start_time
```

```
## Time difference of 20.46576 secs
```

初期設定

最適化を行う際に、変数に [0,1] の制約があると不便なので、シグモイド関数の逆関数で変換したものを利用する。今後、これをで $\theta = \{\theta_{PD}, \theta_{rho}\}$ 表す。シグモイド関数と逆関数を準備。

```
sig<-function(x){(tanh(x)+1)/2}
sig_env<-function(y){(1/2)*log(y/(1-y))}
```

Particlefilter を行う上で必要な初期値を、正規分布からサンプリング
サンプリングの平均値は、PD, ρ のシミュレーションの初期値を、シグモイド逆関数にかけたもの
分散はどちらも 0.01、共分散は 0 でサンプリングする

```
first_theta_m<-c(sig_env(0.06),sig_env(0.035))
first_theta_v<-matrix(c(0.01,0,0,0.01),ncol=2)
```

フィルタリングの際に発生した Particle を記録しておく、変数の準備
フィルタリング用いる分布 (提案分布だとか重点分布という名前) の分布は、
シミュレーションした AR モデルと同じ
・・・去年、それでも構わないと思っていたけど、これだと粒子フィルタリングじゃなくてモンテカルロフィルタリングになっている。

```
theta<-c()
state<-c()
theta_mu<-c(sig_env(0.06),sig_env(0.035))
theta_tau<-c(0.99,0.99)
theta_sigma<-matrix(c(0.0005,0,0,0.0004),ncol=2)
```

フィルタリング

パーティクルの数は、128 個
先ほど設定した初期値で、θ の初期値を発生。状態変数への変換も行う。

```
N <- 1000
proposal_theta_m <- theta_mu
proposal_theta_v <- theta_sigma
theta_0<-rmvnorm(N , mean=theta_mu , sigma = theta_sigma)
state_0<-sig(theta_0)
```

PD や rho の初期値の中に、0 や 1 のものがあると、DR の密度が計算できないので、そこだけ変換

```
state_0[,1][state_0[,1]==0] <- 1e-60
state_0[,2][state_0[,2]==0] <- 1e-60
state_0[,1][state_0[,1]==1] <- 0.9999999
state_0[,2][state_0[,2]==1] <- 0.9999999
weight_0<-rep(1/N,N)
```

時点 1 でのフィルタリング開始

AR モデルに従って t=1 での状態変数の Particle 発生

```
theta<-t(apply(theta_0,1,function(x) rmvnorm(1,mean=theta_mu+theta_tau*(x-theta_mu),sigma = theta_sigma)))
state<-sig(theta)
#PD や rho は 0 と 1 を取れないので、そこだけ変換
state[,1][state[,1]==0] <- 1e-60
state[,2][state[,2]==0] <- 1e-60
state[,1][state[,1]==1] <- 0.9999999
state[,2][state[,2]==1] <- 0.9999999
```

重みを計算

提案分布=実際の分布で、モンテカルロフィルタになっているので、重みを計算する際の分母分子が消える

```
weight<-apply(state,1,function(x) g_DR.fn(rho=x[1],PD=x[2],data_for_Kalman[1,2]))
weight<-weight/sum(weight)
state<-cbind(state,weight=weight)
```

累積相対尤度 (重み) の計算

状態変数と、重み、累積相対尤度がくっついたデータフレームが出来ていることを確認

```
state<-cbind(state,ruiseki=cumsum(state[,3])/sum(state[,3]))
head(state,n=10)
```

```
##                              weight     ruiseki
##  [1,] 0.05889548 0.03571173 0.0010240453 0.001024045
##  [2,] 0.05873341 0.03476612 0.0010148415 0.002038887
##  [3,] 0.06024002 0.03377412 0.0009866088 0.003025496
##  [4,] 0.06502666 0.03420325 0.0009514004 0.003976896
##  [5,] 0.05742771 0.03471218 0.0010269027 0.005003799
##  [6,] 0.06015072 0.03500865 0.0010044208 0.006008219
##  [7,] 0.06208918 0.03372076 0.0009693023 0.006977522
##  [8,] 0.06106500 0.03982895 0.0010218952 0.007999417
##  [9,] 0.05712959 0.03555773 0.0010399133 0.009039330
## [10,] 0.06095315 0.03505726 0.0009975844 0.010036915
```

```
tail(state,n=10)
```

```
##                               weight    ruiseki
##  [991,] 0.05994857 0.03275871 0.0009717712 0.9910396
##  [992,] 0.06184192 0.03922952 0.0010149308 0.9920545
##  [993,] 0.06426672 0.03432256 0.0009593220 0.9930138
##  [994,] 0.06324356 0.03365484 0.0009583671 0.9939722
##  [995,] 0.06673872 0.03472261 0.0009441432 0.9949163
##  [996,] 0.05691694 0.03737927 0.0010562479 0.9959726
##  [997,] 0.06107280 0.03348539 0.0009745099 0.9969471
##  [998,] 0.06032093 0.03463452 0.0009981745 0.9979453
##  [999,] 0.05928935 0.03842688 0.0010369120 0.9989822
```

```
## [1000,] 0.06027556 0.03649702 0.0010178193 1.0000000
```

リサンプリング関数の設定

一様分布 (0,1) から発生した乱数と累積相対尤度を用いてサンプリング

```
A_r<-function(r,state){
  for(i in N:1){
    if(r>=state[i,4]){
      return(i+1)
}
    }
    return(1)
}
```

リサンプリングする必要があるかどうか判定した上で、リサンプリング

リサンプリングしたかどうかのチェック変数も用意。リサンプリングしたら 1、しなかったら 0。

```
re_state_num<-c()
if(sum(weight^2)^-1 < N/10){
  re_state_num <- apply(data.frame(i = 1:N, x = runif(N)),1,
                        function(x) A_r((x[1]-1+x[2])/N),state)
  re_state <- state[re_state_num,c(1,2,3)]
  re_theta <- theta[re_state_num,]
  re_state[,3] <- rep(1/N,N)
  resample_check <- 1
} else{
  re_state_num<-seq(1:N)
  re_state<-state[,c(1,2,3)]
  re_theta<-theta
  resample_check <- 0
}
```

結果の保存

state_100 にリサンプリングする前の状態変数と、重みと、リサンプリングした状態変数のデータのリスト

```
state_100<-list(data.frame(rho=state[,1],
                           PD=state[,2],weight=weight,
                           re_rho=re_state[,1],
                           re_PD=re_state[,2],
                           re_weight=re_state[,3]))
head(state_100[[1]],n=10)
```

```
##         rho        PD       weight     re_rho      re_PD     re_weight
## 1  0.05889548 0.03571173 0.0010240453 0.05889548 0.03571173 0.0010240453
## 2  0.05873341 0.03476612 0.0010148415 0.05873341 0.03476612 0.0010148415
## 3  0.06024002 0.03377412 0.0009866088 0.06024002 0.03377412 0.0009866088
## 4  0.06502666 0.03420325 0.0009514004 0.06502666 0.03420325 0.0009514004
```

```
## 5  0.05742771 0.03471218 0.0010269027 0.05742771 0.03471218 0.0010269027
## 6  0.06015072 0.03500865 0.0010044208 0.06015072 0.03500865 0.0010044208
## 7  0.06208918 0.03372076 0.0009693023 0.06208918 0.03372076 0.0009693023
## 8  0.06106500 0.03982895 0.0010218952 0.06106500 0.03982895 0.0010218952
## 9  0.05712959 0.03555773 0.0010399133 0.05712959 0.03555773 0.0010399133
## 10 0.06095315 0.03505726 0.0009975844 0.06095315 0.03505726 0.0009975844
```

```
Sys.time()-start_time
```

```
## Time difference of 20.77941 secs
```

繰り返し処理の準備。並列処理で複数のコアをつかう準備。
簡単に言うと、各コアに並列処理で使う関数や変数を渡しておく
複数のコアを活用しているのは、Partilce の発生と、weight の計算と、リサンプリング

```
theta_100<-list(data.frame(re_theta))
cl <- makeCluster(rep('localhost', 4))
clusterExport(cl, c("dmvnorm","rmvnorm","A_r","N","g_DR.fn",
                    "theta_tau","theta_mu","theta_sigma","proposal_theta_v","runif"))
Sys.time()-start_time
```

```
## Time difference of 22.50128 secs
```

こっからは繰り返し

```
###2~
for(j in 2:length(data_for_Kalman[,1])){
  #前回の theta を事前分布の平均として利用
  post_theta<-re_theta
  theta<-c()
  state<-c()
  #Particle 発生
  theta<-t(parApply(cl,post_theta,1,function(x){
                  rmvnorm(1,mean=theta_mu+theta_tau*(x-theta_mu),
                        sigma=theta_sigma)}
                  )
          )
  state<-sig(theta)
  state[state[,1]<1e-60,1]<-1e-60
  state[state[,2]<1e-60,2]<-1e-60
  state[state[,1]>0.99999,1]<-0.99999
  state[state[,2]>0.99999,2]<-0.99999
  #重み計算
  weight_solution<-data.frame(state_100[[j-1]][,6],state,data_for_Kalman[j,2])
  weight<-parApply(cl,weight_solution,1,function(x){
    x[1]*g_DR.fn(rho=x[2],PD=x[3],DR=x[4])
      })
```

```
  weight<-weight/sum(weight)
  state<-cbind(state,weight=weight)
  state<-cbind(state,ruiseki=cumsum(state[,3])/sum(state[,3]))
  re_state_num<-c()
  if(sum(weight^2)^-1 < N/10){
    clusterExport(cl,"state")
    re_state_num<-parApply(cl,data.frame(i = 1:N, x = runif(N)),1,
                           function(x) A_r((x[1]-1+x[2])/N,state))
    re_state<-state[re_state_num,c(1,2,3)]
    re_theta<-theta[re_state_num,]
    re_state[,3]<-rep(1/N,N)
    resample_check <- 1
  } else{
    re_state<-state[,c(1,2,3)]
    re_theta<-theta
    resample_check <- 0
  }

  state_100<- c(state_100,list(data.frame(rho=state[,1],PD=state[,2],
                                          weight=weight,re_rho=re_state[,1],
                                          re_PD=re_state[,2],
                                          re_weight=re_state[,3])))
  theta_100<-c(theta_100,list(data.frame(re_theta)))
}
stopCluster(cl)
Sys.time()-start_time
```

```
## Time difference of 27.24659 secs
```

フィルタリング結果の図

各 Particle から期待値を計算して、それを状態変数の推定値とする。

```
tmp<-state_100
re_PD=c()
re_rho=c()
PD=c()
rho=c()
for(i in 1:length(tmp)){
  re_PD_tmp<-sum(tmp[[i]][,5]*tmp[[i]][,6])
  re_rho_tmp<-sum(tmp[[i]][,4]*tmp[[i]][,6])
  PD_tmp<-sum(tmp[[i]][,2]*tmp[[i]][,3])
  rho_tmp<-sum(tmp[[i]][,1]*tmp[[i]][,3])
  re_PD=c(re_PD,re_PD_tmp)
```

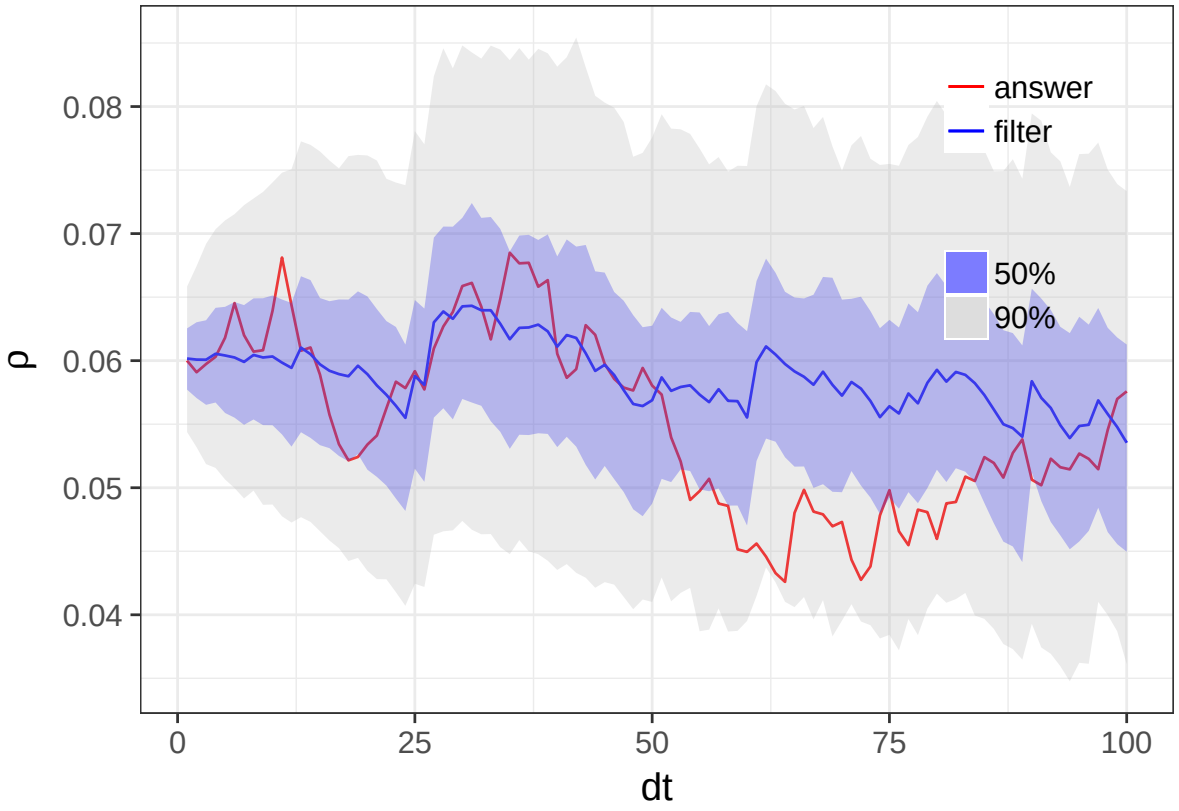
```
  re_rho=c(re_rho,re_rho_tmp)
  PD=c(PD,PD_tmp)
  rho=c(rho,rho_tmp)
}
```

5,25,75,95% 点を計算

```
qu<-data.frame(t(sapply(state_100,function(x) representative_value.fn(x[,c(4,5,6)]))))
```

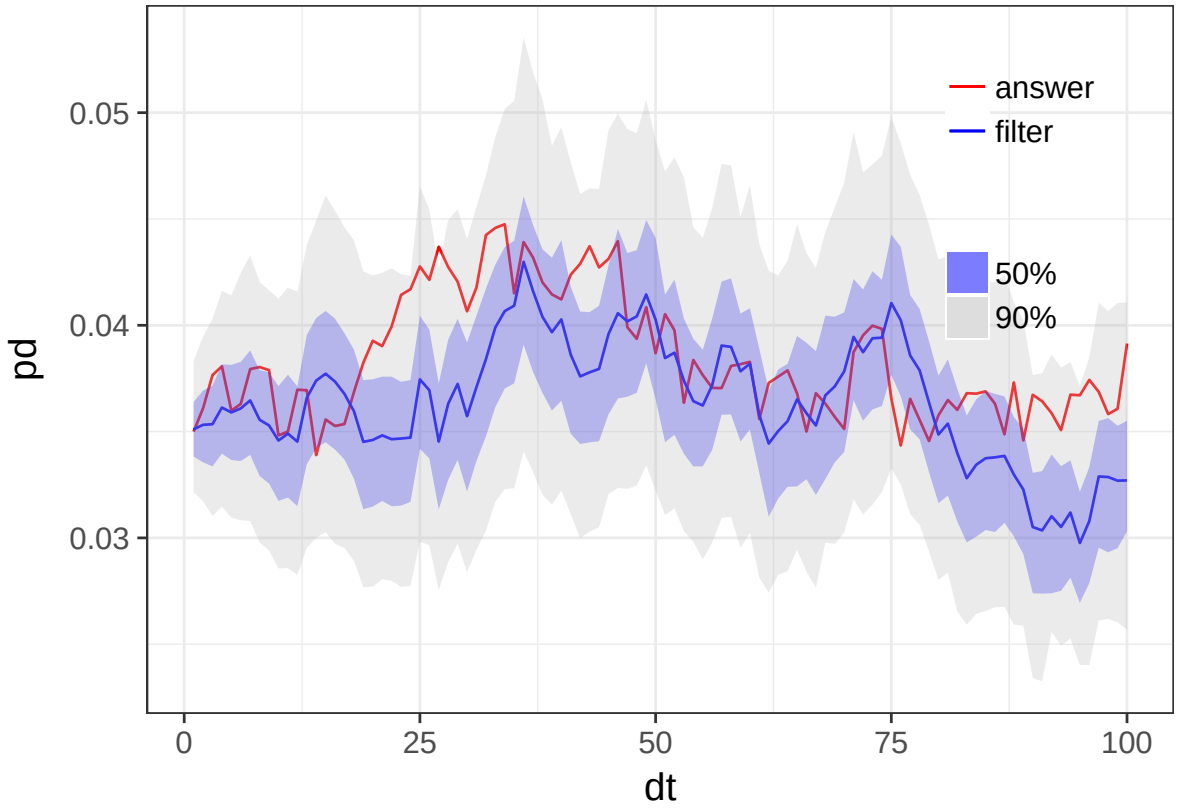
ρ の推定結果をデータフレームにまとめてプロット

```
plot_d<-data.frame(dt=c(1:length(data_for_Kalman[,1])),
                   answer=answer$rho,estimate_rho=re_rho,
                   two_fif_rho=qu$two_fif_rho,
                   twenty_fif_rho=qu$twenty_fif_rho,
                   seventy_fif_rho=qu$seventy_fif_rho,
                   nine_fif=qu$nine_fif_rho)

print(ggplot(plot_d,aes(x=dt))+
        geom_line(aes(y=answer,colour="answer"))+
        geom_line(aes(y=estimate_rho,colour="filter"))+
        geom_ribbon(aes(ymin=twenty_fif_rho, ymax=seventy_fif_rho,fill="50%"),alpha = 0.3)+
        geom_ribbon(aes(ymin=two_fif_rho, ymax=nine_fif,fill="90%"),alpha = 0.3)+
        theme_bw(15)+
        theme(legend.position=c(.85,.75),legend.background=element_blank())+
        ylab(expression(rho))+
        scale_color_manual(name='', values=c("answer" = "red", "filter" = "blue"))+
        scale_fill_manual(name='', values=c("50%" = "blue", "90%" = "gray")))
```



同様にして PD もプロット

```
plot_d<-data.frame(dt=c(1:length(data_for_Kalman[,1])),
                   answer=answer$PD,estimate_PD=re_PD,
                   two_fif_pd=qu$two_fif_pd,
                   twenty_fif_pd=qu$twenty_fif_pd,
                   seventy_fif_pd=qu$seventy_fif_pd,
                   nine_fif=qu$nine_fif_pd)

print(ggplot(plot_d,aes(x=dt))+geom_line(aes(y=answer,colour="answer"))+
      geom_line(aes(y=estimate_PD,colour="filter"))+
      geom_ribbon(aes(ymin=twenty_fif_pd, ymax=seventy_fif_pd,fill="50%"),alpha = 0.3)+
      geom_ribbon(aes(ymin=two_fif_pd, ymax=nine_fif,fill="90%"),alpha = 0.3)+
      theme_bw(15)+
      theme(legend.position=c(.85,.75),legend.background=element_blank())+
      ylab(expression(pd))+
      scale_color_manual(name='', values=c("answer" = "red", "filter" = "blue"))+
      scale_fill_manual(name='', values=c("50%" = "blue", "90%" = "gray")))
```




```
Sys.time()-start_time
```

```
## Time difference of 43.08113 secs
```

平滑化 Forward Filtering Backward Smoothing

フィルタリングの一番最後のウェイトを与える
リサンプリングの関数を平滑化用に修正
フィルタリングと同じようにコアの準備

```
weight_T<-list(state_100[[length(data_for_Kalman[,1])]][,6])
A_r<-function(r,weiht_n){
  for(i in N:1){
    if(r>=weight_n[i]){
      return(i+1)
    }
  }
  return(1)
}
cl <- makeCluster(rep('localhost', 4))
clusterExport(cl, c("dmvnorm","g_DR.fn","theta_tau","theta_mu","theta_sigma","proposal_theta_v"))
sm_state<-list(state_100[[length(data_for_Kalman[,1])]][,c(4,5,6)])
```

平滑化に関しては、コメント参照

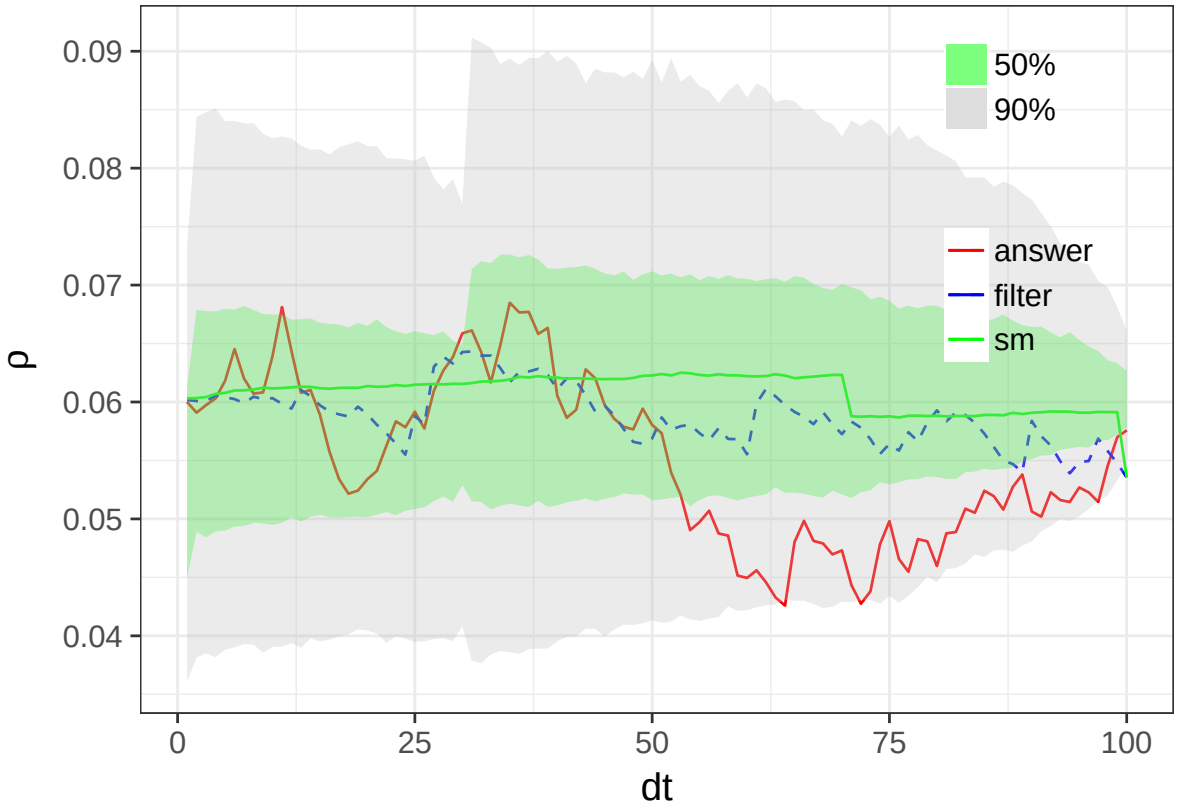
```
for(n in c(length(data_for_Kalman[,1])-1):1){
  weight_n<-c()
  #n+1 と n の状態変数，ウェイトを取得
  X_n_1<-data.frame(theta_100[[n+1]],weight=state_100[[n+1]][,6])
  X_n<-data.frame(theta_100[[n]],weight=state_100[[n]][,6])
  #上記二つと平滑化の n+1 番目のウェイトを取得
  X<-data.frame(X_n_1,X_n,weight_T[[length(data_for_Kalman[,1])-n]])
  colnames(X)<-c("n1_PD","n1_rho","n1_weight","n_PD","n_rho","n_weight","weight_T")
  for(i in 1:N){
    clusterExport(cl, c("X_n","X_n_1","i"))
    #平滑化ウェイトの分子計算
    bunsi<-parApply(cl,X,1,function(x)
      x[7]*dmvnorm(as.numeric(x[c(1,2)]),
                   theta_mu+(as.numeric(X_n[i,c(1,2)])-theta_mu)*
                     theta_tau,theta_sigma))
    #平滑化ウェイトの分母計算
    bunbo<-parApply(cl,X,1,function(x)
      x[6]*dmvnorm(as.numeric(X_n_1[i,c(1,2)]),
                   theta_mu+(x[c(4,5)]-theta_mu)*
                     theta_tau,theta_sigma))
    weight_n<-c(weight_n,X_n[i,3]*sum(bunsi/sum(bunbo)))
  }
  weight_n<-cumsum(weight_n)/sum(weight_n)
  #必要ならば平滑化ウェイトでリサンプリング　いったんなし
  #tmp<-runif(N,0,1)
  #sm_state_num<-parApply(i=1:N)
  #  pforeach(i = 1:N)({
  #  A_r(tmp[i])
  #})
  sm_state_num <- 1:N
  sm_state<-c(sm_state,
              list(data.frame(state_100[[n]][sm_state_num,c(4,5)],
                              weight_n[sm_state_num]/sum(weight_n[sm_state_num]))))
  weight_T<-c(weight_T,list(weight_n/sum(weight_n)))
}
stopCluster(cl)
Sys.time()-start_time
```

```
## Time difference of 2.402418 hours
```

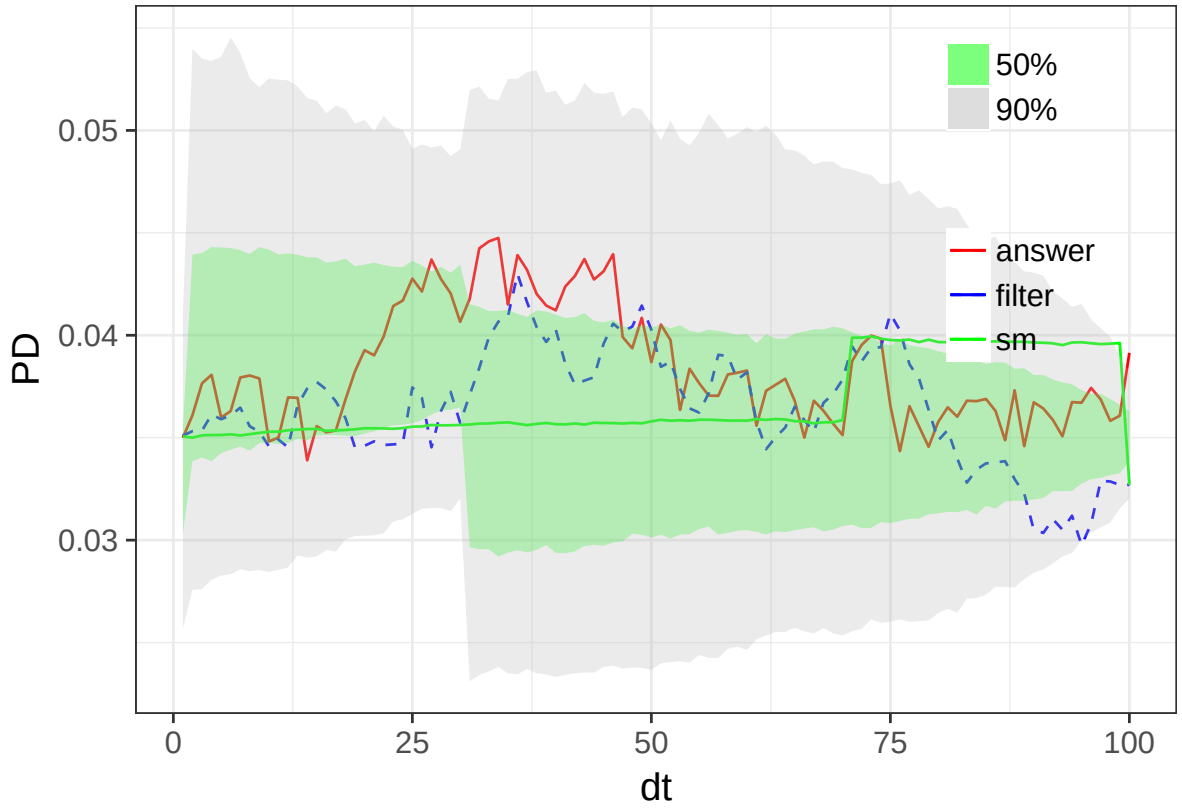
平滑化結果の図

コアの準備をして、プロット

```
cl <- makeCluster(rep('localhost', 4))
clusterExport(cl, c("dmvnorm","g_DR.fn","theta_tau","theta_mu",
                    "sm_state","theta_sigma","proposal_theta_v"))
sm_parameter <- parApply(cl,data.frame(i = length(data_for_Kalman[,1]):1),
                         1,function(i) colSums(sm_state[[i]][,c(1,2)]*
                                                 sm_state[[i]][,3]/
                                                   sum(sm_state[[i]][,3])))
stopCluster(cl)

qu<-data.frame(t(sapply(sm_state,function(x) representative_value.fn(x))))

plot_d<-data.frame(dt=c(1:length(data_for_Kalman[,1])),
                   answer=answer$rho,estimate_rho=re_rho,
                   two_fif_rho=qu$two_fif_rho,
                   sm_rho=sm_parameter[1,],
                   twenty_fif_rho=qu$twenty_fif_rho,
                   seventy_fif_rho=qu$seventy_fif_rho,
                   nine_fif=qu$nine_fif_rho)

print(ggplot(plot_d,aes(x=dt))+geom_line(aes(y=answer,colour="answer"))+
        geom_line(aes(y=estimate_rho,colour="filter"),linetype="dashed")+
        geom_line(aes(y=sm_rho,colour="sm"))+
        geom_ribbon(aes(ymin=twenty_fif_rho, ymax=seventy_fif_rho,fill="50%"),alpha = 0.3)+
        geom_ribbon(aes(ymin=two_fif_rho, ymax=nine_fif,fill="90%"),alpha = 0.3)+
        theme_bw(15)+
        theme(legend.position=c(.85,.75),legend.background=element_blank())+
        ylab(expression(rho))+
        scale_color_manual(name='', values=c("answer" = "red", "filter" = "blue","sm"="green"))+
        scale_fill_manual(name='', values=c("50%" = "green", "90%" = "gray")))
```




```
plot_d<-data.frame(dt=c(1:length(data_for_Kalman[,1])),
                   answer=answer$PD,estimate_PD=re_PD,
                   two_fif_pd=qu$two_fif_pd,
                   sm_pd=sm_parameter[2,],
                   twenty_fif_pd=qu$twenty_fif_pd,
                   seventy_fif_pd=qu$seventy_fif_pd,
                   nine_fif=qu$nine_fif_pd)

print(ggplot(plot_d,aes(x=dt))+geom_line(aes(y=answer,colour="answer"))+
      geom_line(aes(y=estimate_PD,colour="filter"),linetype="dashed")+
      geom_line(aes(y=sm_pd,colour="sm"))+
      geom_ribbon(aes(ymin=twenty_fif_pd, ymax=seventy_fif_pd,fill="50%"),alpha = 0.3)+
      geom_ribbon(aes(ymin=two_fif_pd, ymax=nine_fif,fill="90%"),alpha = 0.3)+
      theme_bw(15)+
      theme(legend.position=c(.85,.75),legend.background=element_blank())+
      ylab(expression(PD))+
      scale_color_manual(name='', values=c("answer" = "red", "filter" = "blue","sm"="green"))+
      scale_fill_manual(name='', values=c("50%" = "green", "90%" = "gray")))
```




```
Sys.time()-start_time
```

```
## Time difference of 2.407362 hours
```