Kalman Filter

Felipe Marques

Introdução

```r
library(tidyverse)
library(stochvol)
library(patchwork)
source("Kalman-filter.R")
```

Escrevi uma função de construção da verossimilhança via filtro de Kalman, baseado no artigo "A note on Stochastic Volatility models"

```r
ll_sv <- function(params, r, fuller=FALSE, c=0.005) {
  # transformando os retornos
  if (fuller){
    s2 <- var(r)
    y <- log(r^2 + c*s2) - (c*s2 / (r^2 + c*s2))
  } else {
    y <- log(r^2)
  }

  # definindo variáveis e os vetores recursivos
  n <- length(y)
  loglik_sv <- 0
  h <- P <- rep(0, n+1)
  epsilon1 <- epsilon2 <- k1 <- k2 <- rep(0, n)
  Sigma1 <- Sigma2 <- pi1 <- pi2 <- rep(0, n)

  # definindo os parâmetros
  alpha <- params[1]; phi <- params[2]; sigmaw <- params[3]
  sigma1 <- params[4]; sigma2 <- params[5]; mu1 <- params[6]
  mu2 <- params[7]
```

```r
  # definindo valores iniciais
  P[1] <- phi^2 + sigmaw^2

  for (t in 1:n) {

    epsilon1[t] <- y[t] - alpha - h[t] - mu1
    epsilon2[t] <- y[t] - alpha - h[t] - mu2
    Sigma1[t] <- P[t] + sigma1^2
    Sigma2[t] <- P[t] + sigma2^2
    k1[t] <- phi^2 * P[t] / Sigma1[t]
    k2[t] <- phi^2 * P[t] / Sigma2[t]

    pi1[t] <- (dnorm(y[t], h[t] + mu1, sqrt(Sigma1[t]))/2) /
      mean(c(dnorm(y[t], h[t] + mu1, sqrt(Sigma1[t])),
             dnorm(y[t], h[t] + mu2, sqrt(Sigma2[t]))))
    pi2[t] <- 1 - pi1[t]

    h[t+1] <- phi * h[t] + (pi1[t]*k1[t]*epsilon1[t]) +
      (pi2[t]*k2[t]*epsilon2[t])
    P[t+1] <- phi^2 * P[t] + sigmaw^2 - pi1[t]*k1[t]^2*epsilon1[t] -
      pi2[t]*k2[t]^2*epsilon2[t]

    loglik_sv <- loglik_sv + log((dnorm(y[t], h[t+1] + mu1, sqrt(Sigma1[t])) +
                                  dnorm(y[t], h[t+1] + mu1, sqrt(Sigma1[t])))/2)
  }

  return(loglik_sv)
}

estimacao_sv <- function(params, y, fuller=FALSE, c=0.005) {
  optim(params, ll_sv, r=y, fuller=fuller, c=c)
}
```

Essa é a primeira versão da função. Caso dê certo, vou tentar otimizá-la.

Estimação do modelo 1 vez

Vou simular o modelo com os seguintes parâmetros:

- $\mu = 0 \implies \beta = 1 \implies \alpha = 0$

- $\phi = 0.97$
- $\sigma_w = 1.5$

Além disso, vou considerar para a estimação:

- $\sigma_1 = \sigma_2 = 1.5$
- $\mu_1 = 0.8, \quad \mu_2 = 0.9$

como as variâncias e médias das normais que compõe a mistura de η_t.

```
amostra <- svsim(1000, mu = 0, phi = .97, sigma = 1.5)
serie <- amostra$y
volatilidade <- amostra$vol
```

```
params <- c(.3, .9, .01, 1.5, 1.5, .8, .9)
ll_sv(params, r = serie, fuller = T)
```

```
[1] 5944.3
```

nota: O algoritmo converge para valores grandes de σ_j^2. Quando as variâncias das normais que compõe a mistura são pequenas, o algoritmo pode gerar variâncias $\Sigma_{t,j}$ negativas, fazendo com que o algoritmo não convirja.

```
suppressWarnings(estimacao_sv(params, serie, fuller = F))
```

```
$par
[1] -0.4836166  0.9003032  0.9851408 -0.3211252  0.7037379  1.8585981  2.5564167

$value
[1] 1869.703

$counts
function gradient
     502       NA

$convergence
[1] 1

$message
NULL
```

nota 2: Aparentemente, é melhor não utilizar a transformação de fuller.

Estudo de simulação

Objetivo: Simular 1000 amostras do modelo com esses parâmetros e avaliar a estimativa de cada modelo simulado. Faço isso para avaliar se a função de estimação está funcionando apropriadamente.

Os parâmetros são:

- $\mu = 0 \implies \alpha = 0$
- $\phi = 0.97$
- $\sigma_w = 0.4$

Além disso, vou considerar para a estimação:

- $\sigma_1 = \sigma_2 = 1.3$
- $\mu_1 = 1, \quad \mu_2 = 1$

Vou fazer esse estudo considerando uma amostra sem transformação de fuller e outra com a transformação.

```r
# Simulação para Fuller = FALSE
n <- 1000
params <- c(1, .9, .6, 1.3, 1.3, 1, 1)
alpha <- phi <- sigmaw <- sigma1 <- sigma2 <- mu1 <- mu2 <- rep(0, n)

for (i in 1:n){
  message(paste0("Iteração: ", i, " / ", n))
  amostra <- svsim(1000, mu = 0, phi = .97, sigma = .4)$y
  estimado <- suppressWarnings(estimacao_sv(params, amostra, fuller = F))
  alpha[i] <- estimado$par[1]
  phi[i] <- estimado$par[2]
  sigmaw[i] <- estimado$par[3]
  sigma1[i] <- estimado$par[4]
  sigma2[i] <- estimado$par[5]
  mu1[i] <- estimado$par[6]
  mu2[i] <- estimado$par[7]
}
```

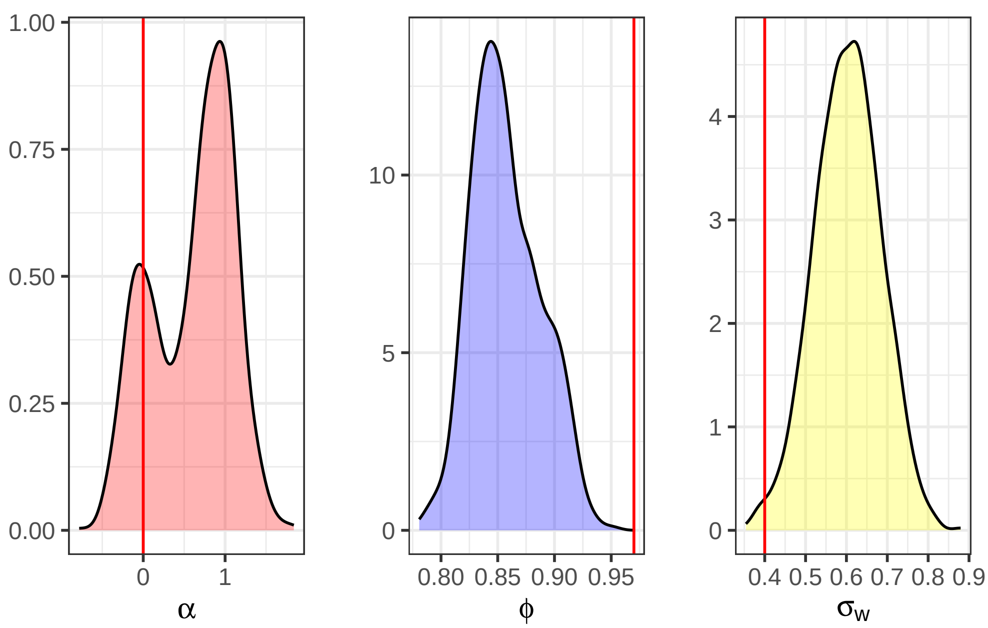
```r
# Simulação para Fuller = TRUE
n <- 1000
params <- c(1, .9, .6, 1.3, 1.3, 1, 1)
alpha_fuller <- phi_fuller <- sigmaw_fuller <- sigma1_fuller <- rep(0, n)
sigma2_fuller <- mu1_fuller <- mu2_fuller <- rep(0, n)
```

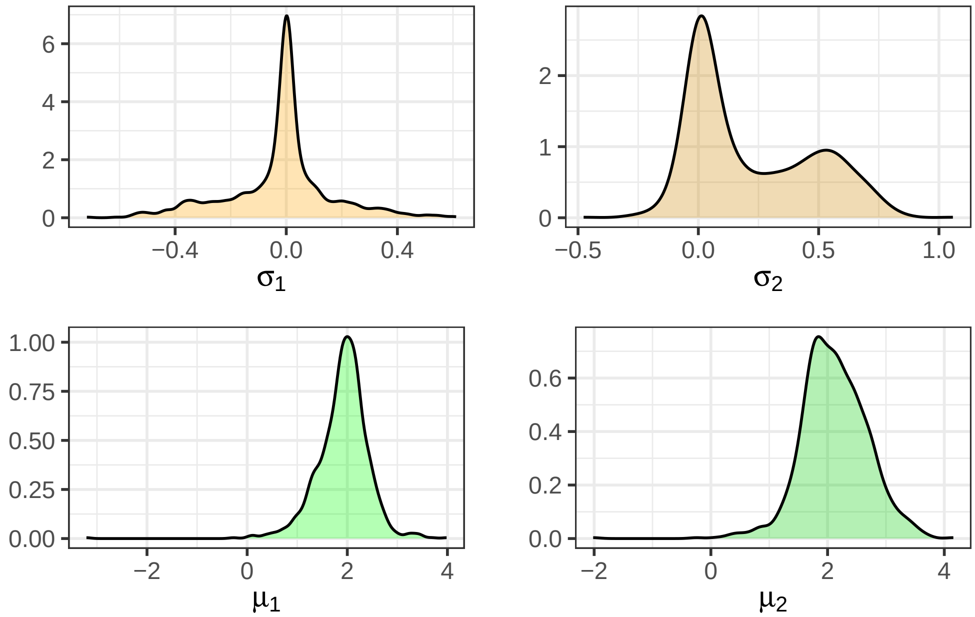
```
for (i in 1:n){
  message(paste0("Iteração: ", i, " / ", n))
  amostra <- svsim(1000, mu = 0, phi = .97, sigma = .4)$y
  estimado <- suppressWarnings(estimacao_sv(params, amostra, fuller = T))
  alpha_fuller[i] <- estimado$par[1]
  phi_fuller[i] <- estimado$par[2]
  sigmaw_fuller[i] <- estimado$par[3]
  sigma1_fuller[i] <- estimado$par[4]
  sigma2_fuller[i] <- estimado$par[5]
  mu1_fuller[i] <- estimado$par[6]
  mu2_fuller[i] <- estimado$par[7]
}
```

Análise Gráfica

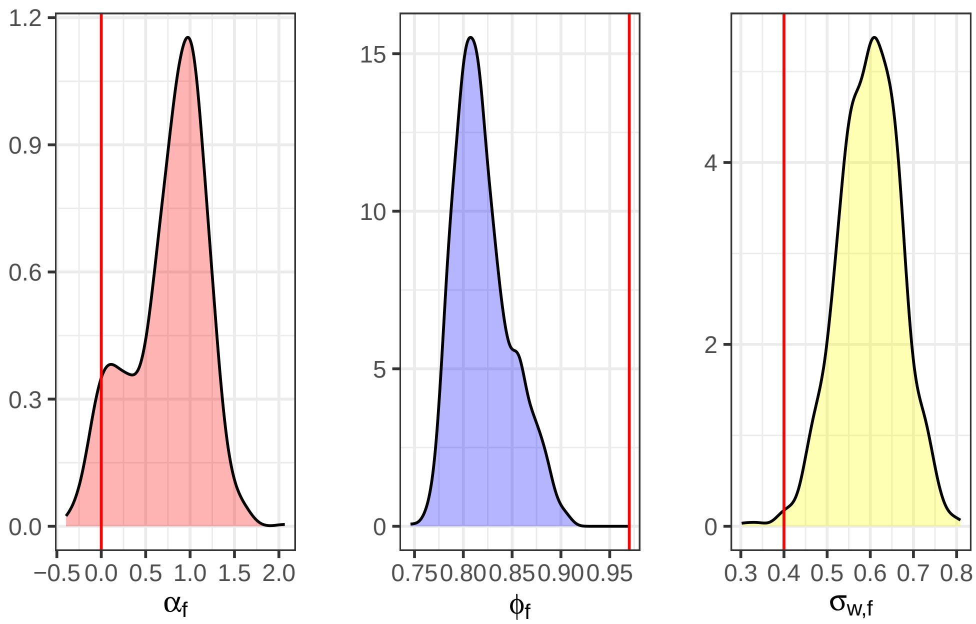
Sem usar transformação de Fuller

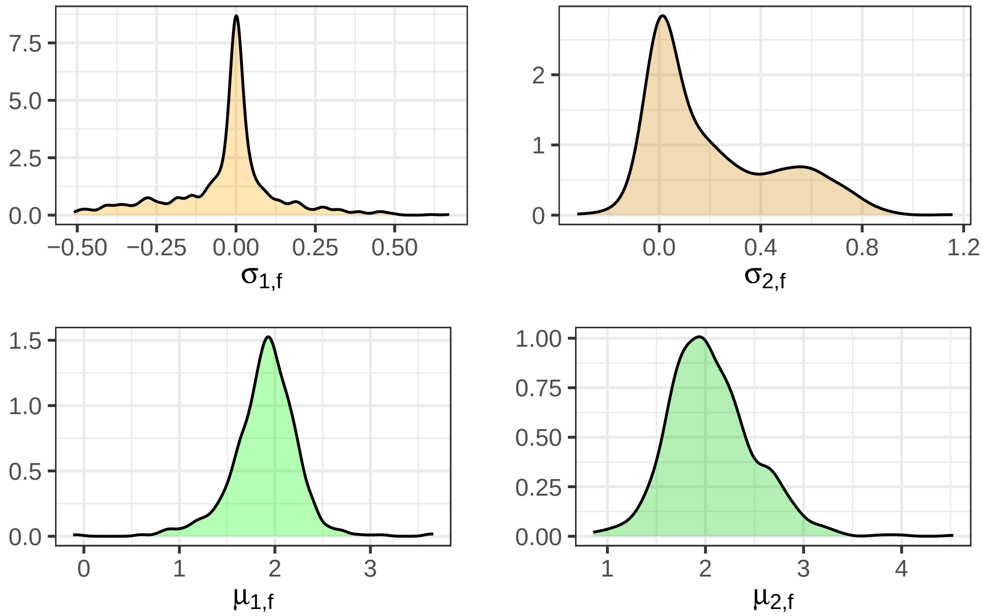
Abaixo é apresentado as densidades dos parâmetros simulados. A barra vermelha indica o valor verdadeiro dos parâmetros do modelo amostrado. Note que as densidades para os valores de $\sigma_1, \sigma_2, \mu 1, \mu 2$ não possuem a barra indicando o valor verdadeiro, já que esses parâmetros são considerados apenas na estimação, e não na simulação.





Com a transformação de Fuller





Conclusão

As simulações deixam claro que o algoritmo não está funcionando adequadamente. Possivelmente há algum erro de implementação do algoritmo. A transformação de Fuller não parece ter surtido efeito.