Modelagem

Aplicando em Modelo com Erro Gamma

Seja $X \sim Gamma(\beta, \lambda)$, então sua densidade é dada por:

$$f(x) = \frac{\lambda^\beta}{\Gamma(\beta)} x^{\beta-1} e^{-\lambda x}$$

Além disso, sua esperança é dada por $\frac{\beta}{\lambda}$ e variância $\frac{\beta}{\lambda^2}$.

```
set.seed(123)

# packages
library(fda.simu)
library(wavethresh)
library(knitr)
library(doFuture)
library(mvtnorm)

# definindo parâmetros
m <- 16                            # quantidade de pontos por curva
beta <- 196/25; lambda <- 2  # parâmetros do erro Gamma(beta, lambda)
alpha <- 0.8                       # parâmetro da mistura priori spike and slab
tau <- 5                           # parâmetro da logística presente na priori

# gerando amostra
f <- f_test(m)$bumps
e <- rgamma(m, shape=beta, rate=lambda)
y <- f + e

# DWT
d <- wd(y, filter.number=5, family='DaubExPhase')
```

```r
theta_true <- wd(f, filter.number=5, family='DaubExPhase') |>
  (\(x) c(accessC(x, lev=0), x$D))()

# RAM
theta_1 <- gera_ponto(y, lim_sup=5)  # chute inicial
post_gamma(theta_1, d, beta, lambda, tau, alpha)
```

```
[1] 1.418767e-45
```

```r
sample <- ram(theta_1, S_1=NULL, y, tau=5, alpha, beta, lambda, n_ite=20000)

sample_burnin_thinning <- sample$theta[seq(5000, 20000, 10),]

kable(data.frame('theta_i'=1:m, 'theta_true'=theta_true,
                 'RAM'=colMeans(sample_burnin_thinning)))
```

theta_i	theta_true	RAM
1	8.3245060	32.4809230
2	-0.0475269	0.2340285
3	-4.5040724	-6.2595043
4	-20.3329499	-18.9561378
5	9.0420742	7.6573027
6	-1.6569478	0.4417119
7	0.2314904	-1.0974330
8	0.8669488	0.0162902
9	0.2164796	-2.8634330
10	1.0862039	-0.9370778
11	-9.8373110	-10.9054048
12	3.2973765	3.2604467
13	-1.8987408	-0.4655492
14	3.9815792	5.4977816
15	0.8085880	1.9904579
16	-9.3267555	-8.6324696

Análise de Dados Funcionais com Erro Gama

Considere o modelo

$$A = \alpha y + e$$

Além disso, considere a amostra composta por $L = 2$ funções componentes, sendo elas a Bumps e a Doppler, medidas em $M = 16$ pontos diferentes $\{t_1, \dots, t_M\}$, e $I = 4$ observaçãoes. Então, nossa amotra consiste em $\{(t_m, A_i(t_m)),\ i = 1, \dots, 4\ ,\ m = 1, \dots, 16\}$, com erro gamma. Ademais, será denotado no código $f = \alpha y$.

Para gerar os elementos e_{ij} de e, foi utilizado $\beta = 49$ e $\lambda = 5$, então

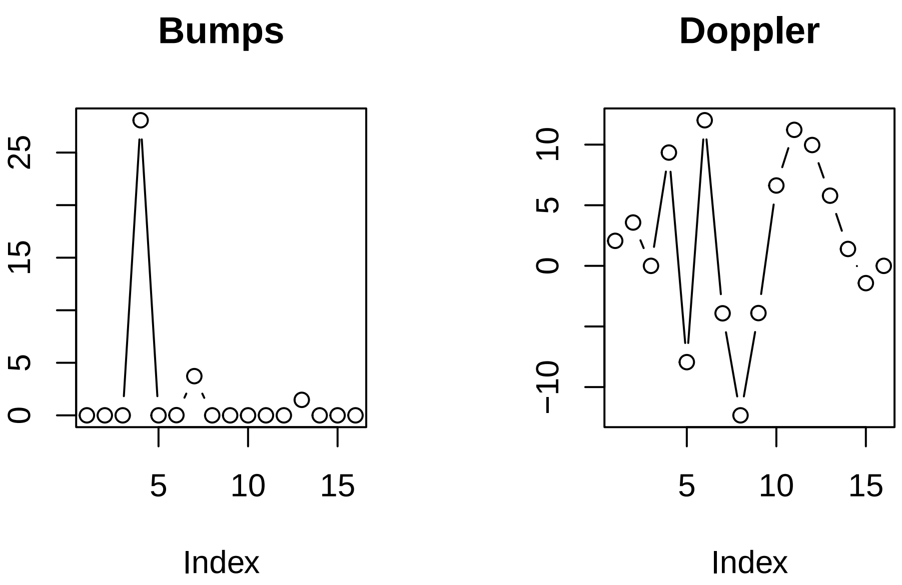
$$\mathbb{E}(e_{ij}) = \frac{\beta}{\lambda} = \frac{\frac{196}{25}}{2} = 3.92 \qquad \text{e} \qquad \mathbb{V}(e_{ij}) = \frac{\beta}{\lambda^2} = \frac{\frac{196}{25}}{2^2} = 1.96$$

assim, obtendo

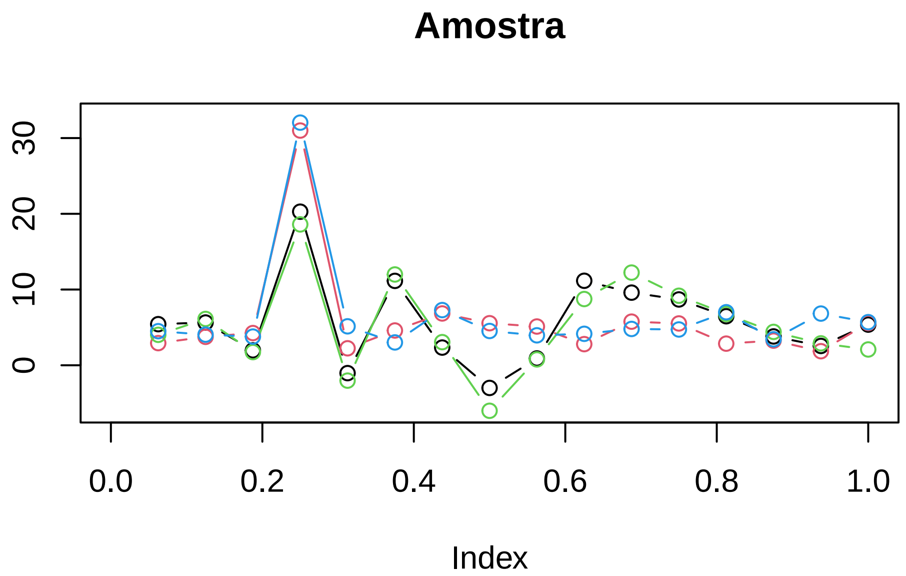
$$SNR = \frac{sd(sinal)}{sd(erro)} = \frac{7}{\sqrt{1.96}} = 5$$

Gerando amostra:

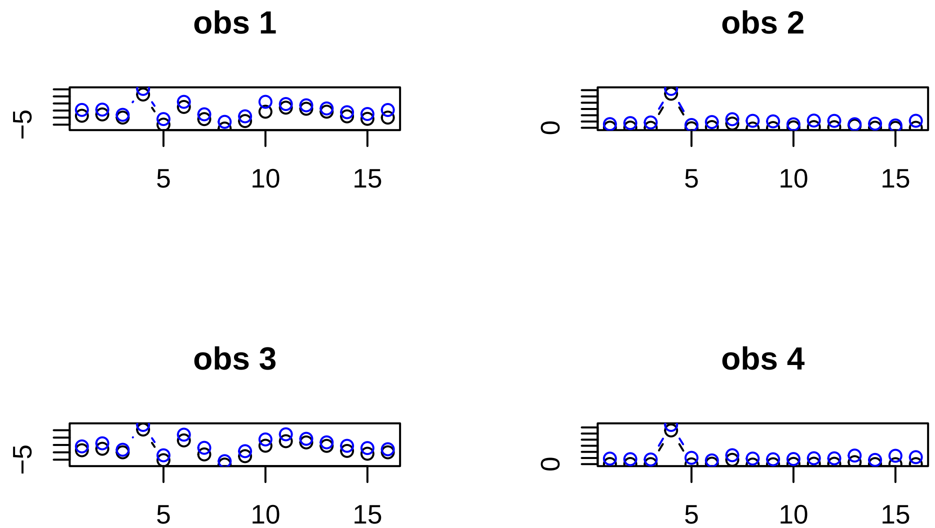
```
set.seed(282)

# gerando amostra -----------------------------------------------------
# parâmetros da amostra
M <- 16                         # quantidade de pontos por função
I <- 4                          # tamanho da amostra
beta <- 196/25; lambda <- 2 # parâmetros do erro Gamma(beta, lambda)

# funções componentes: Bumps e Doppler
fun_comp <- matrix(c(f_test(M)$bumps, f_test(M)$doppler), 2, byrow=T)
alpha_true <- t(fun_comp)  # matriz alpha com as funções componentes
colnames(alpha_true) <- c('bumps', 'doppler')

par(mfrow=c(1,2))
plot(fun_comp[1,], type='b', ylab='', main='Bumps')
plot(fun_comp[2,], type='b', ylab='', main='Doppler')
```




```
par(mfrow=c(1,1))

# pesos
y1 <- runif(I)                                  # pesos da curva 1 (bumps)
y <- matrix(c(y1, 1 - y1), nrow=2, byrow=T)     # matriz de pesos

# amostra
f <- alpha_true %*% y                           # combinação das fç's verdadeiras
e <- matrix(rgamma(M * I, shape=beta, rate=lambda), nrow=M, ncol=I, byrow=T)
A <- f + e

# plot da amostra
plot(x=(1:M)/M, y=NULL, type='n', xlim=c(0,1), ylim=c(-6, 33), ylab='', main = 'Amostra')
for (i in 1:I) lines(x=(1:M)/M, A[,i], type='b', col=i)
```




```r
par(mfrow=c(2,2))
for (i in 1:I) {  # Azul: curva observada, Preto: Curva real
  plot(x=(1:M)/M, y=NULL, ylim=c(min(f[,i]), max(A[,i])),
       ylab='', xlab='', main=paste('obs', i), type='n')
  lines(f[,i], type='b')
  lines(A[,i], type='b', col='blue')
}
```



```r
par(mfrow=c(1,1))
```

Agora, vamos levar o modelo do domínio do tempo para o domínio das ondaletas, ou seja, vamos aplicar a DWT:

$$WA = W\alpha y + We$$

obtendo o modelo

$$D = \Gamma y + \varepsilon$$

Nosso objetivo é estimar Γ através de $\hat{\Gamma} = \delta(D)y'(yy')^{-1}$, onde $\delta(D) = \begin{bmatrix}\delta(d_1) & \dots & \delta(d_I)\end{bmatrix}$ de tal forma que d_i representa a i-ésima coluna de D.

Então, levando nossa amotra para o domínio das ondaletas, temos:

```r
# DWT -----------------------------------------------------------------
D <- apply(A, MARGIN=2, \(x) {  # mesmo que W %*% A
  wd(x, filter.number=5, family='DaubExPhase')
  # c(accessC(d, lev=0), d$D)
})

Gamma_true <- apply(alpha_true, MARGIN=2, \(x) {
  d <- wd(x, filter.number=5, family='DaubExPhase')
  # c(accessC(d, lev=0), d$D)
})

Theta_true <- apply(f, MARGIN=2, \(x) {
  d <- wd(x, filter.number=5, family='DaubExPhase')
  # c(accessC(d, lev=0), d$D)
})
```

Priori Utilizada:

$$\pi(\theta) = \prod_{i=1}^{n}[\alpha\delta_0(\theta_i) + (1-\alpha)g(\theta_i;\tau)]$$

Posteriori:

$$\pi(\theta \mid d) \propto \prod_{i=1}^{n}\left[\alpha\delta_0(\theta_i) + (1-\alpha)\frac{\exp\left\{-\frac{\theta_i}{\tau}\right\}}{\tau\left(1+\exp\left\{-\frac{\theta_i}{\tau}\right\}\right)^2}\right]\exp\left\{-\lambda\sum_{i=1}^{n}\sum_{k=1}^{n}w_{ki}(d_k-\theta_k)\right\}$$
$$\times\left(\prod_{i=1}^{n}\sum_{k=1}^{n}w_{ki}(d_k-\theta_k)\right)^{\beta-1}\prod_{i=1}^{n}\mathcal{I}_{(0,\infty)}\left(\sum_{k=1}^{n}w_{ki}(d_k-\theta_k)\right)$$

Para determinar $\delta(D)$, vamos precisar definir os parâmetros da priori spike and slab (τ e α). Após isso, será necessário efetuar o algorítimo RAM e, para isso, precisa-se de um ponto dentro do domínio para iniciar o algorítimo, então será craida a função `gera_ponto_dominio`. Basicamente, precisamos que $\sum_{k=1}^{n} w_{ki}(d_k - \theta_k) > 0 \; \forall i$, ou seja, precisamos que cada elemento do vetor $a = W'(d-\theta)$ seja maior que zero, daí obtemos que $\theta = d - Wa$

```r
# parâmetros da priori spike and slab ---------------------------------
tau <- 5      # parâmetro da logística
alpha <- 0.8  # parâmetro da priori spike and slab

# função para gerar ponto no domínio ----------------------------------
#' @param a vetor com todas as entradas positivas.
#' @param d objeto da classe `wd`, o qual contém os coeficientes empíricos.
#' @param lim_sup Valor máximo para uma coordenada gerado para o ponto gerado a.
#' @inheritParams wavethresh::wd

gera_ponto_dominio <- function(a = NULL, d, lim_sup = 20,
                               filter.number = 5, family = 'DaubExPhase') {
  M <- 2^nlevelsWT(d)
  W <- t(GenW(M, filter.number, family))
  if (is.null(a)) a <- runif(M, 0.01, lim_sup)
  d_emp <- c(accessC(d, lev=0), d$D)  # coeficientes empíricos
  return(d_emp - W %*% a)  # theta
}

# testando ponto gerado
d1 <- D[[1]]
set.seed(123123)
(theta_teste <- gera_ponto_dominio(NULL, d1, 20))
```

```
              [,1]
 [1,] -18.04701325
 [2,]  -7.09627538
 [3,]  -0.95254069
 [4,] -13.91412947
 [5,]   1.94866472
 [6,]  -2.73413368
 [7,]  -7.40435208
 [8,]  -9.63433097
 [9,] -12.63095643
[10,]   0.06226093
[11,]  -2.87345232
```

```
[12,]  14.67241833
[13,]  -8.37654396
[14,]  -3.07879060
[15,]  -0.25772383
[16,]  -0.03553847
```

```r
post_gamma(theta_teste, d1, beta, lambda, tau, alpha, filter.number=5,
           family='DaubExPhase')
```

```
[1] 4.314074e-76
```

Aplicando $\delta(D) = \begin{bmatrix}\delta(d_1) & \ldots & \delta(d_I)\end{bmatrix}$, contudo, como a posteriori não tem forma fechada vamos ter que usar o algorítimo RAM para amostrar da distribuição. Então vamos fazer a função `ram_manual`

```r
# implementando ram_manual ------------------------------------------
#' @param theta_1 chute inicial para theta.
#' @param S_1 chute inicial para S.
#' @param d coeficientes empíricos de ondaleta.
#' @param n_ite número de iterações.
#' @param alpha coeficiente de mistura da priori spike and slab.
#' @param tau coeficiente da distribuição logística da priori.
#' @param beta,lambda parâmetros da distribuição do erro
#' \eqn{Gamma(\beta, \lambda)}.
#' @param gamma taxa alvo de aceitação (confirmar)

ram_manual <- function(theta_1, S_1, d, n_ite, alpha=0.8, tau=2, beta, lambda,
                       gamma = 2/3, filter.number=5, family='DaubExPhase') {
  # verificando ponto inicial
  if (post_gamma(theta_1, d, beta, tau, lambda, alpha) == 0)
    stop('Ponto inicial inválido, forneça um ponto com densidade maior que 0.')

  M <- length(theta_1)  # quantidade de pontos por função

  # criando objetos
  theta <- matrix(0, nrow=n_ite, ncol=M)    # matriz contendo amostra de theta
  eta <- seq(1, n_ite)^(-gamma)             # parâmetro do algorítimo RAM
  S_l <- vector(mode='list', length=n_ite)  # lista para armazenar S_l
  gamma_l <- vector(mode='numeric', n_ite - 1)  # vetor para armazenar gamma_l

  # chutes iniciais
```

```r
  theta[1,] <- theta_1
  S_l[[1]] <- S_1

  for (i in 2:n_ite) {
    # proposta
    U_l <- t(rmvnorm(1, rep(0, M), diag(M)))
    theta_star <- t(theta[i-1,] + S_l[[i-1]] %*% U_l)

    # taxa de aceitação
    gamma_l[i-1] <- min(1, post_gamma(theta_star, d, beta, tau, lambda, alpha)/
                          post_gamma(theta[i-1,], d, beta, tau, lambda, alpha))

    if (rbinom(1, 1, gamma_l[i-1]) == 1) theta[i,] <- theta_star
    else theta[i,] <- theta[i-1,]

    # atualizando S_l
    A <- S_l[[i-1]] %*% (diag(M) + eta[i]*(gamma_l[i-1] - gamma) *
                           U_l %*% t(U_l) / as.vector(t(U_l) %*% U_l)) %*% t(S_l[[i-1]])
    S_l[[i]] <- t(chol(A))
  }

  return(list('theta'=theta, 'S'=S_l, 'gamma_l'=gamma_l,
              'parametros'=c('n_ite'=n_ite, 'alpha'=alpha, 'tau'=tau, 'beta'=beta,
                             'lambda'=lambda, 'gamma'=gamma,
                             'filter.number'=filter.number, 'family'=family)
  ))
}

# Determinando delta(D) -----------------------------------------------
# parâmetros da simulação
n_ite <- 50000  # número de iterações
plan(multisession, workers = 4)  # cada núcleo faz uma obs para I = 4

delta_D <- foreach(i = 1:I, .options.future = list(seed = TRUE),
                   .combine=cbind) %dofuture% {
  d <- D[[i]]
  for (j in 1:20) {
    theta_1 <- gera_ponto_dominio(a = NULL, d, lim_sup = 10)
    if (post_gamma(theta_1, d, beta, tau, lambda, alpha) != Inf) break
    if (j == 20) stop('nenhum valor valido encontrado para chute incial de theta.')
  }
  sample <- ram_manual(theta_1, S_1=diag(M), d, n_ite, alpha, tau, beta,
```

```
                        lambda, gamma = 2/3, filter.number=5, family='DaubExPhase')
  # efetuando burn-in e thinning
  colMeans(sample$theta[seq(100, n_ite, 10),])
}

# Recuperando Gamma ----------------------------------------------------
Gamma_hat <- delta_D %*% t(y) %*% solve(y %*% t(y))  # delta(D)y'(yy')^-1

# levando para o domínio do tempo
alpha_hat <- GenW(M, filter.number=5, family='DaubExPhase') %*% Gamma_hat
colnames(alpha_hat) <- c('bumps_est', 'doppler_est')

# Resultados -----------------------------------------------------------
kable(data.frame('theta_i'=1:m, 'bumps'=alpha_true[,1], 'bumps_est'=alpha_hat[,1],
                 'doppler'=alpha_true[,2], 'doppler_est'=alpha_hat[,2]))
```

theta_i	bumps	bumps_est	doppler	doppler_est
1	0.0000000	2.177508	2.062911	3.6445667
2	0.0000000	2.141354	3.575490	5.2909160
3	0.0000000	2.638634	0.000000	-0.9392643
4	28.0749879	30.229508	9.344035	10.8701091
5	0.0000000	2.323554	-7.942610	-6.0199564
6	0.0181968	1.733852	12.011638	14.2194243
7	3.7309027	5.709544	-3.914332	-1.2394682
8	0.0000000	4.075654	-12.331902	-11.3495836
9	0.0000000	3.218261	-3.895052	-2.8074594
10	0.0000000	1.590373	6.628780	11.7399958
11	0.0000000	3.257337	11.219565	12.5311336
12	0.0000000	3.294357	9.968256	9.3993001
13	1.4739369	3.251602	5.787962	5.7951764
14	0.0000000	1.738838	1.392679	2.9356829
15	0.0000000	2.899613	-1.425523	0.1951217
16	0.0000000	4.252004	0.000000	0.9689418

```
par(mfrow=c(1,2))
plot(alpha_true[,1], type='b', main='Bumps', ylab='',
     ylim=c(min(alpha_true[,1]), max(alpha_hat[,1])))
lines(alpha_hat[,1], type='b', col='blue')
legend('topright', legend=c('Curva real', 'Cruva recuperada'),
       col=c('black', 'blue'), bty='n', lwd=1, cex=0.6)
```

```
plot(alpha_true[,2], type='b', main='Bumps', ylim=c(min(alpha_true[,2]), max(alpha_hat[,2])))
lines(alpha_hat[,2], type='b', col='blue')
legend('bottomright', legend=c('Curva real', 'Cruva recuperada'),
       col=c('black', 'blue'), bty='n', lwd=1, cex=0.6)
```