Inhomogeneous HMMs

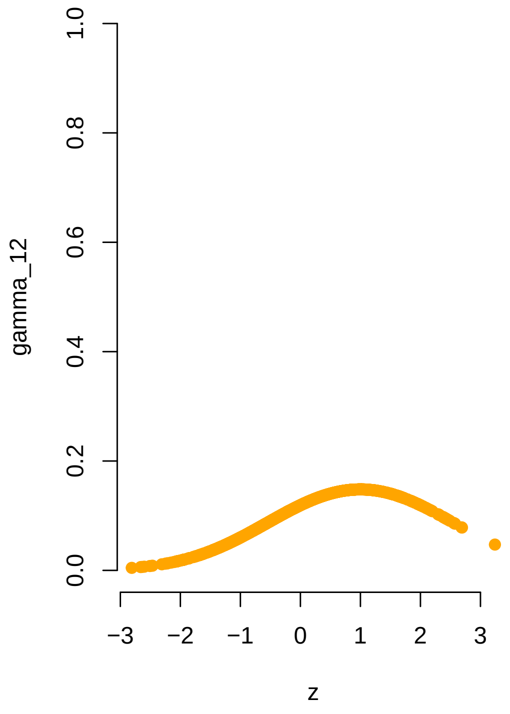
Jan-Ole Koslik

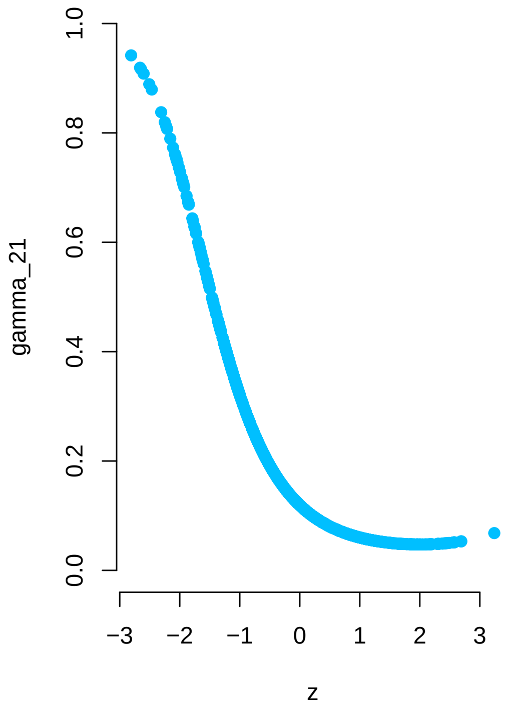
This vignette shows how to fit inhomogeneous HMMs. Inhomogeneity in HMMs can be in the form of covariates affecting the transition probabilities of the underlying Markov chain, or covariates affecting the state-dependent distributions, which would then be called Markov-switching regression. We will begin with effects in the state process

Covariate effects in the state process

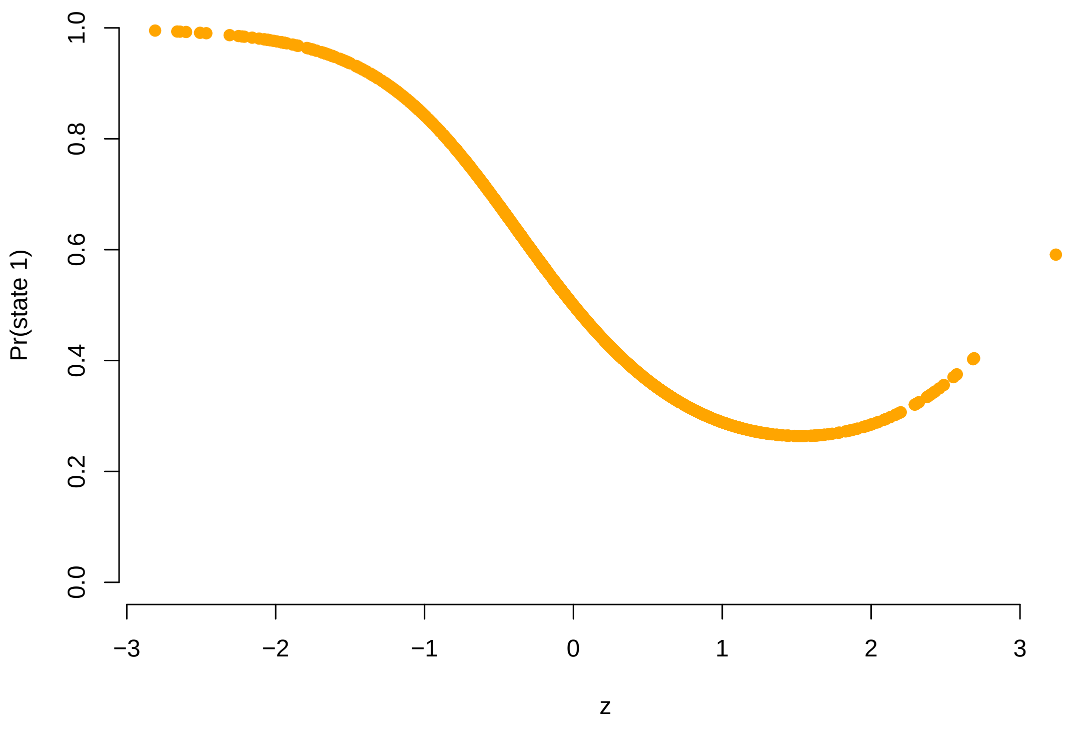
```r
# parameters
mu = c(5, 20)
sigma = c(4, 5)

beta = matrix(c(-2, -2, # intercepts
                -1, 0.5,
                0.25, -0.25), # covariate effects
              nrow = 2)

n = 1000
set.seed(123)
z = rnorm(n) # in practice there will be n covariate values.
# However, we only have n-1 transitions, thererforce we only need n-1 values:
Z = cbind(z, z^2) # quadratic effect of z
Gamma = tpm_g(Z = Z[-1,], beta) # of dimension c(2, 2, n-1)
delta = c(0.5, 0.5) # non-stationary initial distribution

color = c("orange", "deepskyblue")

par(mfrow = c(1,2))
plot(z[-1], Gamma[1,2,], pch = 19, bty = "n", ylim = c(0,1),
     xlab = "z", ylab = "gamma_12", col = color[1])
plot(z[-1], Gamma[2,1,], pch = 19, bty = "n", ylim = c(0,1),
     xlab = "z", ylab = "gamma_21", col = color[2])
```

Setting parameters for simulation



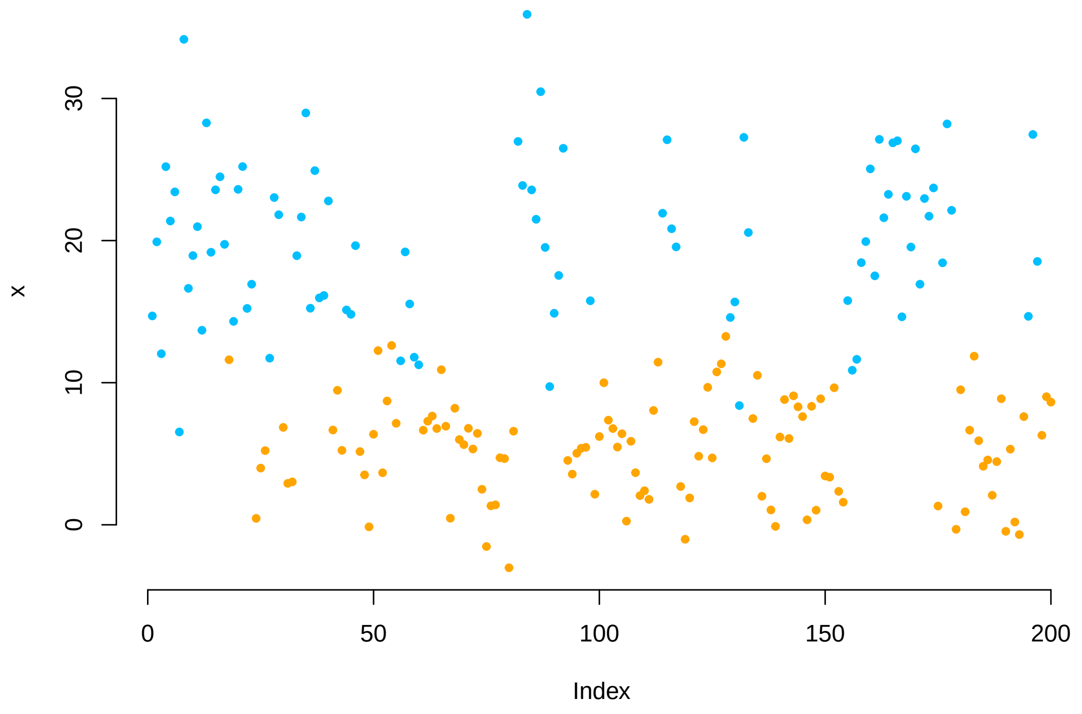


```r
Delta = matrix(nrow = n-1, ncol = 2)
for(i in 1:(n-1)){ Delta[i,] = stationary(Gamma[,,i]) }

par(mfrow = c(1,1))
plot(z[-1], Delta[,1], pch = 19, bty = "n", ylim = c(0,1), xlab = "z",
     ylab = "Pr(state 1)", col = color[1])
```



```
s = x = rep(NA, n)
s[1] = sample(1:2, 1, prob = delta)
x[1] = stats::rnorm(1, mu[s[1]], sigma[s[1]])
for(t in 2:n){
  s[t] = sample(1:2, 1, prob = Gamma[s[t-1],,t-1])
  x[t] = stats::rnorm(1, mu[s[t]], sigma[s[t]])
}

plot(x[1:200], bty = "n", pch = 20, ylab = "x",
     col = c(color[1], color[2])[s[1:200]])
```

Simulating data



Parametric modeling of the transition probabilities

Writing the negative log-likelihood function Here we specify the likelihood function and pretend we know the polynomial degree of the effect of z on the transition probabilities.

```
mllk = function(theta.star, x, Z){
  beta = matrix(theta.star[1:6], nrow = 2) # matrix of coefficients
  Gamma = tpm_g(Z[-1,], beta) # excluding the first covariate value -> n-1 tpms
  delta = c(1, exp(theta.star[7]))
  delta = delta / sum(delta)
  mu = theta.star[8:9]
  sigma = exp(theta.star[10:11])
  # calculate all state-dependent probabilities
  allprobs = matrix(1, length(x), 2)
  for(j in 1:2){ allprobs[,j] = stats::dnorm(x, mu[j], sigma[j]) }
```

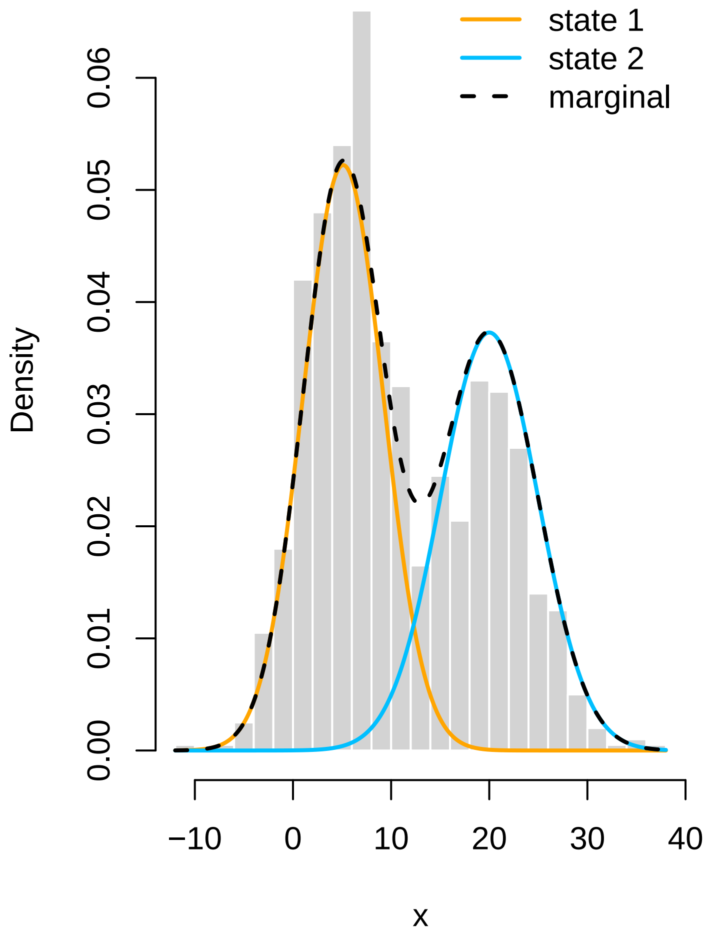
```r
  # return negative for minimization
  -forward_g(delta, Gamma, allprobs)
}
```

```r
theta.star = c(-2, -2, rep(0,4), # initializing with homogeneous tpm
               0, # starting value for initial distribution
               4, 14 ,log(3),log(5)) # starting values state-dependent process
t1 = Sys.time()
mod = stats::nlm(mllk, theta.star, x = x, Z = Z)
Sys.time()-t1
#> Time difference of 0.2223959 secs
```

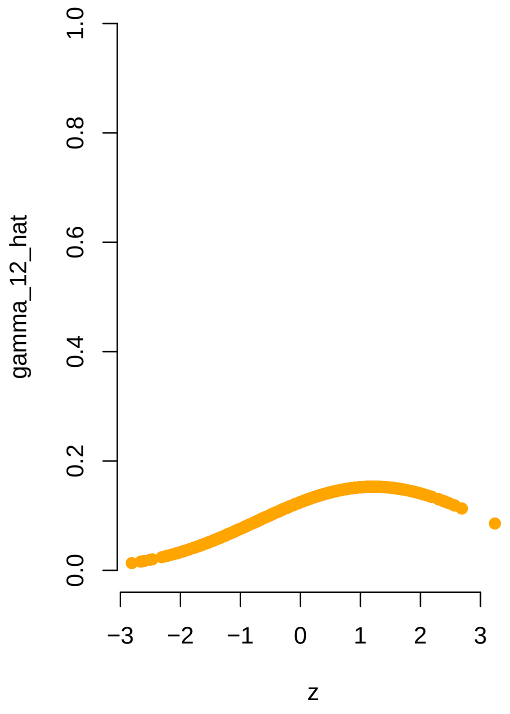
Fitting an HMM to the data Really fast!

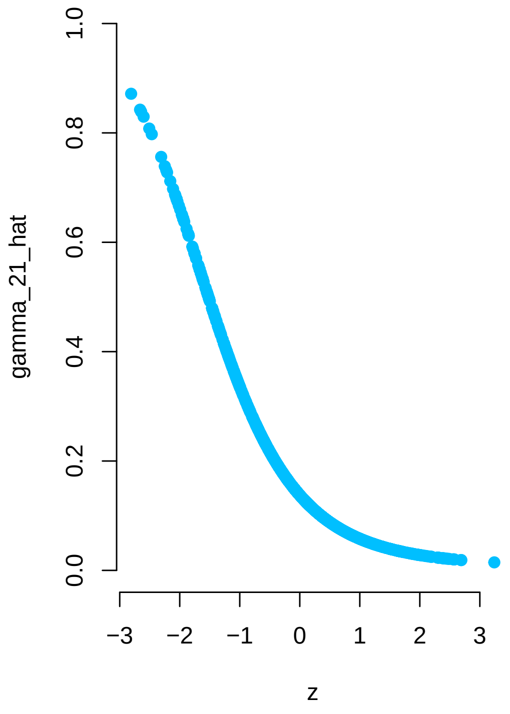
Visualizing results Again, we use `tpm_g()` and `stationary()` to tranform the parameters.

```r
# transform parameters to working
beta_hat = matrix(mod$estimate[1:6], nrow = 2)
Gamma_hat = tpm_g(Z = Z[-1,], beta_hat)
delta_hat = c(1, exp(mod$estimate[7]))
delta_hat = delta_hat / sum(delta_hat)
mu_hat = mod$estimate[8:9]
sigma_hat = exp(mod$estimate[10:11])

# we calculate the average state distribution overall all covariate values
Prob = matrix(nrow = n-1, ncol = 2)
for(i in 1:(n-1)){ Prob[i,] = stationary(Gamma_hat[,,i]) }
prob = apply(Prob, 2, mean)

par(mfrow = c(1,2))
hist(x, prob = TRUE, bor = "white", breaks = 20, main = "")
curve(prob[1]*dnorm(x, mu_hat[1], sigma_hat[1]), add = TRUE, lwd = 2,
      col = color[1], n=500)
curve(prob[2]*dnorm(x, mu_hat[2], sigma_hat[2]), add = TRUE, lwd = 2,
      col = color[2], n=500)
curve(prob[1]*dnorm(x, mu_hat[1], sigma_hat[1])+
        prob[2]*dnorm(x, mu[2], sigma_hat[2]),
      add = TRUE, lwd = 2, lty = "dashed", n = 500)
legend("topright", col = c(color[1], color[2], "black"), lwd = 2, bty = "n",
       lty = c(1,1,2), legend = c("state 1", "state 2", "marginal"))
par(mfrow = c(1,2))
```

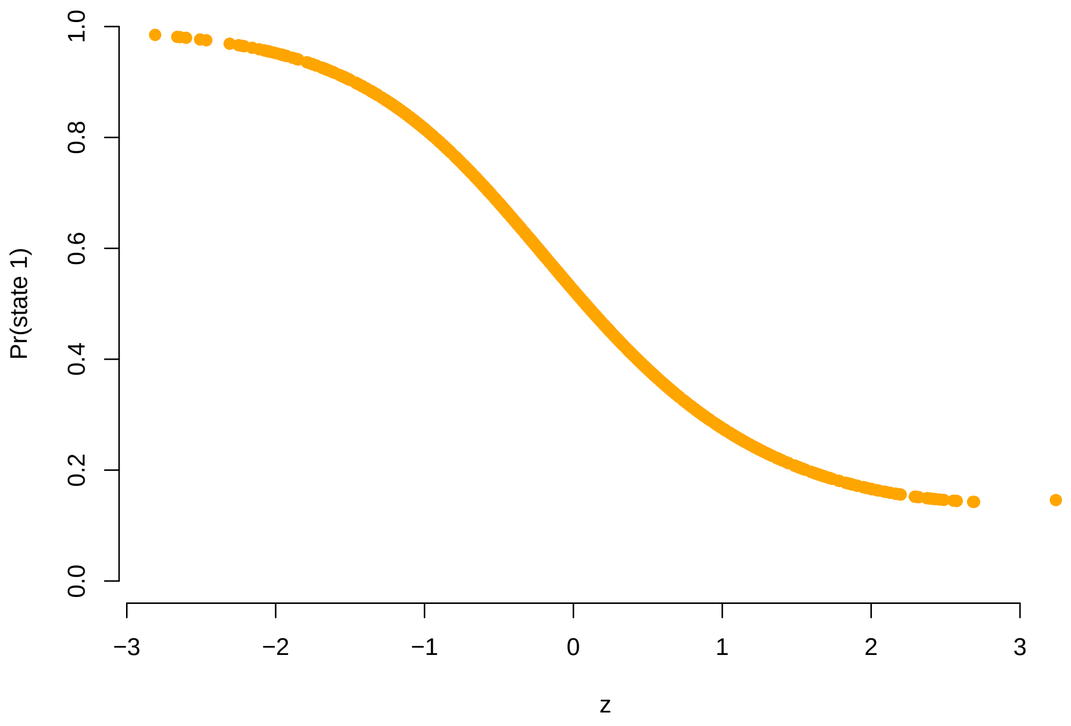


```
plot(z[-1], Gamma_hat[1,2,], pch = 19, bty = "n", ylim = c(0,1),
     xlab = "z", ylab = "gamma_12_hat", col = color[1])
plot(z[-1], Gamma_hat[2,1,], pch = 19, bty = "n", ylim = c(0,1),
     xlab = "z", ylab = "gamma_21_hat", col = color[2])
```





```
par(mfrow = c(1,1))
plot(z[-1], Prob[,1], pch = 19, bty = "n", ylim = c(0,1), xlab = "z",
     ylab = "Pr(state 1)", col = color[1])
```

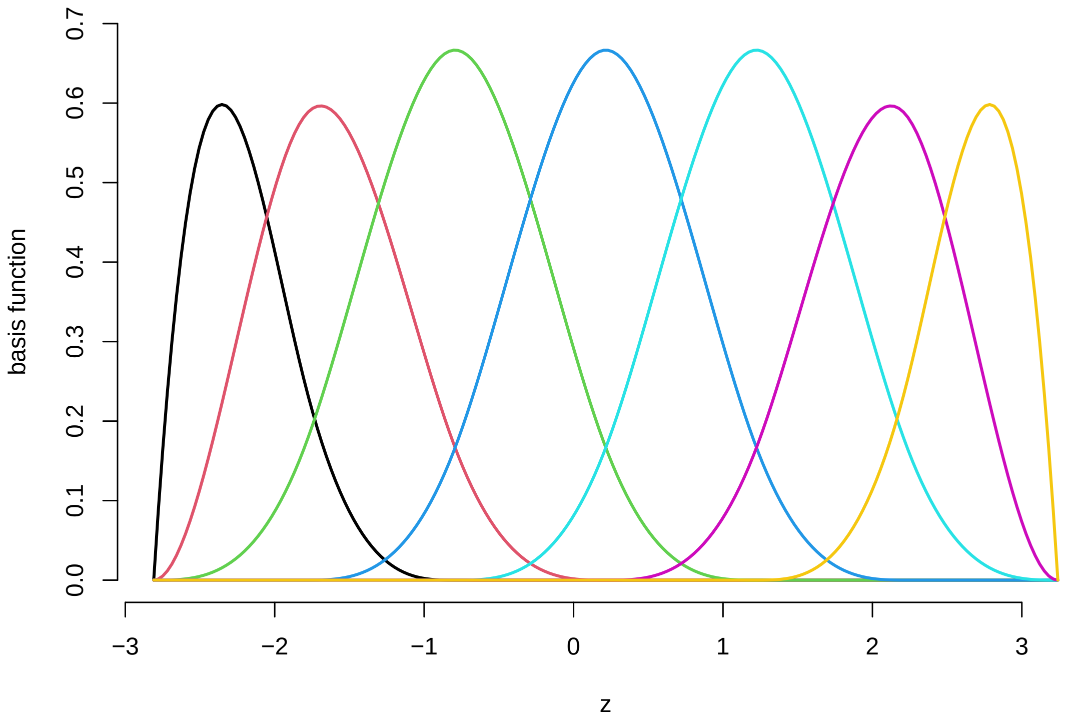


Non-parametric modeling of the transition probalities

In practice, of course we do not know the exact form of the relationship between z and the transition probabilities. Therefore, `Lcpp` also makes non-parametric modeling trivially easy. Here we model the transition probabilities using P-splines. We do so in first calculating the design matrix using the `splines` package which we can easily be handled by `tpm_g()`.

```r
Z = splines::bs(x = z, df = 6) ## B-spline design matrix

# visualizing the splines
zseq = seq(min(z), max(z), length = 200)
Zplot = splines::bs(x = zseq, df = 8)

plot(zseq, Zplot[,1], type = "l", lwd = 2, bty = "n",
     xlim = c(zseq[1], zseq[200]), ylim = c(0,0.7), xlab = "z", ylab = "basis function")
for(i in 2:(ncol(Zplot)-1)){
  lines(zseq, Zplot[,i], lwd = 2, col = i)
}
```

Building the B-spline design matrix



Writing the negative log-likelihood function We only need to make small changes to the likelihood function. In general, a penalty for the curvature should also be added, which is done in the last lines.

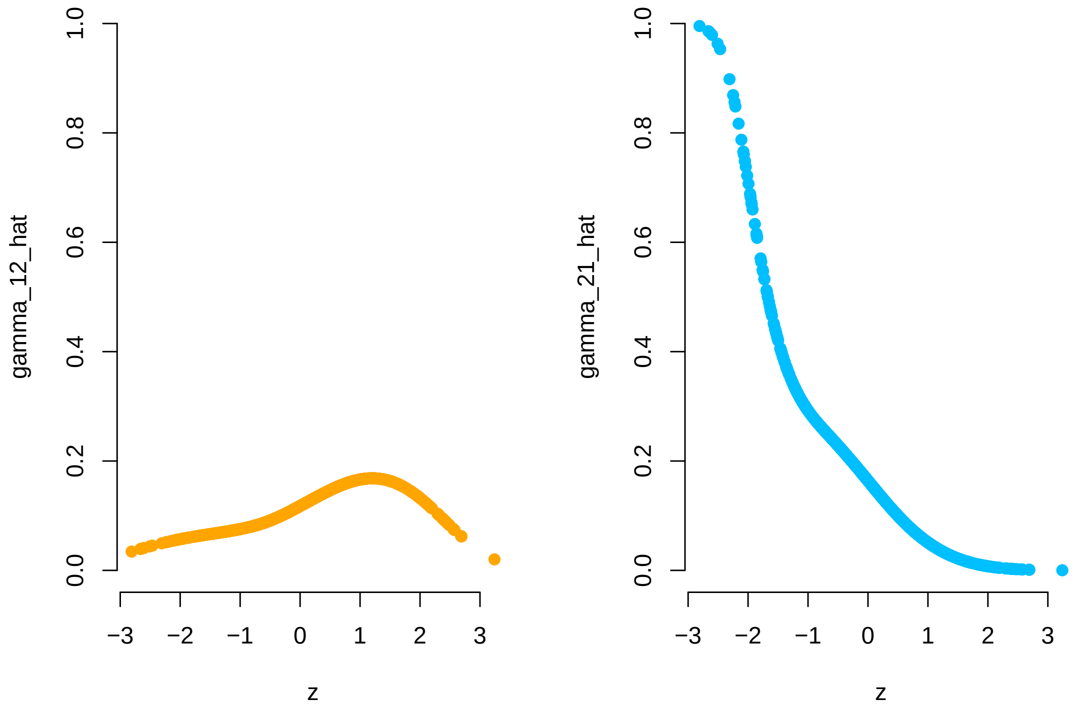
```r
mllk_np = function(theta.star, x, Z, lambda){
  beta = matrix(theta.star[1:(2+2*ncol(Z))], nrow = 2)
  Gamma = tpm_g(Z = Z[-1,], beta = beta) # calculating all tpms
  delta = c(1, exp(theta.star[2+2*ncol(Z)+1]))
  delta = delta / sum(delta)
  mu = theta.star[2+2*ncol(Z)+1+1:2]
  sigma = exp(theta.star[2+2*ncol(Z)+3+1:2])
  # calculate all state-dependent probabilities
  allprobs = matrix(1, length(x), 2)
  for(j in 1:2){ allprobs[,j] = stats::dnorm(x, mu[j], sigma[j]) }
  # return negative for minimization
  l = forward_g(delta, Gamma, allprobs)
  # penalize curvature
  penalty = sum(diff(beta[1,-1], differences = 4)^2)+
    sum(diff(beta[2,-1], differences = 4)^2)
  return(-l + lambda*penalty)
}
```

```r
theta.star = c(-2,-2, rep(0, 2*ncol(Z)), # starting values state process
               0, # starting value initial distribution
               4, 14 ,log(3),log(5)) # starting values state-dependent process
t1 = Sys.time()
mod_np = stats::nlm(mllk_np, theta.star, x = x, Z = Z, lambda = 50)
# in this case we don't seem to need a lot of penalization
Sys.time()-t1
#> Time difference of 0.6083891 secs
```

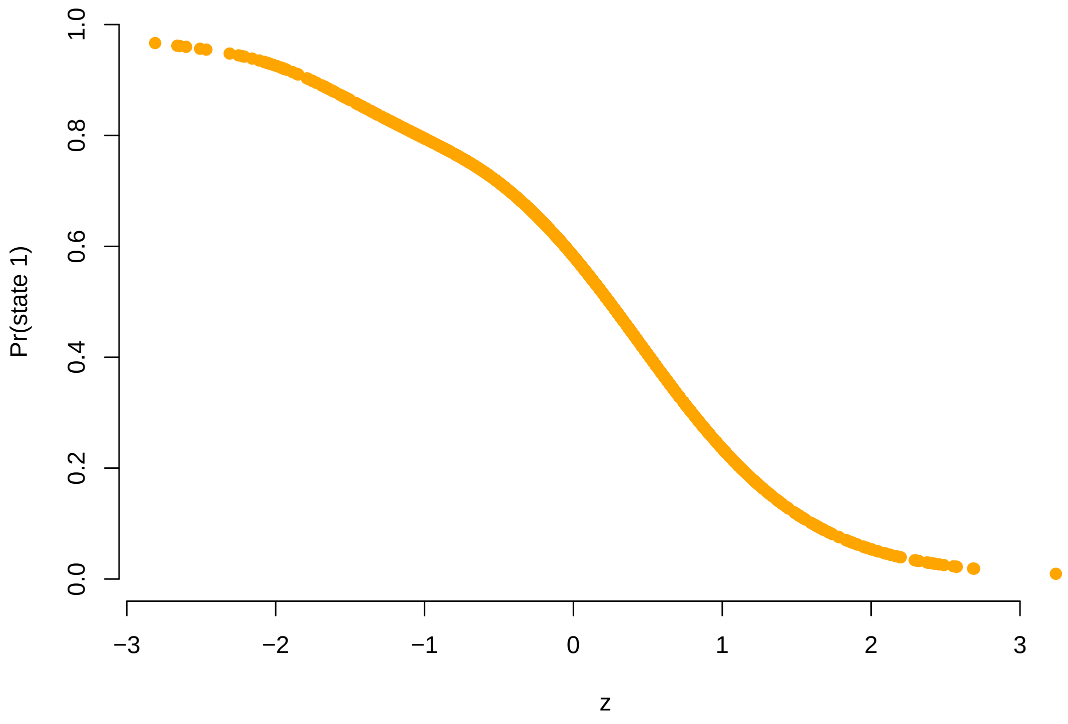
Fitting a non-parametric HMM The model fit is still quite fast for non-parametric modeling.

Visualizing results Again, we use `tpm_g()` and `stationary()` to tranform the unconstraint parameters to working parameters.

```r
# transform parameters to working
beta_hat_np = matrix(mod_np$estimate[1:(2+2*ncol(Z))], nrow = 2)
Gamma_hat_np = tpm_g(Z = Z[-1,], beta = beta_hat_np)

# we calculate the average state distribution overall all covariate values
Prob_np = matrix(nrow = n-1, ncol = 2)
for(i in 1:(n-1)){ Prob_np[i,] = stationary(Gamma_hat_np[,,i]) }

# visualizing the Spline fit
par(mfrow = c(1,2))
plot(z[-1], Gamma_hat_np[1,2,], pch = 19, bty = "n", ylim = c(0,1),
     xlab = "z", ylab = "gamma_12_hat", col = color[1])
plot(z[-1], Gamma_hat_np[2,1,], pch = 19, bty = "n", ylim = c(0,1),
     xlab = "z", ylab = "gamma_21_hat", col = color[2])
```



```
par(mfrow = c(1,1))
plot(z[-1], Prob_np[,1], pch = 19, bty = "n", ylim = c(0,1), xlab = "z",
     ylab = "Pr(state 1)", col = color[1])
```



Covariate effects in the state-dependent process

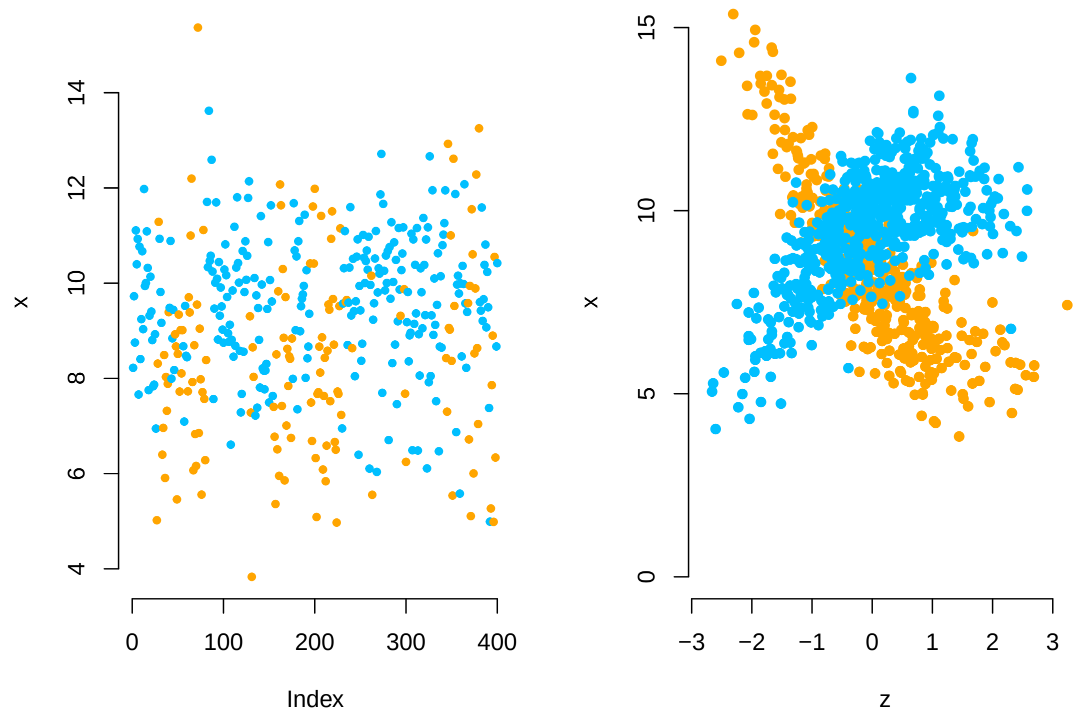
We now look at a setting, where covariates influence the mean of the state-dependent distribution, while the state switching is controlled by a homogeneous Markov chain. This is often called Markov-switching regression.

```r
sigma = c(1, 1)
# each row is now the vector of state-dependent regression parameters
beta = matrix(c(8, 10, # intercepts
                -2, 1, 0.5, -0.5), # covariate effects
              nrow = 2)
n = 1000
set.seed(123)
z = rnorm(n)
Z = cbind(z, z^2) # quadratic effect of z

Gamma = matrix(c(0.9, 0.1, 0.05, 0.95), nrow = 2, byrow = TRUE) # homogeneous
delta = stationary(Gamma) # stationary Markov chain
```

Setting parameters for simulation

```r
s = x = rep(NA, n)
s[1] = sample(1:2, 1, prob = delta)
x[1] = stats::rnorm(1, beta[s[1],]%*%c(1, Z[1,]), # state-dependent regression
                    sigma[s[1]])
for(t in 2:n){
  s[t] = sample(1:2, 1, prob = Gamma[s[t-1],])
  x[t] = stats::rnorm(1, beta[s[t],]%*%c(1, Z[t,]), # state-dependent regression
                      sigma[s[t]])
}

par(mfrow = c(1,2))
plot(x[1:400], bty = "n", pch = 20, ylab = "x",
     col = c(color[1], color[2])[s[1:400]])

plot(z[which(s==1)], x[which(s==1)], pch = 16, col = color[1], bty = "n",
     ylim = c(0,15), xlab = "z", ylab = "x")
points(z[which(s==2)], x[which(s==2)], pch = 16, col = color[2])
```

Simulation



Parametric modeling of the state-dependent regressions

```
mllk_reg = function(theta.star, x, Z){
  Gamma = tpm(theta.star[1:2]) # homogeneous tpm
  delta = stationary(Gamma) # stationary Markov chain
  beta = matrix(theta.star[2+1:(2+2*2)], nrow = 2)
  sigma = exp(theta.star[2+2+2*2 +1:2])
  # calculate all state-dependent probabilities
  allprobs = matrix(1, length(x), 2)
  # state-dependent regression
  for(j in 1:2){ allprobs[,j] = stats::dnorm(x, cbind(1,Z)%*%beta[j,], sigma[j]) }
  # return negative for minimization
  -forward(delta, Gamma, allprobs)
}
```

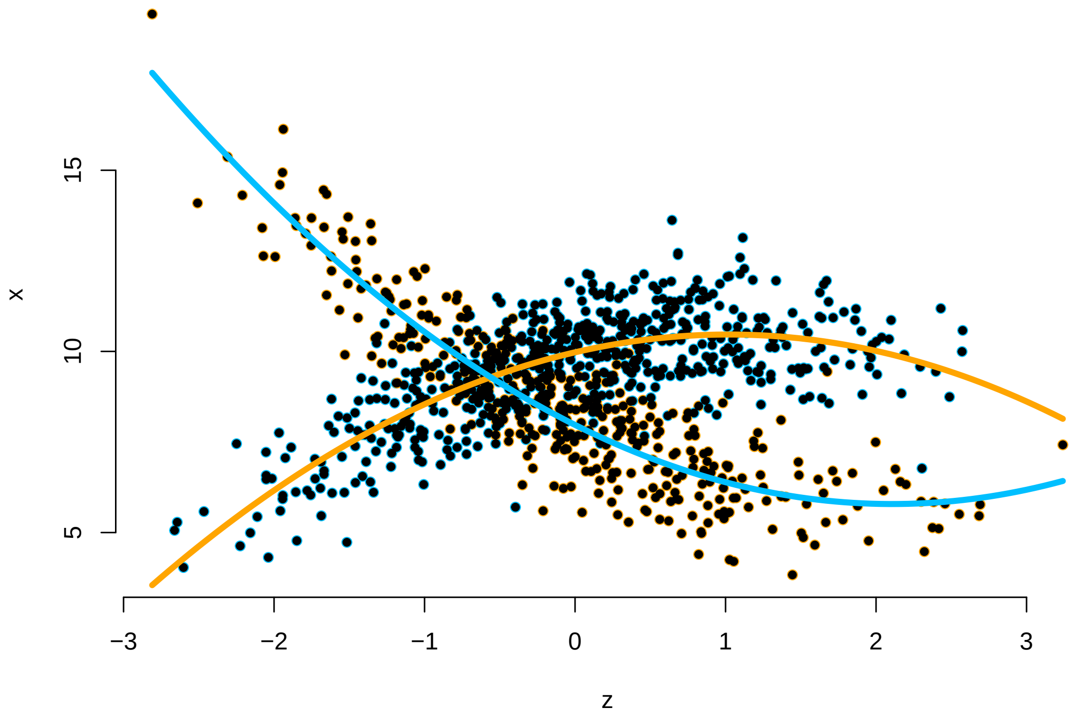
Writing the negative log-likelihood function

```
theta.star = c(-2,-3, # starting values state process
               8, 10, rep(0,4), # starting values for regression
               log(1),log(1)) # starting values for sigma
t1 = Sys.time()
mod_reg = stats::nlm(mllk_reg, theta.star, x = x, Z = Z)
Sys.time()-t1
#> Time difference of 0.04106903 secs
```

Fitting a Markov-switching regression model

```
Gamma_hat_reg = tpm(mod_reg$estimate[1:2]) # calculating all tpms
delta_hat_reg = stationary(Gamma_hat_reg)
beta_hat_reg = matrix(mod_reg$estimate[2+1:(2*2+2)], nrow = 2)
sigma_hat_reg = exp(mod_reg$estimate[2+2*2+2 +1:2])

plot(z, x, pch = 16, bty = "n", xlab = "z", ylab = "x", col = color[s])
points(z, x, pch = 20)
curve(beta_hat_reg[1,1] + beta_hat_reg[1,2]*x + beta_hat_reg[1,3]*x^2,
      add = T, lwd = 4, col = color[1])
curve(beta_hat_reg[2,1] + beta_hat_reg[2,2]*x + beta_hat_reg[2,3]*x^2,
      add = T, lwd = 4, col = color[2])
```

Visualizing results



Non-parametric modeling of the state-dependent regressions

This is now a trivial task, just combininig the previous two examples.

```
Z = splines::bs(x = z, df = 6) ## B-spline design matrix
```

Again building the B-spline design matrix

```r
mllk_npreg = function(theta.star, x, Z, lambda){
  Gamma = tpm(theta.star[1:2]) # homogeneous tpm
  delta = stationary(Gamma) # stationary Markov chain
  beta = matrix(theta.star[2+1:(2+2*ncol(Z))], nrow = 2)
  sigma = exp(theta.star[2+2+2*ncol(Z) + 1:2])
  # calculate all state-dependent probabilities
  allprobs = matrix(1, length(x), 2)
  # state-dependent regression
  for(j in 1:2){ allprobs[,j] = stats::dnorm(x, cbind(1,Z)%*%beta[j,], sigma[j]) }
  # return negative for minimization
  l = forward(delta, Gamma, allprobs)
  # penalize curvature
  penalty = sum(diff(beta[1,-1], differences = 3)^2)+
    sum(diff(beta[2,-1], differences = 3)^2)
  return(-l + lambda*penalty)
}
```

Writing the negative log-likelihood function

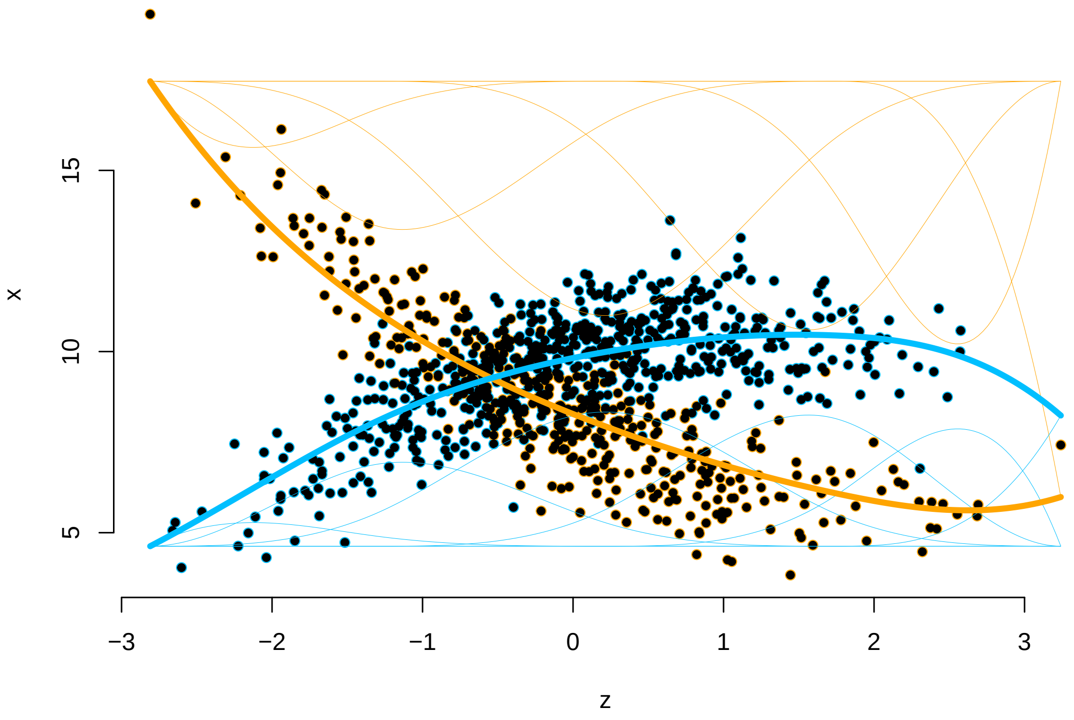
```r
theta.star = c(-2,-3, # starting values state process
               8, 10, rep(0, 2*ncol(Z)), # starting values for regression
               log(1),log(1)) # starting values for sigma
t1 = Sys.time()
mod_npreg = stats::nlm(mllk_npreg, theta.star, x = x, Z = Z, lambda = 10)
# small penalty
Sys.time()-t1
#> Time difference of 0.1977751 secs
```

Fitting a non-parametric Markov-switching regression model

```r
Gamma_hat_npreg = tpm(mod_npreg$estimate[1:2]) # calculating all tpms
delta_hat_npreg = stationary(Gamma_hat_npreg)
beta_hat_npreg = matrix(mod_npreg$estimate[2+1:(2+2*ncol(Z))], nrow = 2)
sigma_hat_npreg = exp(mod_npreg$estimate[2+2+2*ncol(Z) + 1:2])

zseq = seq(min(z), max(z), length = 200)
Zplot = splines::bs(x = zseq, df = 6)
xhat = cbind(1, Zplot)%*%t(beta_hat_npreg)

plot(z, x, pch = 16, bty = "n", xlab = "z", ylab = "x", col = color[s])
points(z, x, pch = 20)
for(j in 1:2){
  for(i in 1:ncol(Zplot)){
  lines(zseq, beta_hat_npreg[j,1] + Zplot[,i]*beta_hat_npreg[j,1+i], lwd = 0.3, col = color[j])
  }
}
```

```
lines(zseq, xhat[,1], lwd = 4, col = color[1])
lines(zseq, xhat[,2], lwd = 4, col = color[2])
```

Visualizing results



x
15
10
5
−3
−2
−1
0
1
2
3
z