Example for paper

Jan Schneider

7 8 2021

Example of asp21bpsplines

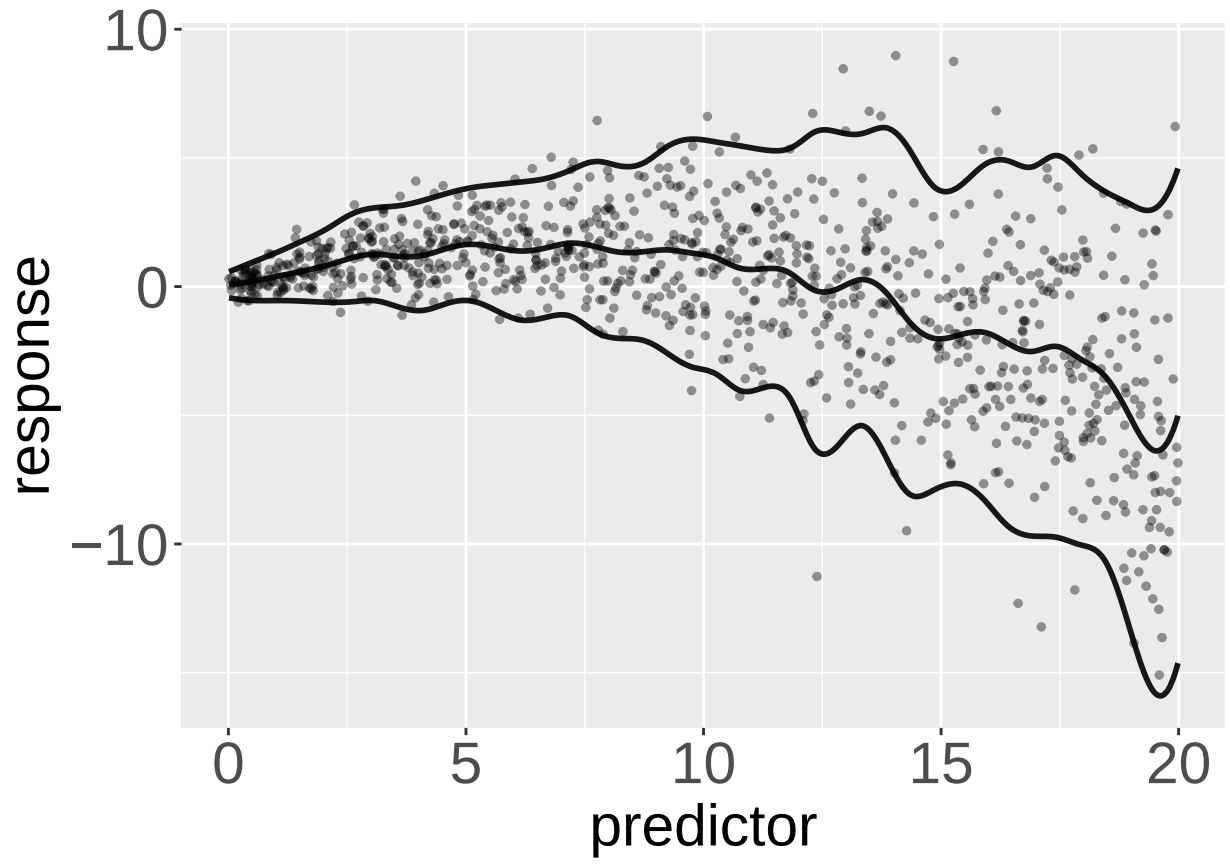
```
require(asp21bpspline)
set.seed(2)
n = 1000
x = c(runif(n, 0, 20))
mean = -0.04*x^2 + 0.5* x
y =  mean + rnorm(n, 0, 0.2 *x + 0.3)
m1 = list(x = x, z = x, y = y)
```

Consider you have the above data and want to fit a spline to it. With the asp21bpsplines package you would first call the spline function:

```
m = spline(m1, kn = c(40, 40), order = c(3,3), p_order = c(3, 3),
           smooth = c(1, 1))
summary(m)
```

```
##
## Call:
## spline(m = m1, kn = c(40, 40), order = c(3, 3), p_order = c(3,
##     3), smooth = c(1, 1))
##
##
##  predicted location and scale values:
##        location     scale
## [1,]  1.1593216 1.0271969
## [2,] -0.6962225 3.4022534
## [3,]  0.7050909 2.3334286
## [4,]  1.2198436 0.9608334
## [5,] -4.7071084 4.0837528
## [6,] -4.6820040 4.0742158
## ... ommited rows for better readability
```

```
plot(m)
```



add plot as separate file

The model estimates separate parameters for the location and the scale of the data. In this plot the middle line represents the prediction for the mean value and the outer lines represent the 95 % confidence interval for new predictions. However it seems that the predictions are overfitting a bit. One could now raise the values of the smoothing parameters in the call to the spline function. An alternative is a call to the MCMC function.

```
sample = mcmc.spline(m, it = 3000, burning = 1000, thinning = 10)
```

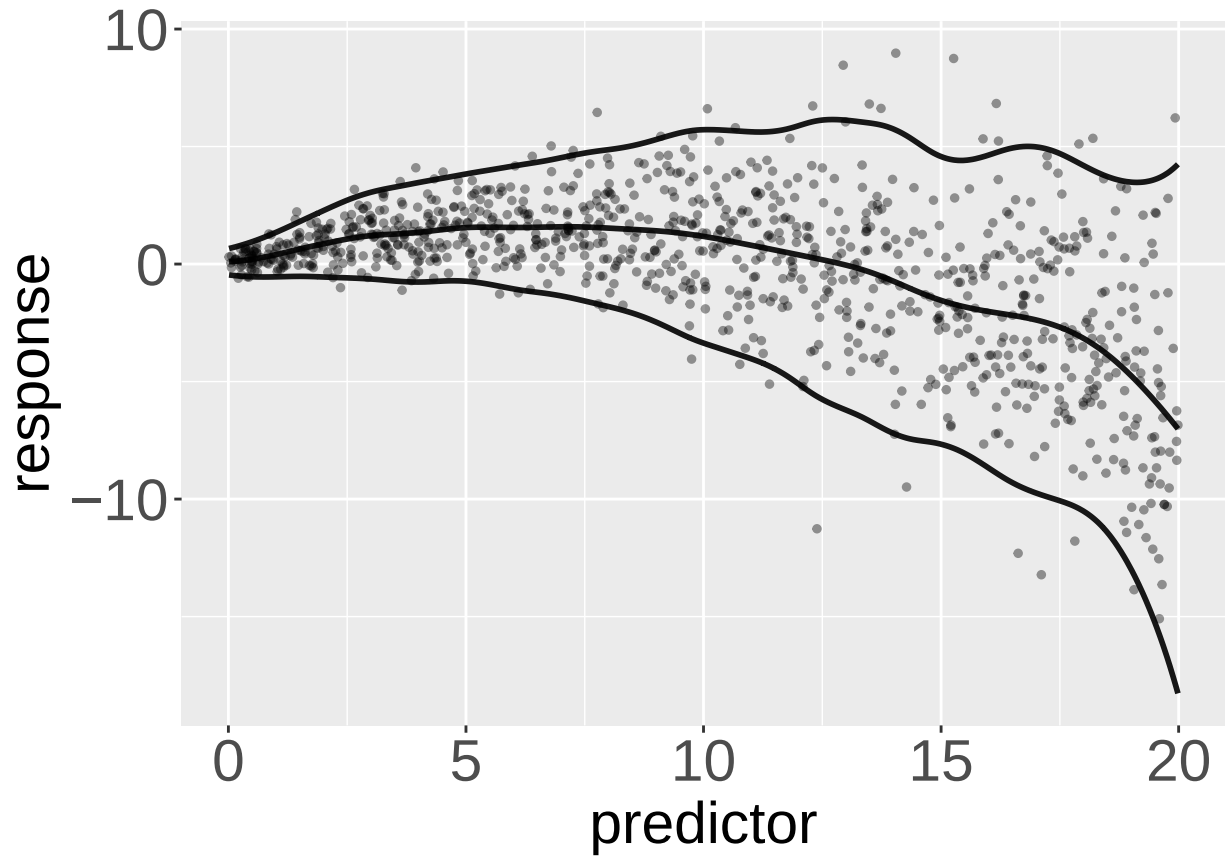
As the mcmc function samples directly from the posterior, the result is a sample for the beta and gamma parameters as well as for the smoothing values. The prediction with this sample is in result much smoother:

```
summary(sample)
```

```
##
## Call:
## spline(m = m1, kn = c(40, 40), order = c(3, 3), p_order = c(3,
##     3), smooth = c(1, 1))
##
## Posterior mean location coefficients:
##  pointwise quantiles obtained from posterior
##
```

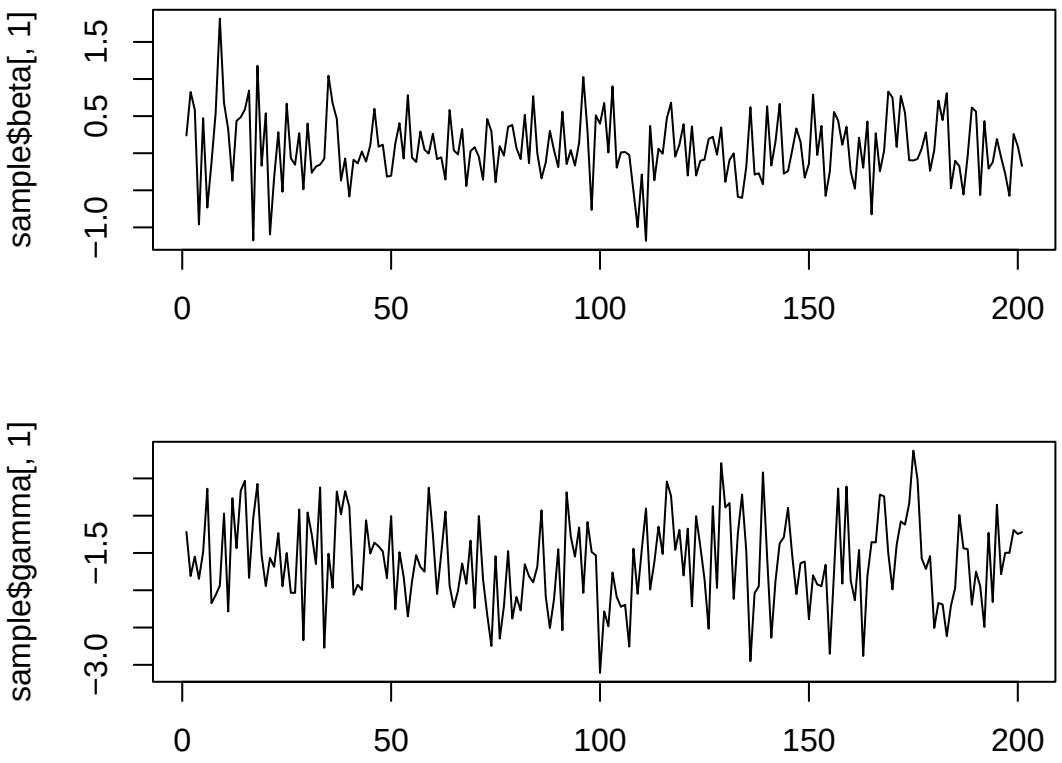
```
##         loc0.05     loc0.1     loc0.9    loc0.95 scale0.05  scale0.1 scale0.9
## [1,]  1.0905286  1.1356316  1.4341153  1.4827861 0.9023107 0.9349875 1.165731
## [2,] -1.3032634 -1.1616054 -0.3108342 -0.1872316 2.8863932 2.9406258 3.720967
## [3,]  0.1298224  0.2514631  0.9469496  1.0190048 2.1861501 2.2652788 2.919936
## [4,]  1.0712693  1.1080516  1.3952807  1.4616025 0.8781144 0.9046784 1.119026
## [5,] -5.1782263 -5.0460002 -3.9682080 -3.8169131 3.5343620 3.6181982 4.720823
## [6,] -5.1670267 -5.0333293 -3.9524769 -3.8044746 3.5298253 3.6110992 4.713276
##      scale0.95
## [1,]  1.188540
## [2,]  3.982602
## [3,]  2.984490
## [4,]  1.150923
## [5,]  4.796639
## [6,]  4.785620
## ...ommited rows for better readability
```

```
plot(sample, m)
```

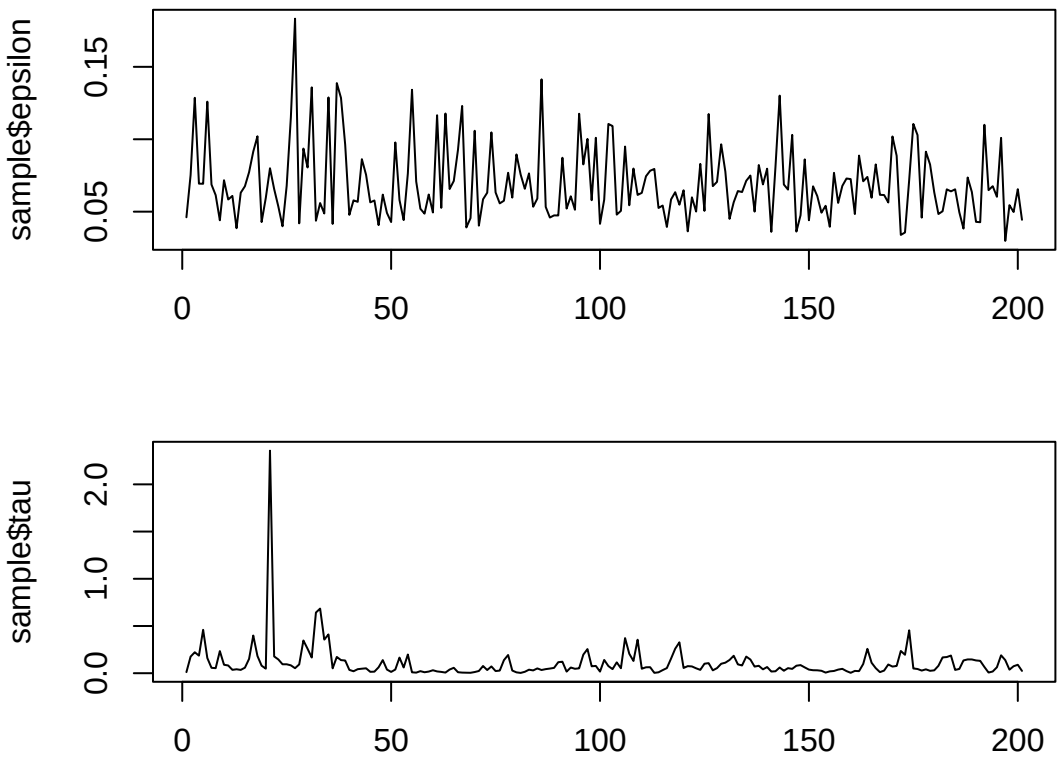


To investigate properties of the sample we can now look at the random walk for different parameters.

```
par(mfrow = c(2, 1))
par(mar = c(3, 5, 2, 4))
index = seq(1, length(sample$beta[,1]))
plot(index, sample$beta[,1], type = "l", xlab = "")
plot(index, sample$gamma[,1], type = "l", xlab = "")
```




```
plot(index, sample$epsilon, type = "l", xlab = "")
plot(index, sample$tau, type = "l")
```



While there seems to be no getting stuck, it seems that consecutive iterations are correlated. We now could choose a higher thinning value or alternatively set a custom stepsize for the sampling of the gamma values for the MCMC function.