Appendix A: Example for package asp21bpsplines

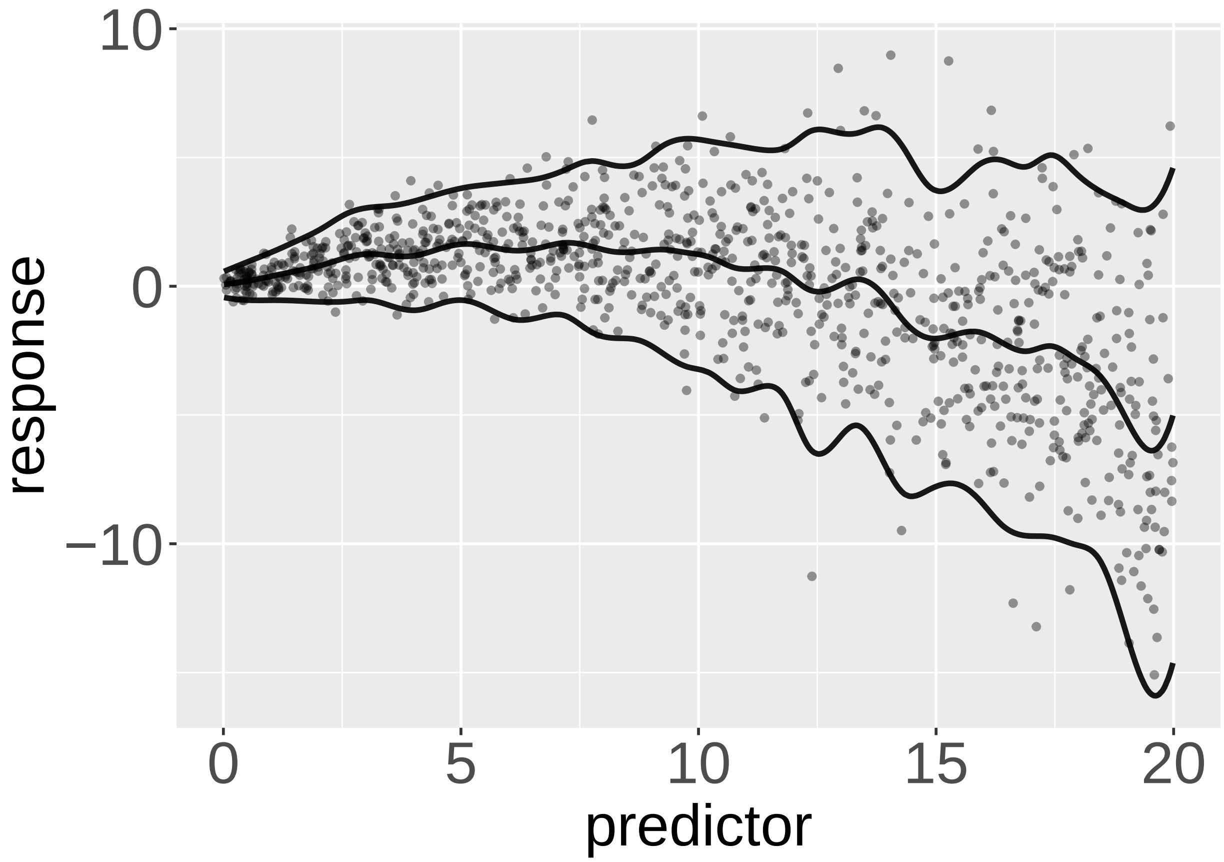
```r
require(asp21bpspline)
set.seed(2)
n = 1000
x = c(runif(n, 0, 20))
mean = -0.04*x^2 + 0.5* x
y =  mean + rnorm(n, 0, 0.2 *x + 0.3)
m1 = list(x = x, z = x, y = y)
```

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. The parameter m1 contains a list with the above structure, where x is the predictor for the mean, z is the predictor for the scale and y is the response. The parameter kn is short for knots and specifies the number of knots separately for the mean and the scale. In the same way the parameter stands for the order of the spline, p_order stands for the order of the smoothness penalty and smooth specifies the penalization strength.

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

```
##
## Call:
## spline(m = m1, kn = c(40, 40), order = c(3, 3), p_order = c(3,
##     3), smooth = c(1, 1))
##
## Location coefficients:
##  [1]  -0.01004   0.05816   0.20045   0.39816   0.60367   0.78701   1.12899
##  [8]   1.31770   1.13720   1.15841   1.52651   1.70267   1.51014   1.32741
## [15]   1.47338   1.76374   1.58687   1.28502   1.33392   1.48259   1.28549
## [22]   1.20726   0.59270   0.71117   0.72973  -0.31139  -0.18020   0.46701
## [29]  -0.06945  -1.55891  -2.16357  -1.88410  -1.63250  -2.21873  -2.70749
## [36]  -2.08524  -2.89393  -3.24740  -4.97449  -6.86876  -5.73026  -0.31415
##
## Scale coefficients:
##  [1]  -2.037993  -1.291260  -0.934212  -0.736558  -0.514893  -0.327901
##  [7]  -0.109484  -0.098214   0.007503   0.109413   0.076077   0.102036
## [13]   0.227166   0.335080   0.312734   0.329984   0.562001   0.534393
## [19]   0.523903   0.733961   0.834234   0.811065   0.913872   0.848839
## [25]   0.830546   1.204878   1.152149   1.004304   1.205800   1.271068
## [31]   1.064764   1.053703   1.202539   1.308075   1.280613   1.350264
## [37]   1.302844   1.259890   1.440140   1.583490   1.613201   1.503776
```

```r
plot(m)
```



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)
```

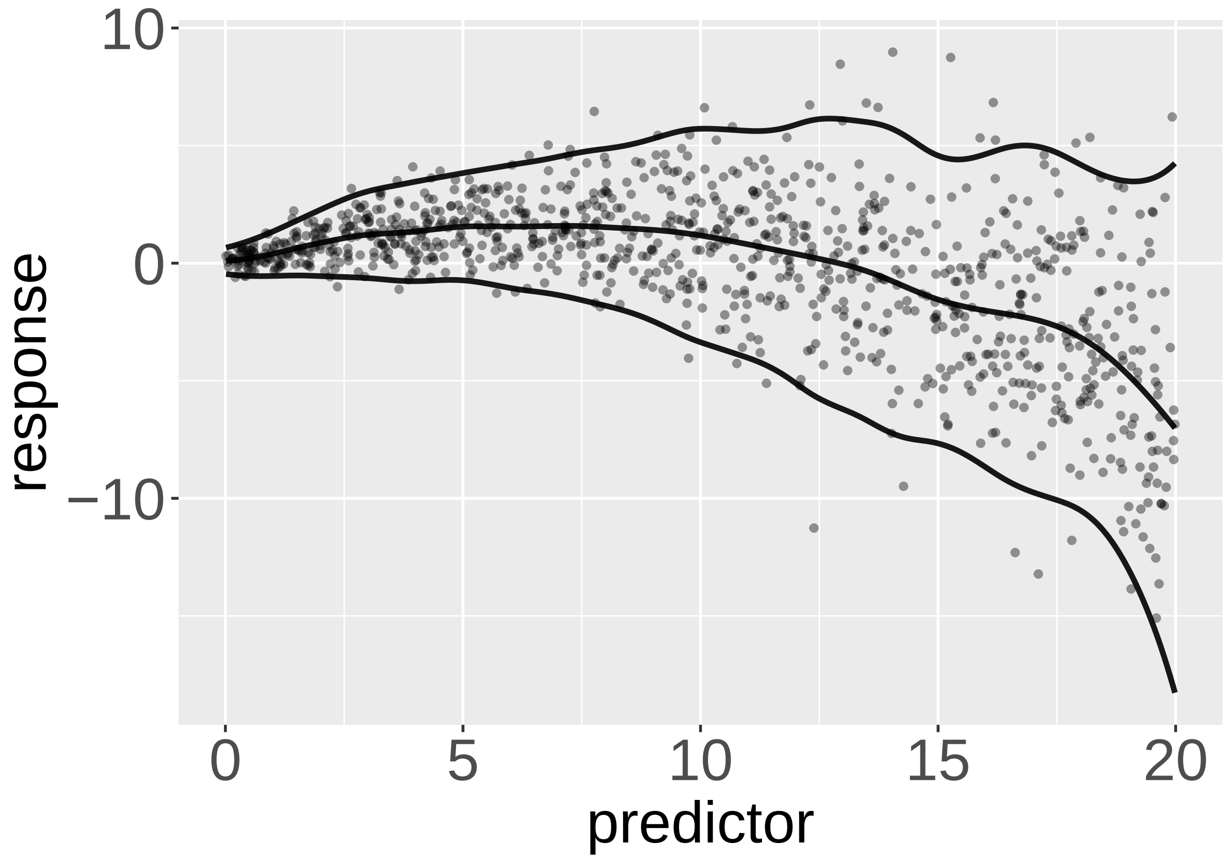
As the mcmc function samples directly from the posterior, the result is a sample for the beta and gamma parameters as well as for the smoothing values. The prediction with this sample is in result much smoother:

```
print(sample)
```

```
##
## Call:
## mcmc.spline(m = m, it = 3000, burning = 1000, thinning = 10)
##
## Posterior mean location coefficients:
##  [1]   0.07013   0.08235   0.19628   0.41081   0.67372   0.88701   1.10408
##  [8]   1.24236   1.27852   1.36592   1.49846   1.57784   1.56632   1.55068
## [15]   1.57373   1.59465   1.56635   1.50472   1.45330   1.38649   1.26112
## [22]   1.09661   0.89301   0.68837   0.47791   0.26807   0.05046  -0.21123
## [29]  -0.58268  -1.04506  -1.47314  -1.77933  -1.98965  -2.14580  -2.32080
## [36]  -2.60126  -3.05563  -3.72793  -4.64300  -5.74116  -6.98388  -8.38905
##
```

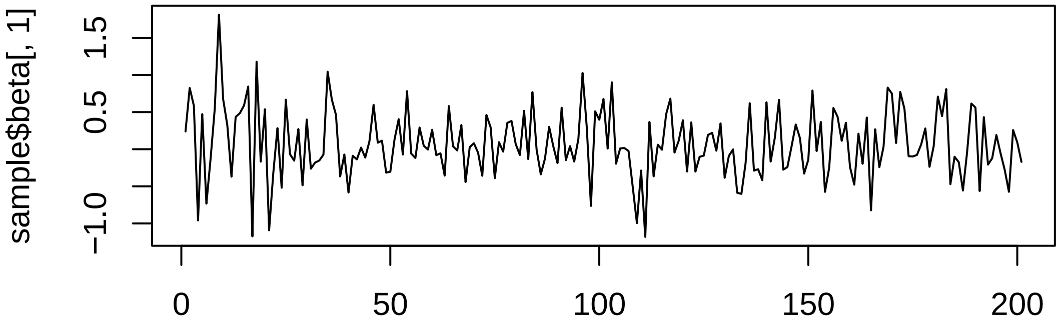
```
## Posterior mean scale coefficients:
##  [1]  -1.61617  -1.23520  -0.94355  -0.70916  -0.49745  -0.29903  -0.14274
##  [8]  -0.04028   0.03505   0.09194   0.11207   0.16372   0.23756   0.31270
## [15]   0.35964   0.43009   0.50132   0.55247   0.62782   0.72897   0.82257
## [22]   0.85963   0.88756   0.91642   0.98529   1.09457   1.13974   1.16014
## [29]   1.20051   1.19204   1.13349   1.13640   1.20382   1.28541   1.32647
## [36]   1.33085   1.31596   1.33539   1.41842   1.55369   1.73943   1.97703
```

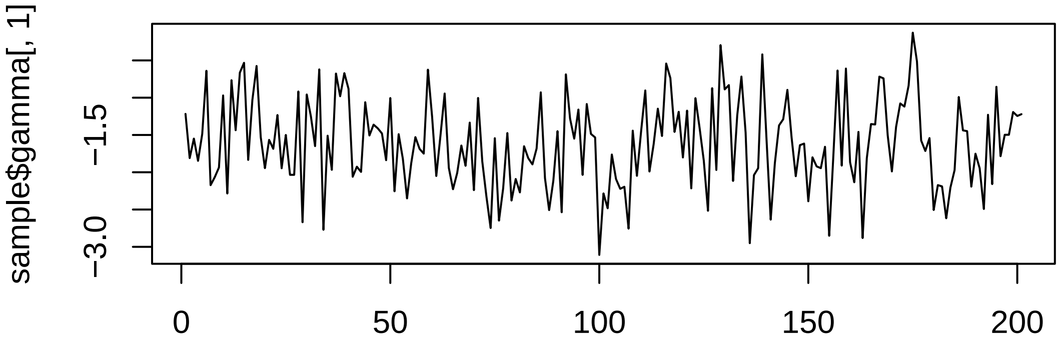
```
plot(sample, m)
```



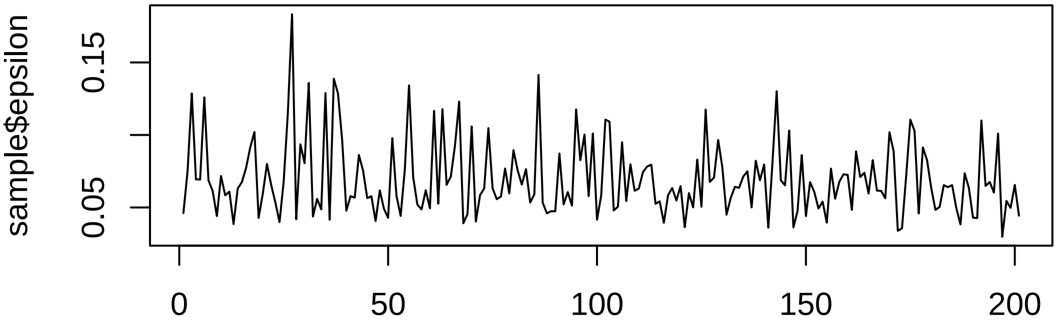
To investigate properties of the sample we can now look at the random walk for different parameters.

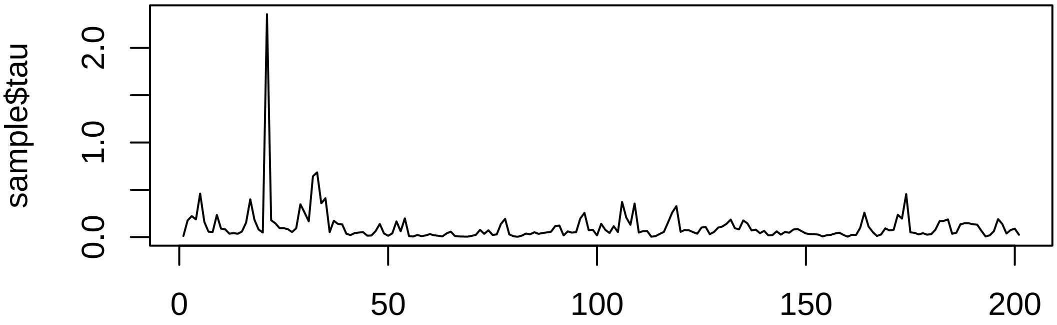
```
par(mfrow = c(2, 1))
par(mar = c(3, 5, 2, 4))
index = seq(1, length(sample$beta[,1]))
plot(index, sample$beta[,1], type = "l", xlab = "")
plot(index, sample$gamma[,1], type = "l", xlab = "")
```





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





For the sampling of gamma it is important that enough proposals are accepted in order to see movement in the random walk. This seems to be the case. However, it might be that consecutive iterations are correlated, since consecutive values tend to be near to each other. 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 to alleviate this issue.