BDA - Assignment 8

Anonymous

16 11 2019

Contents

Used libraries:

```
#install.packages('remotes')
#remotes::install_github("avehtari/BDA_course_Aalto", subdir = "rpackage")
library(dplyr)
library(ggplot2)
library(rstan)
library(gdata)
library(bayesplot)
library(aaltobda)
library(loo)
data("factory")
```

```
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
```

Exercise 1

I will write 3 stan models, by default with uniform priors. Only if some models fail with PSIS-LOO (i.e., k_hat values over 0.7) I will try inserting priors to get better loo draws. I hope the Grader will consider the possible differences in elpd_loo and k_hat values, as well as possibly differing order of preferred models (**this topic was discussed in slack with Aki**).

NOTE: I have the predictive sampling in the generated quantities, even though I won't be needing them in this exercise - they are for additional model and convergence checking.

Stan models:

Pooled

```
writeLines(readLines('ass8_pooled.stan'))
```

```
##
## data {
##   int<lower=0> N;
##   vector[N] y;
## }
```

```
##
## parameters {
##   real mu_P;
##   real<lower=0> sigma_P;
## }
##
## transformed parameters {
## }
##
## model {
##   y ~ normal(mu_P, sigma_P); //uniform priors
## }
##
## generated quantities {
##   real ypred_P;
##   vector[N] log_lik;
##   //prediction (not for any specific machine)
##   ypred_P = normal_rng(mu_P, sigma_P);
##   //likelihood
##   for(i in 1:N) {
##     log_lik[i] = normal_lpdf(y[i] | mu_P, sigma_P);
##   }
## }
```

Separate

```
writeLines(readLines('ass8_separate.stan'))
```

```
##
##       data {
##         int<lower=0> N;
##         int<lower=0> K;
##         int<lower=0, upper=K> x[N];
##         vector[N] y;
##       }
##       transformed data {
##         int<lower=0> n = N/K;
##       }
##       parameters {
##         //PARAMS FOR SEPARATE
##         vector[K] mu_S;
##         vector<lower=0>[K] sigma_S;
##       }
##       model {
##         //MODEL FOR SEPARATE (S)
##         //mu_S ~ normal(90,50);//Optional weak prior
##         //sigma_S ~ scaled_inv_chi_square(K-1,8); //weak prior (for better convergence)
##         y ~ normal(mu_S[x], sigma_S[x]);
##       }
##       generated quantities {
##         real ypred_S[6];
##         vector[N] log_lik;
##
##         //PREDICTION FOR SEPARATE
##         ypred_S = normal_rng(mu_S, sigma_S);
```

```
##
##         //likelihood
##         //NOTE: Indexing (per group) is 1,2,3,4,5,6,1,2,3,4,5,6,...
##         for(obs in 1:n) {
##           for(group in 1:K) {
##             log_lik[group+6*(obs-1)] = normal_lpdf(y[group+6*(obs-1)] | mu_S[group], sigma_S[group])
##           }
##         }
##
##       }
```

Hierarchical

```
writeLines(readLines('ass8_hierarch.stan'))
```

```
##
##
##       data {
##         int<lower=0> N;
##         int<lower=0> K;
##         int<lower=0, upper=K> x[N];
##         vector[N] y;
##       }
##       transformed data {
##         int<lower=0> n = N/K;
##       }
##       parameters {
##         real mu_0_H;
##         real<lower=0> sigma_0_H;
##         vector[K] mu_H;
##         real<lower=0> sigma_H;
##       }
##       transformed parameters {
##
##       }
##       model {
##         //NOTE: I removed the hyperpriors for easier comparison with reference results.
##         //mu_0_H ~ normal(90,50); // weak hyperprior (approx. the pooled distribution)
##         //sigma_0_H ~ scaled_inv_chi_square(K-1,8); // weak hyperprior
##         //sigma_H ~ cauchy(0,8); // weak prior
##         mu_H ~ normal(mu_0_H,sigma_0_H); // population prior
##         y ~ normal(mu_H[x], sigma_H);
##       }
##       generated quantities {
##         real ypred_H[6];
##         vector[N] log_lik;
##         real mu_7_p;
##         real mu_7;
##
##         //NOTE: Indexing (per group) is 1,2,3,4,5,6,1,2,3,4,5,6,...
##         for(obs in 1:n) {
##           for(group in 1:K) {
##             log_lik[group+6*(obs-1)] = normal_lpdf(y[group+6*(obs-1)] | mu_H[group], sigma_H);
##           }
##         }
```

```
##
##         //PREDICTION FOR HIERACHICAL
##         ypred_H = normal_rng(mu_H,sigma_H);
##         mu_7_p = normal_rng(mu_0_H, sigma_0_H);
##         mu_7 = normal_rng(mu_7_p, sigma_H);
##         /*
##         These mu_7 draws could also be pulled from
##         normal_rng(mu_0_H, sigma_H), since sigma_H
##         is shared between all groups, and mu_7_p is
##         just mu_0_H. But I felt this would be more
##         explicit...
##         */
##       }
##
```

NOTE: The indexing is not linear, but 1,2,3,4,5,6,1,2,... - the reason for this is, that the input data is indexed in following way:

```
all_data <-list(N = 6*nrow(factory),
             K = 6,
             x = rep(1:6, nrow(factory)),
             y = c(t(factory[,1:6])))

pooled_data <- list(N = 6*nrow(factory),
             y = c(t(factory[,1:6])))

all_data$x
```

```
##  [1] 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6
```

```
all_data$y
```

```
##  [1]  83 117 101 105  79  57  92 109  93 119  97  92  92 114  92 116 103
## [18] 104  46 104  86 102  79  77  67  87  67 116  92 100
```

And so for separate and hierarchical models I have to draw the samples as if from a matrix.

Model compilation:

Sampling:Four chains did suffice with weak priors. Warmup period is on the longer side, but that's mostly an added insurance. Overall 8000 draws are produced for each posterior mu and prediction. Adapt_delta is high mostly for consistency, and only hierarchical model needs such a high value (priors can also be used to assure convergence). Tuning is mostly done with predictive distributions (2 weeks ago).

```
stan_model_pooled <- "ass8_pooled.stan"
stan_model_separate <- "./ass8_separate.stan"
stan_model_hierarch <- "ass8_hierarch.stan"

fit_pooled <- stan(file = stan_model_pooled, data = pooled_data, warmup = 1000, iter = 3000,
                   control=list(adapt_delta=0.98))


fit_separate <- stan(file = stan_model_separate, data = all_data, warmup = 1000, iter = 3000,
                   control=list(adapt_delta=0.98))

fit_hierarch <- stan(file = stan_model_hierarch, data = all_data, warmup = 1000, iter = 3000,
                   control=list(adapt_delta=0.98))
```

Just a quick look that everything converges and Rhat's look good:

```
####################
#POOLED
posterior_pooled <- extract(fit_pooled)
m_pooled <- monitor(fit_pooled)
```

```
## Inference for the input samples (4 chains: each with iter=2000; warmup=0):
##
##              mean se_mean   sd   2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu_P         92.9     0.0  3.5   86.1  90.7  92.9  95.2  99.8  4938    1
## sigma_P      18.8     0.0  2.6   14.5  17.0  18.5  20.3  24.7  4501    1
## ypred_P      93.1     0.2 19.1   54.9  80.5  93.1 105.7 130.6  7919    1
## log_lik[1]   -4.0     0.0  0.1   -4.3  -4.1  -4.0  -3.9  -3.8  4555    1
## log_lik[2]   -4.7     0.0  0.3   -5.3  -4.9  -4.7  -4.5  -4.3  5069    1
## log_lik[3]   -4.0     0.0  0.1   -4.3  -4.0  -4.0  -3.9  -3.7  4593    1
## log_lik[4]   -4.1     0.0  0.1   -4.4  -4.2  -4.1  -4.0  -3.8  4888    1
## log_lik[5]   -4.2     0.0  0.2   -4.5  -4.2  -4.1  -4.0  -3.9  4986    1
## log_lik[6]   -5.8     0.0  0.5   -7.0  -6.1  -5.7  -5.4  -4.9  5369    1
## log_lik[7]   -3.9     0.0  0.1   -4.1  -3.9  -3.9  -3.8  -3.6  4261    1
## log_lik[8]   -4.2     0.0  0.2   -4.6  -4.3  -4.2  -4.1  -4.0  5073    1
## log_lik[9]   -3.9     0.0  0.1   -4.1  -3.9  -3.9  -3.8  -3.6  4271    1
## log_lik[10]  -4.9     0.0  0.3   -5.5  -5.0  -4.8  -4.7  -4.4  5033    1
## log_lik[11]  -3.9     0.0  0.1   -4.2  -4.0  -3.9  -3.8  -3.6  4379    1
## log_lik[12]  -3.9     0.0  0.1   -4.1  -3.9  -3.9  -3.8  -3.6  4261    1
## log_lik[13]  -3.9     0.0  0.1   -4.1  -3.9  -3.9  -3.8  -3.6  4261    1
## log_lik[14]  -4.5     0.0  0.2   -5.0  -4.6  -4.5  -4.4  -4.2  5110    1
## log_lik[15]  -3.9     0.0  0.1   -4.1  -3.9  -3.9  -3.8  -3.6  4261    1
## log_lik[16]  -4.7     0.0  0.2   -5.2  -4.8  -4.6  -4.5  -4.2  5085    1
## log_lik[17]  -4.0     0.0  0.1   -4.3  -4.1  -4.0  -3.9  -3.8  4736    1
## log_lik[18]  -4.0     0.0  0.1   -4.3  -4.1  -4.0  -3.9  -3.8  4812    1
## log_lik[19]  -7.2     0.0  0.9   -9.2  -7.7  -7.1  -6.5  -5.7  5207    1
## log_lik[20]  -4.0     0.0  0.1   -4.3  -4.1  -4.0  -3.9  -3.8  4812    1
## log_lik[21]  -3.9     0.0  0.1   -4.2  -4.0  -3.9  -3.8  -3.7  4367    1
## log_lik[22]  -4.0     0.0  0.1   -4.3  -4.1  -4.0  -3.9  -3.7  4662    1
## log_lik[23]  -4.2     0.0  0.2   -4.5  -4.2  -4.1  -4.0  -3.9  4986    1
## log_lik[24]  -4.2     0.0  0.2   -4.6  -4.3  -4.2  -4.1  -3.9  5182    1
## log_lik[25]  -4.9     0.0  0.3   -5.5  -5.0  -4.8  -4.7  -4.4  5544    1
## log_lik[26]  -3.9     0.0  0.1   -4.2  -4.0  -3.9  -3.8  -3.7  4325    1
## log_lik[27]  -4.9     0.0  0.3   -5.5  -5.0  -4.8  -4.7  -4.4  5544    1
## log_lik[28]  -4.7     0.0  0.2   -5.2  -4.8  -4.6  -4.5  -4.2  5085    1
## log_lik[29]  -3.9     0.0  0.1   -4.1  -3.9  -3.9  -3.8  -3.6  4261    1
## log_lik[30]  -3.9     0.0  0.1   -4.2  -4.0  -3.9  -3.8  -3.7  4529    1
## lp__        -99.3     0.0  1.0 -102.2 -99.7 -99.0 -98.6 -98.3  2614    1
##
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
paste('Rhat values over 1.05: ',sum(m_pooled[,10] > 1.05)) %>% print()
```

```
## [1] "Rhat values over 1.05:  0"
```

```
####################
#SEPARATE
posterior_sep <- extract(fit_separate)
```

```
m_sep <- monitor(fit_separate)
```

```
## Inference for the input samples (4 chains: each with iter=2000; warmup=0):
##
##               mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu_S[1]       75.4     0.3 15.6  41.1  68.1  75.8  83.5 105.0  2143    1
## mu_S[2]      106.0     0.2  9.2  86.6 101.6 106.0 110.7 124.2  2779    1
## mu_S[3]       87.7     0.2  9.7  67.8  82.9  87.8  92.6 106.6  2799    1
## mu_S[4]      111.8     0.2  6.7  98.9 108.6 111.7 114.6 125.0  1612    1
## mu_S[5]       89.2     0.6 14.8  70.6  85.7  89.9  94.1 106.6   572    1
## mu_S[6]       87.4     0.6 18.3  57.3  78.8  86.3  94.0 123.6   971    1
## sigma_S[1]    30.7     0.4 19.0  12.9  19.3  25.3  35.4  79.3  2205    1
## sigma_S[2]    18.3     0.3 12.4   7.8  11.7  15.2  20.8  48.3  2279    1
## sigma_S[3]    19.8     0.3 13.7   8.3  12.5  16.3  22.8  51.1  2047    1
## sigma_S[4]    12.1     0.2  8.0   5.0   7.5  10.0  13.9  31.1  1530    1
## sigma_S[5]    17.6     0.8 19.3   7.0  10.7  14.0  19.5  45.7   660    1
## sigma_S[6]    31.6     0.7 26.7  12.5  19.0  24.9  35.2  93.8  1301    1
## ypred_S[1]    76.0     0.5 39.3  -4.7  57.5  76.5  94.9 151.0  6080    1
## ypred_S[2]   105.9     0.3 24.4  59.8  94.8 106.0 117.3 150.4  6850    1
## ypred_S[3]    87.7     0.3 26.2  38.5  75.4  87.3  99.5 136.8  5847    1
## ypred_S[4]   112.0     0.3 16.5  81.3 104.5 111.6 119.0 143.3  3626    1
## ypred_S[5]    89.4     0.5 32.3  43.9  79.6  90.0 100.4 132.6  3619    1
## ypred_S[6]    87.5     0.7 45.9  10.7  67.4  86.3 105.4 170.5  4471    1
## log_lik[1]    -4.4     0.0  0.5  -5.5  -4.6  -4.3  -4.0  -3.6  2287    1
## log_lik[2]    -4.1     0.0  0.5  -5.2  -4.4  -4.1  -3.8  -3.4  3239    1
## log_lik[3]    -4.3     0.0  0.5  -5.4  -4.5  -4.2  -3.9  -3.5  3267    1
## log_lik[4]    -3.7     0.0  0.5  -4.8  -3.9  -3.6  -3.3  -2.9  2631    1
## log_lik[5]    -4.1     0.0  0.5  -5.3  -4.4  -4.0  -3.8  -3.3  2017    1
## log_lik[6]    -5.2     0.0  0.7  -7.1  -5.5  -5.0  -4.7  -4.2  6166    1
## log_lik[7]    -4.6     0.0  0.5  -5.7  -4.8  -4.5  -4.2  -3.8  2980    1
## log_lik[8]    -3.8     0.0  0.5  -4.9  -4.1  -3.8  -3.5  -3.1  2060    1
## log_lik[9]    -3.9     0.0  0.5  -5.0  -4.2  -3.9  -3.6  -3.2  2167    1
## log_lik[10]   -3.7     0.0  0.5  -4.9  -4.0  -3.7  -3.4  -3.0  2799    1
## log_lik[11]   -3.9     0.0  0.5  -5.0  -4.1  -3.8  -3.5  -3.1  1373    1
## log_lik[12]   -4.3     0.0  0.5  -5.6  -4.6  -4.3  -4.0  -3.5  1504    1
## log_lik[13]   -4.6     0.0  0.5  -5.7  -4.8  -4.5  -4.2  -3.8  2980    1
## log_lik[14]   -4.0     0.0  0.5  -5.0  -4.2  -3.9  -3.6  -3.2  2459    1
## log_lik[15]   -3.9     0.0  0.5  -5.0  -4.2  -3.9  -3.6  -3.2  2150    1
## log_lik[16]   -3.5     0.0  0.5  -4.6  -3.8  -3.5  -3.2  -2.7  1981    1
## log_lik[17]   -4.3     0.0  0.6  -5.6  -4.6  -4.2  -3.9  -3.5  2602    1
## log_lik[18]   -4.6     0.0  0.5  -5.8  -4.9  -4.6  -4.3  -3.9  2280    1
## log_lik[19]   -5.2     0.0  0.7  -7.0  -5.5  -5.0  -4.7  -4.3  8325    1
## log_lik[20]   -3.8     0.0  0.5  -4.9  -4.1  -3.8  -3.5  -3.0  2040    1
## log_lik[21]   -3.9     0.0  0.5  -5.0  -4.2  -3.8  -3.5  -3.1  2180    1
## log_lik[22]   -4.0     0.0  0.6  -5.4  -4.2  -3.9  -3.6  -3.1  5214    1
## log_lik[23]   -4.1     0.0  0.5  -5.3  -4.4  -4.0  -3.8  -3.3  2017    1
## log_lik[24]   -4.4     0.0  0.5  -5.6  -4.7  -4.3  -4.0  -3.6  1473    1
## log_lik[25]   -4.4     0.0  0.5  -5.5  -4.7  -4.3  -4.1  -3.6  2337    1
## log_lik[26]   -4.8     0.0  0.8  -6.8  -5.1  -4.6  -4.3  -3.8  9573    1
## log_lik[27]   -4.9     0.0  0.8  -7.0  -5.2  -4.7  -4.4  -3.9  9186    1
## log_lik[28]   -3.5     0.0  0.5  -4.6  -3.8  -3.5  -3.2  -2.7  1981    1
## log_lik[29]   -3.7     0.0  0.5  -4.9  -4.0  -3.7  -3.4  -2.9  1294    1
## log_lik[30]   -4.5     0.0  0.5  -5.6  -4.8  -4.5  -4.2  -3.7  1805    1
## lp__         -81.2     0.1  3.2 -88.6 -83.1 -80.8 -78.9 -76.3  1261    1
```

```
##
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
paste('Rhat values over 1.05: ',sum(m_sep[,10] > 1.05)) %>% print()
```

```
## [1] "Rhat values over 1.05:  0"
```

```
####################
#HIERARCH
posterior_hierarch <- extract(fit_hierarch)
m_hierarch <- monitor(fit_hierarch)
```

```
## Inference for the input samples (4 chains: each with iter=2000; warmup=0):
##
##               mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff
## mu_0_H        93.3     0.2  9.6   76.2   88.7   93.0   97.5  110.4  1793
## sigma_0_H     16.9     0.3 12.5    5.2   10.6   14.4   19.9   42.1  1683
## mu_H[1]       79.7     0.1  6.8   66.3   75.3   79.6   84.2   93.3  5410
## mu_H[2]      103.3     0.1  6.6   90.5   99.0  103.4  107.7  116.7  6582
## mu_H[3]       89.0     0.1  6.1   77.1   85.0   89.0   92.9  101.0  7132
## mu_H[4]      107.5     0.1  6.8   93.9  103.1  107.7  112.1  120.7  5004
## mu_H[5]       90.7     0.1  6.3   78.2   86.4   90.7   94.9  103.1  7519
## mu_H[6]       87.6     0.1  6.3   75.1   83.6   87.7   91.6  100.0  7755
## sigma_H       15.2     0.0  2.4   11.3   13.6   15.0   16.6   20.6  5536
## ypred_H[1]    80.0     0.2 16.8   47.7   68.9   79.6   90.9  113.9  7605
## ypred_H[2]   103.0     0.2 16.8   69.6   92.1  103.2  114.1  136.1  7873
## ypred_H[3]    88.7     0.2 16.6   56.6   77.7   88.4   99.7  121.5  6994
## ypred_H[4]   107.3     0.2 16.9   73.3   96.2  107.7  118.5  140.6  7431
## ypred_H[5]    90.6     0.2 16.6   57.9   79.7   90.5  101.7  123.2  7982
## ypred_H[6]    87.7     0.2 16.8   54.8   76.7   87.6   98.5  120.8  8156
## log_lik[1]    -3.8     0.0  0.2   -4.3   -3.9   -3.7   -3.6   -3.4  4969
## log_lik[2]    -4.1     0.0  0.4   -5.1   -4.4   -4.1   -3.8   -3.6  6584
## log_lik[3]    -4.0     0.0  0.4   -4.9   -4.2   -4.0   -3.8   -3.6  7344
## log_lik[4]    -3.7     0.0  0.2   -4.2   -3.9   -3.7   -3.6   -3.4  4838
## log_lik[5]    -4.0     0.0  0.4   -4.9   -4.2   -4.0   -3.8   -3.5  6683
## log_lik[6]    -5.9     0.0  1.0   -8.1   -6.4   -5.7   -5.2   -4.4  9000
## log_lik[7]    -4.1     0.0  0.4   -5.1   -4.3   -4.0   -3.8   -3.6  6460
## log_lik[8]    -3.8     0.0  0.2   -4.4   -3.9   -3.8   -3.6   -3.4  4609
## log_lik[9]    -3.7     0.0  0.2   -4.2   -3.9   -3.7   -3.6   -3.4  4937
## log_lik[10]   -4.0     0.0  0.4   -4.9   -4.2   -3.9   -3.7   -3.5  5062
## log_lik[11]   -3.8     0.0  0.2   -4.4   -3.9   -3.8   -3.6   -3.4  5758
## log_lik[12]   -3.8     0.0  0.2   -4.3   -3.9   -3.7   -3.6   -3.4  4837
## log_lik[13]   -4.1     0.0  0.4   -5.1   -4.3   -4.0   -3.8   -3.6  6460
## log_lik[14]   -4.0     0.0  0.3   -4.8   -4.2   -3.9   -3.7   -3.5  5868
## log_lik[15]   -3.7     0.0  0.2   -4.2   -3.8   -3.7   -3.6   -3.4  4542
## log_lik[16]   -3.9     0.0  0.3   -4.6   -4.0   -3.8   -3.7   -3.4  4664
## log_lik[17]   -4.1     0.0  0.4   -5.0   -4.3   -4.0   -3.8   -3.5  7126
## log_lik[18]   -4.3     0.0  0.5   -5.5   -4.6   -4.2   -4.0   -3.7  7295
## log_lik[19]   -6.3     0.0  1.1   -8.8   -7.0   -6.2   -5.5   -4.5  8273
## log_lik[20]   -3.7     0.0  0.2   -4.2   -3.8   -3.7   -3.6   -3.4  4052
## log_lik[21]   -3.7     0.0  0.2   -4.2   -3.8   -3.7   -3.6   -3.4  4249
## log_lik[22]   -3.8     0.0  0.2   -4.4   -3.9   -3.8   -3.6   -3.5  5613
## log_lik[23]   -4.0     0.0  0.4   -4.9   -4.2   -4.0   -3.8   -3.5  6683
```

```
## log_lik[24]   -4.0     0.0  0.3   -4.8   -4.1   -3.9   -3.7   -3.5  6835
## log_lik[25]   -4.1     0.0  0.4   -5.0   -4.3   -4.0   -3.8   -3.5  5146
## log_lik[26]   -4.4     0.0  0.5   -5.6   -4.6   -4.2   -4.0   -3.7  6824
## log_lik[27]   -4.8     0.0  0.6   -6.3   -5.2   -4.7   -4.4   -3.9  7654
## log_lik[28]   -3.9     0.0  0.3   -4.6   -4.0   -3.8   -3.7   -3.4  4664
## log_lik[29]   -3.7     0.0  0.2   -4.1   -3.8   -3.7   -3.6   -3.4  4530
## log_lik[30]   -4.1     0.0  0.4   -5.0   -4.2   -4.0   -3.8   -3.6  6921
## mu_7_p         94.0     0.4 24.9   52.1   83.2   93.5  104.2  137.8  3085
## mu_7           94.0     0.5 29.2   43.8   78.0   93.4  109.1  146.9  3729
## lp__         -109.0     0.1  2.5 -115.0 -110.4 -108.6 -107.1 -105.3  2232
##              Rhat
## mu_0_H          1
## sigma_0_H       1
## mu_H[1]         1
## mu_H[2]         1
## mu_H[3]         1
## mu_H[4]         1
## mu_H[5]         1
## mu_H[6]         1
## sigma_H         1
## ypred_H[1]      1
## ypred_H[2]      1
## ypred_H[3]      1
## ypred_H[4]      1
## ypred_H[5]      1
## ypred_H[6]      1
## log_lik[1]      1
## log_lik[2]      1
## log_lik[3]      1
## log_lik[4]      1
## log_lik[5]      1
## log_lik[6]      1
## log_lik[7]      1
## log_lik[8]      1
## log_lik[9]      1
## log_lik[10]     1
## log_lik[11]     1
## log_lik[12]     1
## log_lik[13]     1
## log_lik[14]     1
## log_lik[15]     1
## log_lik[16]     1
## log_lik[17]     1
## log_lik[18]     1
## log_lik[19]     1
## log_lik[20]     1
## log_lik[21]     1
## log_lik[22]     1
## log_lik[23]     1
## log_lik[24]     1
## log_lik[25]     1
## log_lik[26]     1
## log_lik[27]     1
## log_lik[28]     1
```

```
## log_lik[29]    1
## log_lik[30]    1
## mu_7_p         1
## mu_7           1
## lp__           1
##
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
paste('Rhat values over 1.05: ',sum(m_hierarch[,10] > 1.05)) %>% print()
```

```
## [1] "Rhat values over 1.05:  0"
```

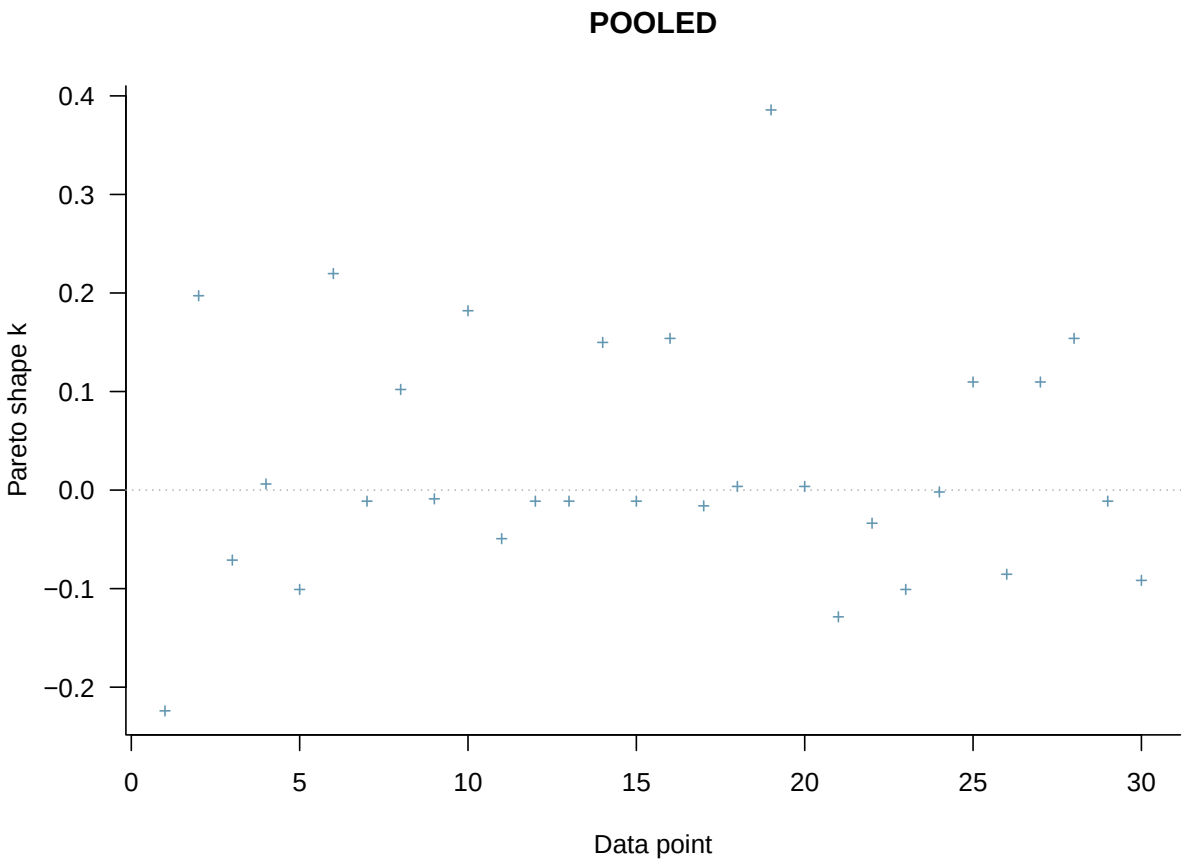
All chains seem to have converged.

Exercise 2

PSIS-LOO elpd and k_hat values.

POOLED LOO:

```
####################
#POOLED
res_pooled <- extract_log_lik(fit_pooled)
loo_pooled <- loo(fit_pooled)
plot(loo_pooled, main='POOLED')
```




```
#k_hat
pareto_k_table(loo_pooled)
```

```
##
## All Pareto k estimates are good (k < 0.5).
```

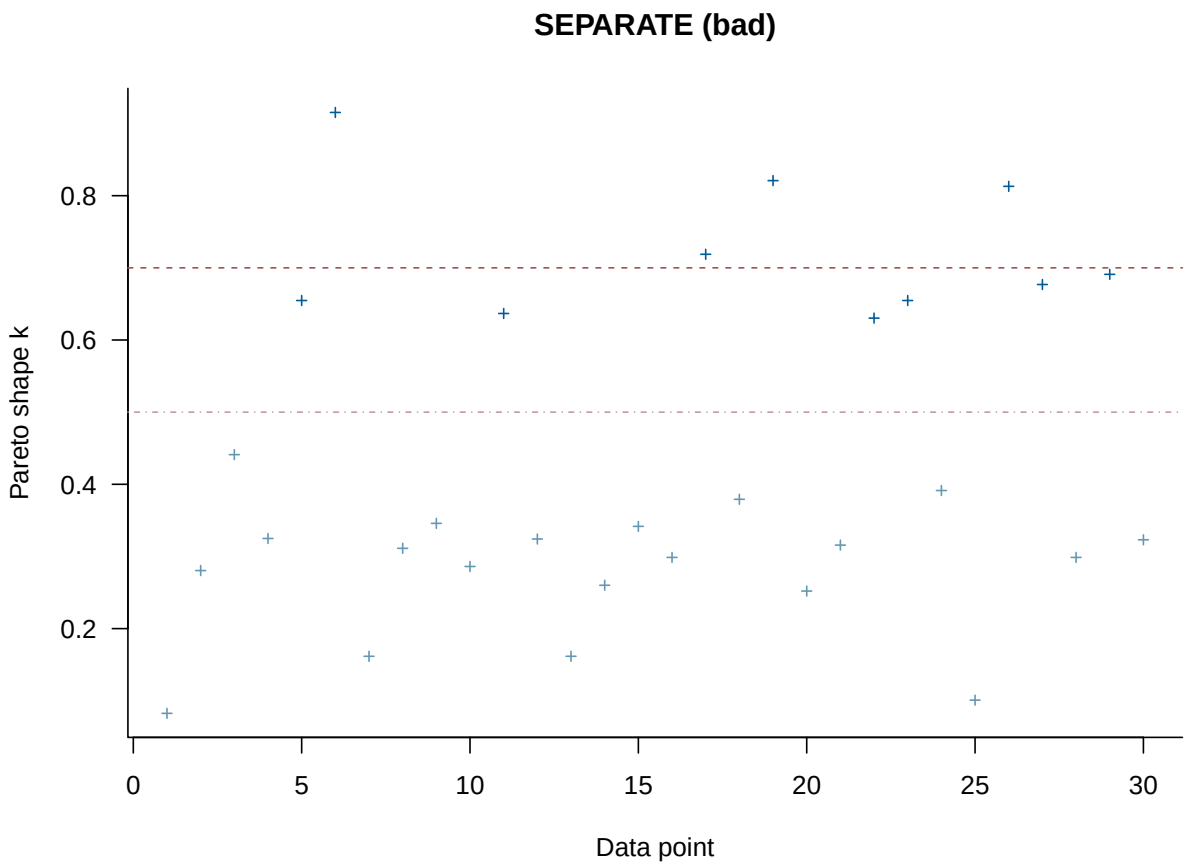
For pooled model, PSIS-LOO estimate looks valid (uniform priors). Here's the return value for loo(...)

```
loo_pooled
```

```
##
## Computed from 8000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -131.0 4.3
## p_loo         2.0 0.8
## looic       261.9 8.5
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

SEPARATE LOO:

```
####################
#SEPARATE
res_separate <- extract_log_lik(fit_separate)
loo_separate <- loo(fit_separate)
plot(loo_separate, main='SEPARATE (bad)')
```




```
#k_hat
pareto_k_table(loo_separate)
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     20    66.7%   1457
##  (0.5, 0.7]   (ok)        6    20.0%   436
##    (0.7, 1]   (bad)       4    13.3%   177
##    (1, Inf)   (very bad)  0     0.0%   <NA>
```

For separate model, with uniform prior, psis-loo extimate can't be trusted:

```
loo_separate
```

```
##
## Computed from 8000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -132.3 3.1
```

```
## p_loo         9.8 1.0
## looic       264.5 6.2
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     20    66.7%   1457
##  (0.5, 0.7]   (ok)        6    20.0%   436
##    (0.7, 1]   (bad)       4    13.3%   177
##    (1, Inf)   (very bad)  0     0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

This is not surprising, since each separate machine is modelled only with 5 data points; leave one out and you loose 20% of your information (per machine). PSIS-LOO estimate with non-informative prior can not therefore be trusted.

To fix the issue, I will regularize the model with weakly informative priors:

```
writeLines(readLines('ass8_separate_priors.stan'))
```

```
##
##       data {
##         int<lower=0> N;
##         int<lower=0> K;
##         int<lower=0, upper=K> x[N];
##         vector[N] y;
##       }
##       transformed data {
##         int<lower=0> n = N/K;
##       }
##       parameters {
##         //PARAMS FOR SEPARATE
##         vector[K] mu_S;
##         vector<lower=0>[K] sigma_S;
##       }
##       model {
##         //MODEL FOR SEPARATE (S)
##         mu_S ~ normal(90,50);//Optional weak prior
##         sigma_S ~ scaled_inv_chi_square(K-1,8); //weak prior (for better convergence)
##         y ~ normal(mu_S[x], sigma_S[x]);
##       }
##       generated quantities {
##         real ypred_S[6];
##         vector[N] log_lik;
##
##         //PREDICTION FOR SEPARATE
##         ypred_S = normal_rng(mu_S, sigma_S);
##
##         //likelihood
##         //NOTE: Indexing (per group) is 1,2,3,4,5,6,1,2,3,4,5,6,...
##         for(obs in 1:n) {
##           for(group in 1:K) {
##             log_lik[group+6*(obs-1)] = normal_lpdf(y[group+6*(obs-1)] | mu_S[group], sigma_S[group])
##           }
##         }
```

```
##
##       }
```

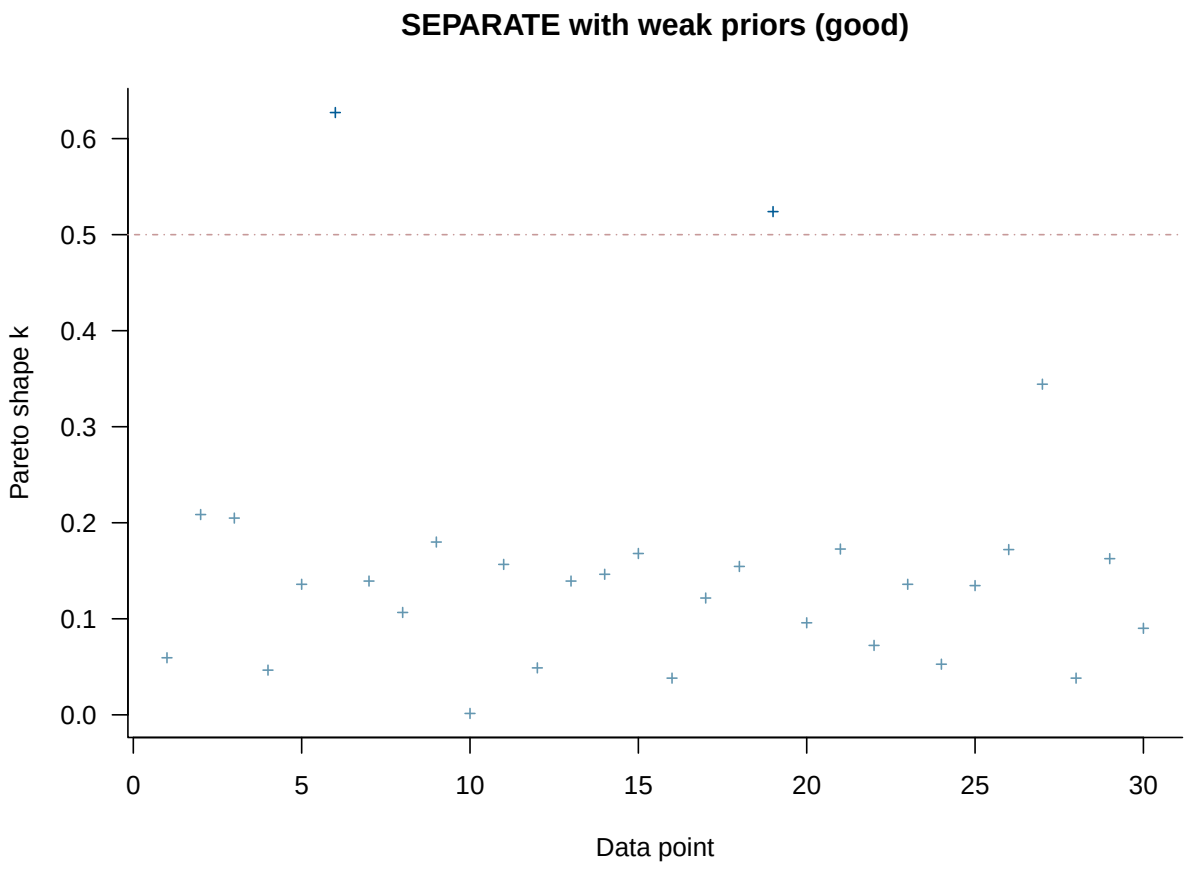
Let's try fitting the new model:

```
fit_separate_priors <- stan(file = stan_model_separate_priors, data = all_data, warmup = 1000, iter = 3
                    control=list(adapt_delta=0.98))
```

```
## DIAGNOSTIC(S) FROM PARSER:
## Info: integer division implicitly rounds to integer. Found int division: N / K
##  Positive values rounded down, negative values rounded up or down in platform-dependent way.
```

And check the results:

```
####################
#SEPARATE_PRIORS
res_separate_priors <- extract_log_lik(fit_separate_priors)
loo_separate_priors <- loo(fit_separate_priors)
plot(loo_separate_priors, main='SEPARATE with weak priors (good)')
```




```
#k_hat
pareto_k_table(loo_separate_priors)
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     28    93.3%   3725
##  (0.5, 0.7]   (ok)        2     6.7%   4473
```

```
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
```

Clearly, the result is sufficient to use PSIS-LOO for model comparison. Here's the return value from loo(...):

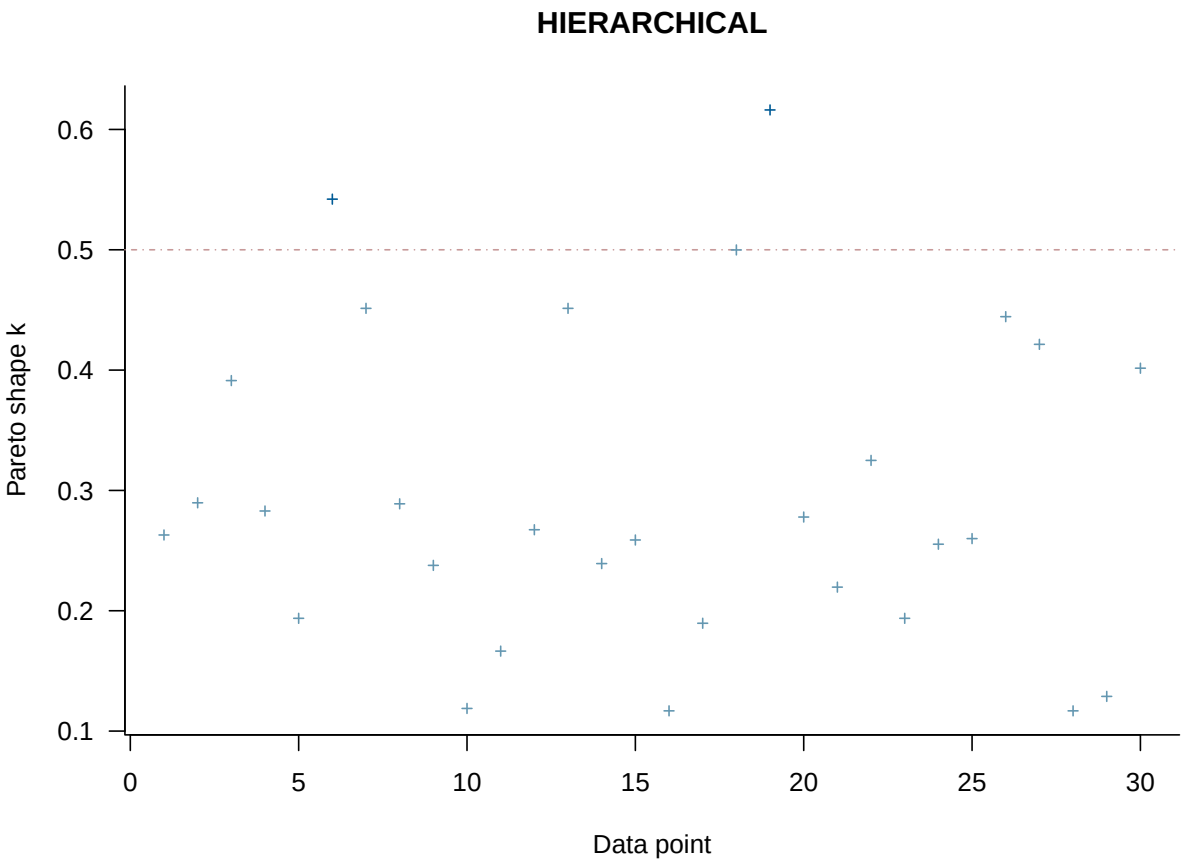
```
loo_separate_priors
```

```
##
## Computed from 8000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -138.8 1.0
## p_loo         4.8 0.1
## looic       277.7 2.0
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     28    93.3%   3725
##  (0.5, 0.7]   (ok)        2     6.7%   4473
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

And finally, the hierarchical model:

HIERARCHICAL LOO

```
####################
#HIERARCH
res_hierarch <- extract_log_lik(fit_hierarch)
loo_hierarch <- loo(fit_hierarch)
plot(loo_hierarch, main='HIERARCHICAL')
```




```
pareto_k_table(loo_hierarch)
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     28    93.3%   3143
##  (0.5, 0.7]   (ok)        2     6.7%   373
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
```

PSIS-LOO estimate for hierarchical model looks valid, regardles of the uniform prior. For better comparison for reference values, I'll stick with these results. Maybe at the end I can also run a more informative hierarchical model - although, I believe the hierarchical model will end-up being better than pooled or separate models, even without informative priors.

Here's the return values for loo(...):

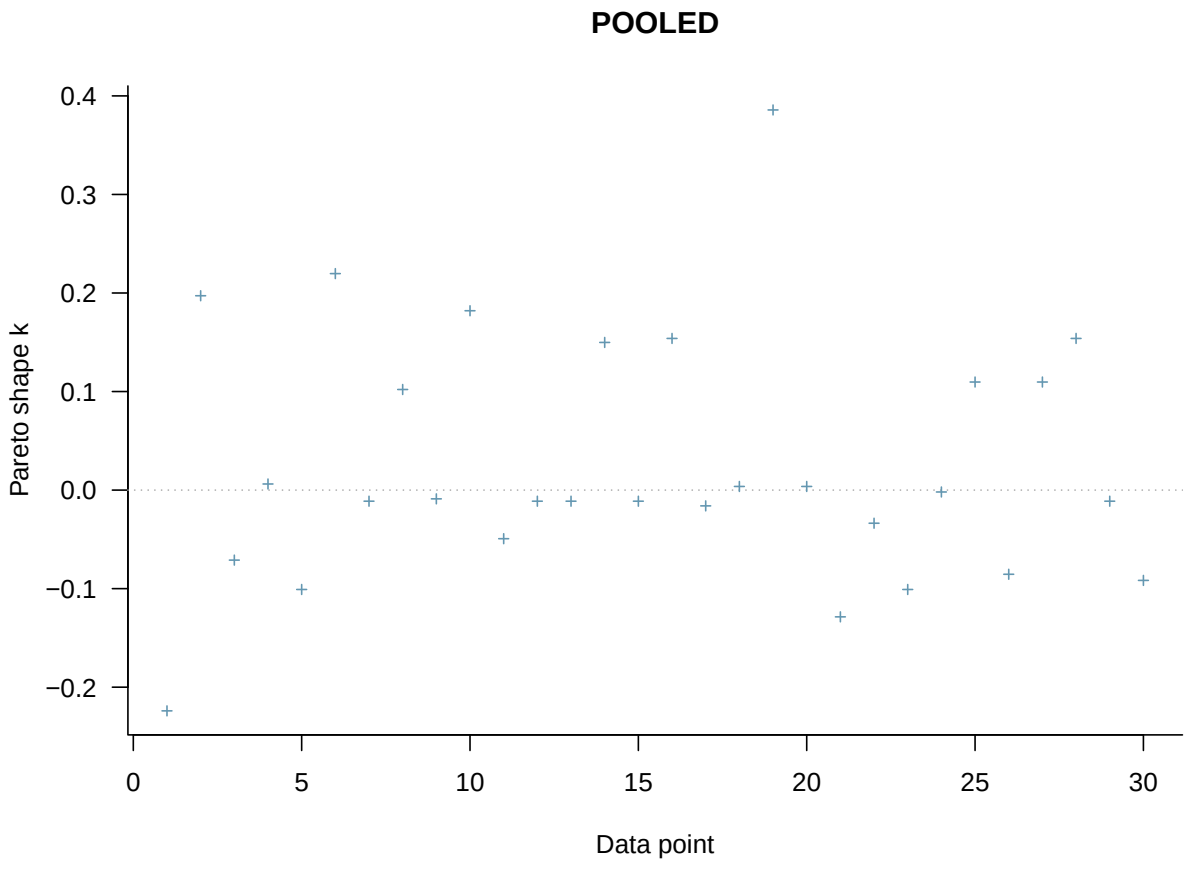
```
loo_hierarch
```

```
##
## Computed from 8000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -126.8 4.2
## p_loo         5.7 1.5
```

```
## looic       253.7 8.4
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     28    93.3%   3143
##  (0.5, 0.7]   (ok)        2     6.7%   373
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

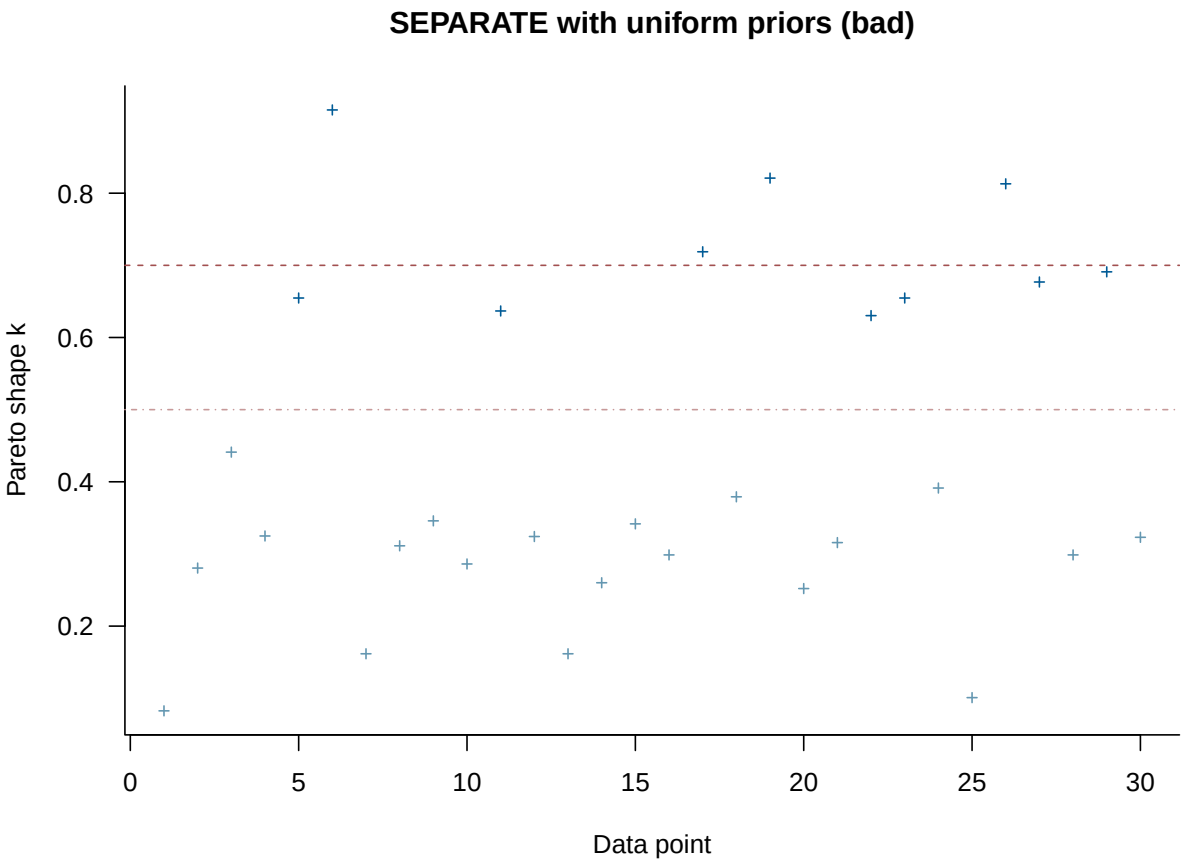
Results more spesifically: NOTE: I'm only plotting k_hat values, because listing a vector of values would be of no use to the grader.

```
# k_hat values for pooled:
plot(loo_pooled, main='POOLED')
```



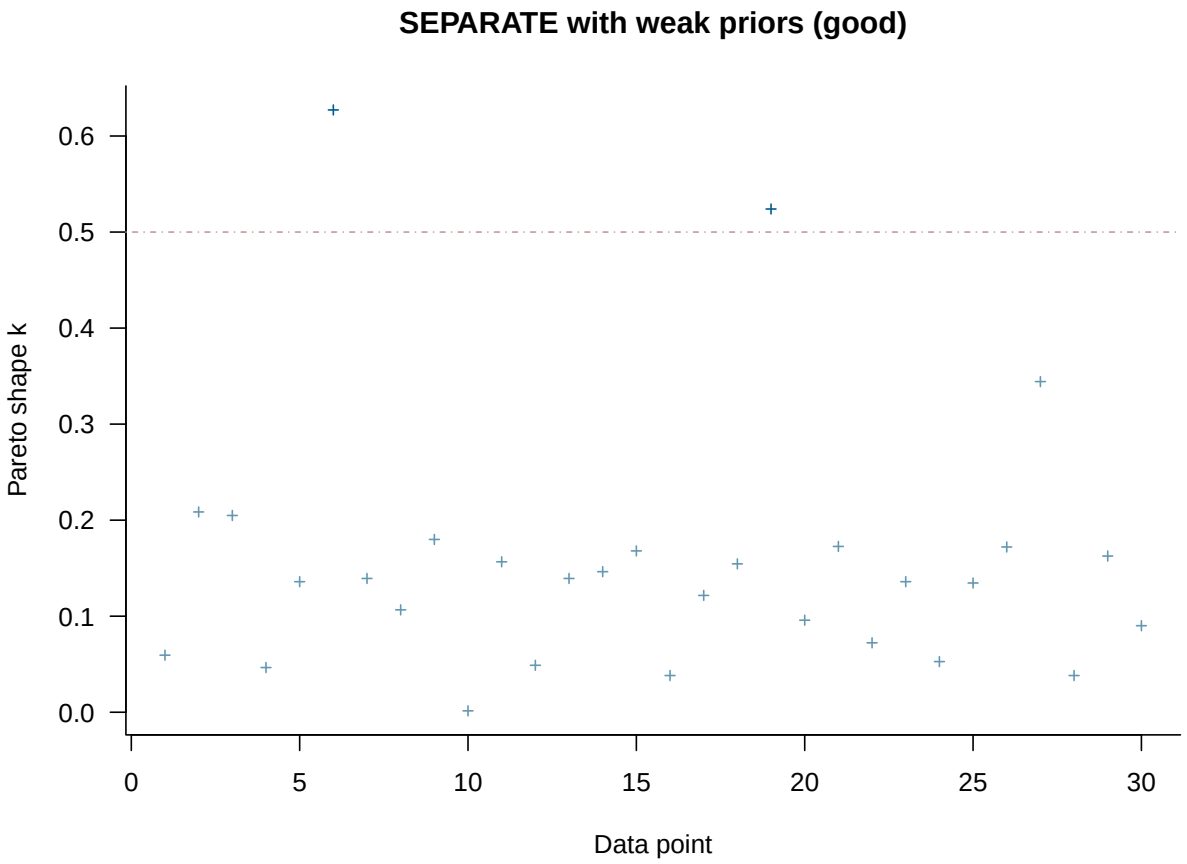
And elpd_loo for pooled is -130.9638247.

```
# k_hat for bad separate case:
plot(loo_separate, main='SEPARATE with uniform priors (bad)')
```



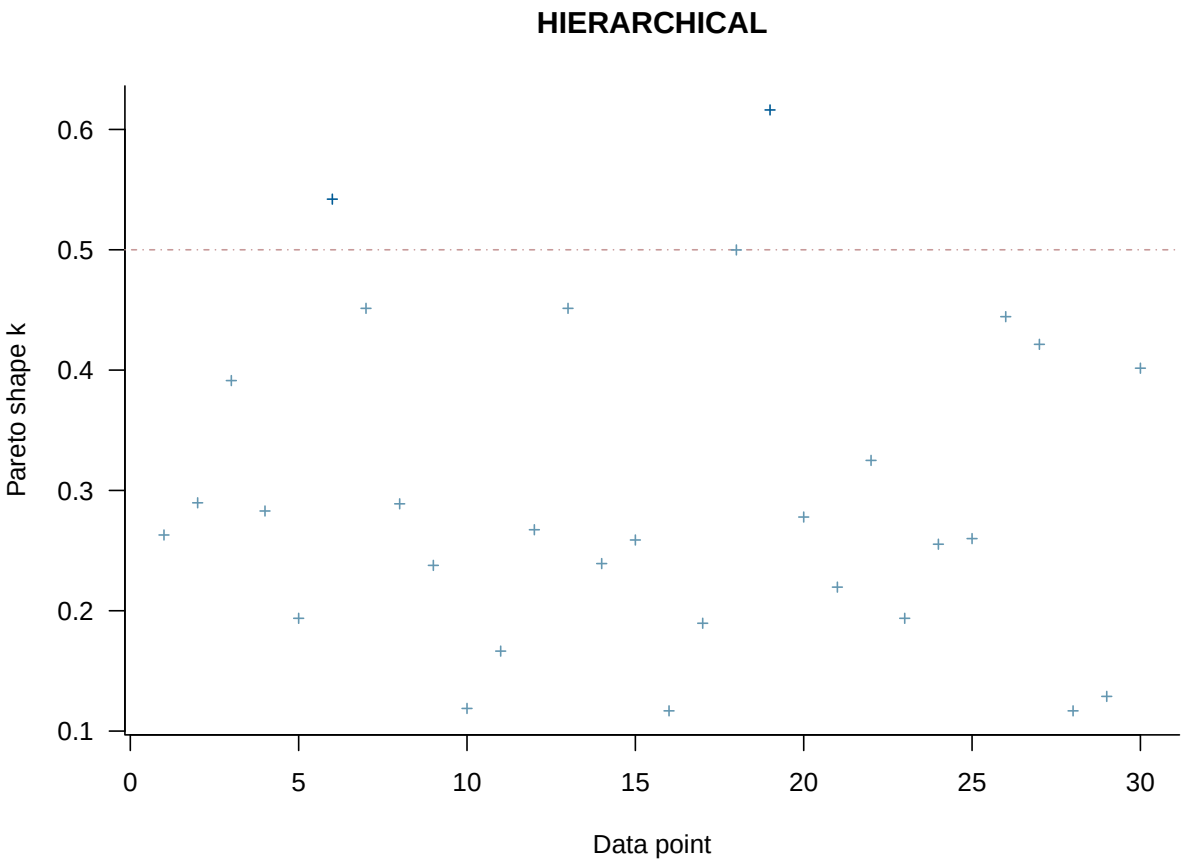
There is no sense in using elpd_loo value for the uniform priored separate models, since it can not be trusted. (you can still see it above where I calculated it first). Instead, here's the result for weakly informative priors case:

```
#k_hat for the good case:
plot(loo_separate_priors, main='SEPARATE with weak priors (good)')
```



And elpd_loo for separate is `-138.8373058`.

```
# k_hat for hierarchical model
plot(loo_hierarch, main = 'HIERARCHICAL' )
```



And elpd_loo for hierarchical case is `-126.8469672`.

Exercise 3

Effective number of parameters could be calculated from the log_lik draws, and the elpd_loo values, but since the loo(...) already did the calculations, I'm going to use them as is. So,

Results:

POOLED

```
loo_pooled$estimates[2,1]
```

```
## [1] 2.018537
```

SEPARATE (weak priors)

```
loo_separate_priors$estimates[2,1]
```

```
## [1] 4.777546
```

HIERARCHICAL

```
loo_hierarch$estimates[2,1]
```

```
## [1] 5.744526
```

And just for comparison, for the badly conditioned separate with uniform priors: **BAD SEPARATE**

```
loo_separate$estimates[2,1]
```

```
## [1] 9.786777
```

Still, as I said earlier, the last result (~10) cannot be trusted (i.e., used).

Exercise 4

Based on the K_hat values, only the first try at separate models, with uniform priors failed to draw loo-cv samples. It had k_hat values above 0.7 — on occasion actually above 1 (unbound variance and mean).

```
pareto_k_table(loo_separate) #BAD
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     20    66.7%   1457
##  (0.5, 0.7]   (ok)        6    20.0%   436
##    (0.7, 1]   (bad)       4    13.3%   177
##    (1, Inf)   (very bad)  0     0.0%   <NA>
```

Other models could be assessed with PSIS-LOO estimates. Here's a quick table to help check the k_hat values:

POOLED

```
#help('pareto-k-diagnostic')
pareto_k_table(loo_pooled)
```

```
##
## All Pareto k estimates are good (k < 0.5).
```

SEPARATE

```
pareto_k_table(loo_separate_priors)
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     28    93.3%   3725
##  (0.5, 0.7]   (ok)        2     6.7%   4473
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
```

HIERARCH

```
pareto_k_table(loo_hierarch)
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     28    93.3%   3143
##  (0.5, 0.7]   (ok)        2     6.7%   373
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
```

So, these PSIS-LOO estimates can be used for model comparison (next).

Exercise 5

Assesment of the differences of the models can be done quickly by checking the elpd differences:

```
loo_compare(loo_pooled, loo_separate_priors, loo_hierarch)
```

```
##        elpd_diff se_diff
## model3   0.0       0.0
## model1  -4.1       2.1
## model2 -12.0       3.3
```

Based on elpd_loo, no surprise, the hierarchical model should be preferred. Pooled model is second, and the separate model least preferred.

NOTE: There are also other considerations when selecting the model. For example, based on low number of effective parameters, the pooled model is likely to generalize better, if there is no additional information on the machine for which one wishes to predict something. For the separate model with uniform prior, there was a relatively large number of eff. predictors, which in that case likely ment that the model was highly succeptible to noise (able to fit noise).