Bayesian Dual GBTM Model Selection

In this vignette we will demonstrate how to use BayesTraj to use Bayesian Model Averaging to estimate a Bayesian Dual GBTM with normal likelihoods. We will use simulated data in order to verify that the estimation routine can select the correct functional forms, recover the true parameters, and to demonstrate how the data should be formatted before calling the estimation routines.

Begin by loading the BayesTraj library:

```
library(BayesTraj)
```

Simulating Data

First, we will simulate data. This will not be necessary in your own projects, but it is useful both for testing the package and for using as a template for formatting your own datasets.

```
N=1000 #number of units
T1=9 #Time periods for Group 1
T2=9 #Time periods for Group 2
pi1=c(0.5,0.2,0.3) #Group 1 membership probabilities
#Transition Matrix
pi1_2=matrix(c(0.3,0.3,0.4,
               0.2,0.5,0.3,
               0.7,0.2,0.1),
             nrow=3,ncol=3,byrow=TRUE)
K1 = length(pi1) #Number of groups in series 1
K2 = dim(pi1_2)[2] #Number of groups in series 2
#Coefficients for Series 1
beta1=matrix(c(100,10,-0.5,0.1,
               80,-1,0.5,0,
               120,20,-2,0),nrow=3,ncol=4,byrow=TRUE)
#Coefficients for Series 2
beta2=matrix(c(90,0.4,0,0,
               50,1,-0.5,0,
               100,-30,3,-0.3),nrow=3,ncol=4,byrow=TRUE)
sigma1=16 #standard deviation of Series 1 outcomes
sigma2=36 #standard deviation of Series 2 outcomes

set.seed(1)
data = gen_data_dual(N=N,
                T1=T1,
                T2=T2,
                pi1=pi1,
                pi2=pi1_2,
                beta1=beta1,
                beta2=beta2,
                sigma1=sigma1,
                sigma2=sigma2,
                poly = 3) #degree of polynomial
```

In this example we have simulated data for 1000 paired-units with 9 time periods each, for a total of 18000 observations. While we have set each unit to have observations in 9 time periods, the estmation function below allows for the number of observations to vary. We have chosen the Group 1 membership probabilities to be 50%, 20%, and 30%. `pi1_2` is the transition matrix. `pi1_2[i,j]` represents the probability that the second pair member is in Group j conditional on the first pair member being in Group i. From this, the `gen_data_dual` function can infer that there should be three groups for both series.

Each row of the `beta` matrices defines the trajectory coefficients for the respective groups. For example, the expected value at time t in Series 1 Group 1 is $100 + 10t - 0.5t^2 + 0.1t^3$. Sigma1 and sigma2 define the standard deviation of the outcomes.

When calling the `gen_data_dual` function, we also specify `poly=2` in order to tell model to use a second-degree polynomial for time. If there are more non-intercept columns of `beta` than `poly`, `gen_data_dual` will generate random covariates corresponding to the remaining columns. In general, the last `poly` columns of the `beta` matrices correspond to the polynomial coefficients.

Please note that we have selected `beta` coefficients corresponding to various polynomial degress. Of the six groups in the data, 3 have third-degree polynomials, two have second-degree polynomials, and one has a first-degree polynomial.

Now let's take a look at the generated data. We can unpack the individual attributes from the data object.

```
X1=data$X1
X2=data$X2
y1=data$Y1
y2=data$Y2
```

While we will restrict our discussion to the first series, everything applies to the second series as well. The first 18 rows of `X1` are:

```
print(head(X1,18))
#>       [,1] [,2] [,3] [,4]
#>  [1,]    1    1    1    1
#>  [2,]    1    2    4    8
#>  [3,]    1    3    9   27
#>  [4,]    1    4   16   64
#>  [5,]    1    5   25  125
#>  [6,]    1    6   36  216
#>  [7,]    1    7   49  343
#>  [8,]    1    8   64  512
#>  [9,]    1    9   81  729
#> [10,]    2    1    1    1
#> [11,]    2    2    4    8
#> [12,]    2    3    9   27
#> [13,]    2    4   16   64
#> [14,]    2    5   25  125
#> [15,]    2    6   36  216
#> [16,]    2    7   49  343
#> [17,]    2    8   64  512
#> [18,]    2    9   81  729
```

The first column identifies the unit. For example, the first 9 rows correspond to unit 1, the second 9 rows correspond to unit 2, and so forth. The second column is the time variable. Rows 1 and 10 correspond to time 1, rows 2 and 11 correspond to time 2, and so forth. Similarly, the third column is the square of the time column and the fourth column is the cube.

Now we take a look at `y1`. These are the outcomes. `y1[1]` corresponds to the outcome for unit 1 at time 1. `y1[2]` corresponds to the outcome for unit 1 at time 2, and so forth. The values of `y1` must correspond with

the rows of `X1`. Therefore `X1` and `y1` should have the same length.

```
print(head(y1,18))
#>  [1] 114.1399 123.2477 124.7169 139.2429 150.2776 156.9494 183.0434
#>  [8] 191.5506 217.4130 113.5926 116.6365 127.3345 131.9123 144.1961
#> [15] 165.0036 179.1018 196.8343 217.0639
```

Estimating the model

We now turn our attention toward estimating the model. We can do this by calling the `dualtrajMS` function. This function uses the following weakly-informative hyperparameters: a uniform prior on group membership probabilities, Jeffery's prior on the variances, and a unit-information g-prior on the regression coefficients.

```
iter = 5000
thin = 1
model = dualtrajMS(X1=X1, #data matrix Series 1
                   X2=X2, #data matrix Series 2
                   y1=y1, #outcomes Series 1
                   y2=y2, #outcomes Series 2
                   K1=K1, #number of groups Series 1
                   K2=K2, #number of groups Series 2
                   time_index=2, #column of X corresponding to time
                   iterations=iter, #number of iterations
                   thin=thin, #thinning
                   dispIter=1000) #Print a message every 1000 iterations
#> [1] 1000
#> [1] 2000
#> [1] 3000
#> [1] 4000
#> [1] 5000
```

First, let's clarify the model specification. `dualtrajMS` will sample the polynomial degree in the MCMC samples, rather than independently choose whether or not to include each covariate. For example, if 3 is sampled then we will estimate a cubic polynomial, if 2 is sampled then we will estimate a squared polynomial, and so forth. Users wishing to average over more complicated functions than polynomials, or to estimate models in which high order polynomials can be included even if a lower polynomial was selected out, will need make corresponding edits to the `dualtrajMS` function.

Here we run the model for 5000 MCMC iterations. In practice, more iterations may be desirable to ensure the posterior results are valid. Setting the `thin` parameter to 1 tells us to keep every sample. We can set `thin=10`, for example, to only keep 1 out of every 10 samples. Thinning is not necessary unless your computer has memory limitations. We also set `dispIter=1000` to tell the program to send us a message every 1000 MCMC iterations. This will help us monitor the progress of the program.

The only argument we have not touched on yet is `time_index`. This parameter specified which column of `X` corresponds to the time variable. If the data does not contain any covariates, this should be the second column of `X`. If, for example, we were using a dataset with additional covariates in columns 2 and 3, time in column 4, and time-squared in column 5, we would set `time_index=4`.

Analyzing the Model

The model object contains the MCMC samples for each of the model's parameters. We can access the MCMC samples as follows, where each row represent an iteration of the MCMC:

```
head(model$beta1[[1]]) #Series 1 group 1's coefficients
#>           [,1]      [,2]       [,3]      [,4]
#> [1,] 105.56339  7.755373  0.0000000 0.0000000
```

```
#> [2,] 106.50248  8.186797  0.0000000 0.0000000
#> [3,] 111.02443  9.144762  0.0000000 0.0000000
#> [4,]  99.92845 14.638564 -1.3810026 0.1205504
#> [5,]  99.95446 10.058606 -0.5239753 0.1020357
#> [6,]  99.91526 10.162504 -0.5468373 0.1033624
head(model$beta1[[2]]) #Series 1 group 2's coefficients
#>          [,1]      [,2]       [,3] [,4]
#> [1,] 102.5045  8.071105  0.0000000    0
#> [2,] 103.5695  8.051928  0.0000000    0
#> [3,] 114.5299  8.371167  0.0000000    0
#> [4,] 113.4299 13.089365 -0.6768584    0
#> [5,] 120.1869 19.831635 -1.9805797    0
#> [6,] 119.8036 20.135981 -2.0157279    0
head(model$beta1[[3]]) #Series 1 group 3's coefficients
#>           [,1]       [,2]       [,3] [,4]
#> [1,] 103.09077  8.4253057 -0.0614617    0
#> [2,] 102.87467  6.1443306  0.0000000    0
#> [3,]  85.38751 -0.5091793  0.4540651    0
#> [4,]  79.80471 -1.1125931  0.5202069    0
#> [5,]  79.64531 -1.0812855  0.5236552    0
#> [6,]  79.72681 -1.0606995  0.5147198    0
head(model$sigma1) #Series 1 variance - NOT THE STANDARD DEVIATION
#>           [,1]       [,2]       [,3]
#> [1,] 992.79938 1001.40668 1084.06381
#> [2,] 855.07114 1003.53018 1335.46090
#> [3,] 378.13753  425.15170  428.80401
#> [4,] 247.00031  417.48545   16.12653
#> [5,]  16.15578   21.28314   16.69968
#> [6,]  16.63348   16.77541   15.64830
model$c1[1:6,1:10] #Series 1 unit-level group memberships
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#> [1,]    2    2    3    2    2    3    1    2    2     2
#> [2,]    2    2    2    3    2    3    2    2    2     2
#> [3,]    1    1    2    3    1    3    3    1    2     1
#> [4,]    2    1    2    3    1    3    3    2    2     2
#> [5,]    1    1    2    3    1    3    3    2    2     1
#> [6,]    1    1    2    3    1    3    3    2    2     1
head(model$pi1) #Series 1  group-membership probabilities
#>           [,1]      [,2]      [,3]
#> [1,] 0.3461449 0.4855474 0.1683076
#> [2,] 0.3536529 0.4929497 0.1533974
#> [3,] 0.4071274 0.3555797 0.2372929
#> [4,] 0.4220945 0.3924430 0.1854625
#> [5,] 0.5383237 0.2755262 0.1861502
#> [6,] 0.5262326 0.2949097 0.1788576
head(model$pi2) #Series 2  group-membership probabilities
#>             [,1]      [,2]      [,3]
#> [1,] 0.492469476 0.1499625 0.3575681
#> [2,] 0.503553683 0.1352396 0.3612067
#> [3,] 0.290409966 0.1122119 0.5973781
#> [4,] 0.008133891 0.3000159 0.6918502
#> [5,] 0.304073537 0.3125256 0.3834009
#> [6,] 0.327765409 0.2865403 0.3856943
```

```
model$pi1_2[1,,] #Transition probabilities from Series 1 Group 1.
#>           [,1]        [,2]       [,3]
#> [1,] 0.3300167 0.008194046 0.66178925
#> [2,] 0.7290138 0.181563670 0.08942251
#> [3,] 0.1441710 0.350360607 0.50546842
model$pi12[1,,] #Joint probability of both Series group memberships
#>            [,1]        [,2]       [,3]
#> [1,] 0.11423360 0.002836327 0.22907498
#> [2,] 0.35397080 0.088157776 0.04341887
#> [3,] 0.02426508 0.058968368 0.08507420
```

A conveniant way to summarize the posterior is with the `summary_dual_MS` function:

```
burn = 0.9
summary = summary_dual_MS(model,X1,X2,y1,y2,burn)
```

The `burn` parameter specifies the fraction of draws to keep. In this example, we keep the last 90% of MCMC samples. The first 10% are discarded as the burn-in period.

We can now print out a posterior summary to obtain the posterior mean, standard deviation, and 95% credible interval, as follows:

```
print(summary$estimates)
#>                 Estimate Standard Deviation          2.5%         50%
#> beta1_1[1]  9.993229e+01       0.3849188074  99.186655738  99.9382766
#> beta1_1[2]  1.003222e+01       0.3136473976   9.423656938  10.0260182
#> beta1_1[3] -5.079515e-01       0.0708615726  -0.647367984  -0.5073678
#> beta1_1[4]  1.007345e-01       0.0046756956   0.091769012   0.1006844
#> sigma1_1    4.038240e+00       0.0422121607   3.954711949   4.0377701
#> beta1_2[1]  1.198396e+02       0.3181681290 119.212895946 119.8387246
#> beta1_2[2]  2.006374e+01       0.1543118623  19.751102681  20.0670543
#> beta1_2[3] -2.005846e+00       0.0194428061  -2.035115692  -2.0066300
#> beta1_2[4] -4.937830e-05       0.0008989989   0.000000000   0.0000000
#> sigma1_2    4.082878e+00       0.0579422375   3.973887961   4.0816279
#> beta1_3[1]  7.974191e+01       0.3938774162  79.033186396  79.7280328
#> beta1_3[2] -1.058160e+00       0.2316240301  -1.548475016  -1.0364262
#> beta1_3[3]  5.176239e-01       0.0415415170   0.478403158   0.5118625
#> beta1_3[4] -4.250873e-04       0.0025402693  -0.008746408   0.0000000
#> sigma1_3    3.977847e+00       0.0666261471   3.848889668   3.9766650
#> pi1[1]      5.191275e+01       1.5808346741  48.736589700  51.9199888
#> pi1[2]      2.824179e+01       1.4134860192  25.524880807  28.2388691
#> pi1[3]      1.984546e+01       1.2512353815  17.491982053  19.8204426
#> beta2_1[1]  4.924984e+01       0.4639150933  48.406663599  49.2350201
#> beta2_1[2]  1.406983e+00       0.2575625638   0.931772597   1.4244367
#> beta2_1[3] -5.377047e-01       0.0439341407  -0.580819018  -0.5428144
#> beta2_1[4] -3.735729e-04       0.0026235051  -0.004331308   0.0000000
#> sigma2_1    5.923119e+00       0.0798612827   5.761952858   5.9236856
#> beta2_2[1]  9.955316e+01       0.7457653550  98.049968230  99.5555515
#> beta2_2[2] -2.991691e+01       0.6154421017 -31.125834531 -29.9236260
#> beta2_2[3]  3.050123e+00       0.1391346883   2.774152605   3.0512312
#> beta2_2[4] -3.066100e-01       0.0091447660  -0.324399527  -0.3066759
#> sigma2_2    5.955801e+00       0.0811040939   5.800948050   5.9555145
#> beta2_3[1]  8.993266e+01       0.2291611808  89.504879087  89.9279549
#> beta2_3[2]  3.949163e-01       0.0587923482   0.303877663   0.3993582
#> beta2_3[3]  4.912008e-04       0.0059825562   0.000000000   0.0000000
```

```
#> beta2_3[4] -1.717224e-06       0.0002978377   0.000000000   0.0000000
#> sigma2_3    5.933092e+00       0.0695942401   5.801221320   5.9322400
#> pi2[1]      3.091413e+01       1.4717898636  28.091235960  30.9295710
#> pi2[2]      3.033974e+01       1.4602270800  27.474969183  30.3422094
#> pi2[3]      3.874613e+01       1.5413331487  35.826250585  38.7333135
#> 1->2,1->1   2.719338e+01       1.9243582145  23.522164757  27.1677361
#> 1->2,1->2   4.240965e+01       2.1707437686  38.136891539  42.4000042
#> 1->2,1->3   3.039698e+01       2.0335872268  26.538216801  30.3824593
#> 1->2,2->1   1.997762e+01       2.3738642145  15.636829426  19.8790751
#> 1->2,2->2   1.264244e+01       1.9735464318   9.130860065  12.5456241
#> 1->2,2->3   6.737994e+01       2.7672304715  61.904813286  67.4734118
#> 1->2,3->1   5.621397e+01       3.4833433041  49.290876457  56.2322322
#> 1->2,3->2   2.394348e+01       3.0249683860  18.295823338  23.8416521
#> 1->2,3->3   1.984255e+01       2.7906553295  14.581361741  19.7529917
#> 2->1,1->1   4.566555e+01       2.7976550846  40.100679802  45.6919049
#> 2->1,1->2   1.824625e+01       2.1570322163  14.358828574  18.1626955
#> 2->1,1->3   3.608821e+01       2.7070820264  30.958915848  36.0259931
#> 2->1,2->1   7.256638e+01       2.5811321597  67.370024560  72.6258559
#> 2->1,2->2   1.176939e+01       1.8534110016   8.478420249  11.6723898
#> 2->1,2->3   1.566423e+01       2.0958387407  11.753999614  15.5872955
#> 2->1,3->1   4.072545e+01       2.5027981897  35.953477676  40.6833080
#> 2->1,3->2   4.911311e+01       2.5121061952  44.139623225  49.1215724
#> 2->1,3->3   1.016144e+01       1.5036577308   7.379188655  10.0839840
#> 1,1         1.411653e+01       1.0840498783  12.001274281  14.1091614
#> 1,2         2.201617e+01       1.3168395382  19.506445305  22.0053546
#> 1,3         1.577965e+01       1.1592859217  13.601344720  15.7540672
#> 2,1         5.641490e+00       0.7261760456   4.359874672   5.6046752
#> 2,2         3.570922e+00       0.5900868463   2.523344538   3.5418983
#> 2,3         1.902926e+01       1.2331017036  16.631013304  19.0121864
#> 3,1         1.115664e+01       0.9917151970   9.301033627  11.1231875
#> 3,2         4.752357e+00       0.6765817212   3.528740571   4.7255372
#> 3,3         3.936986e+00       0.6028858106   2.832030284   3.8995216
#>                    97.5% Inclusion Prob.
#> beta1_1[1]  1.006930e+02          1.0000
#> beta1_1[2]  1.064579e+01          1.0000
#> beta1_1[3] -3.721845e-01          0.9994
#> beta1_1[4]  1.100415e-01          0.9994
#> sigma1_1    4.121489e+00              NA
#> beta1_2[1]  1.204839e+02          1.0000
#> beta1_2[2]  2.035241e+01          1.0000
#> beta1_2[3] -1.974639e+00          0.9994
#> beta1_2[4]  0.000000e+00          0.0222
#> sigma1_2    4.197962e+00              NA
#> beta1_3[1]  8.054309e+01          1.0000
#> beta1_3[2] -7.093075e-01          1.0000
#> beta1_3[3]  6.417338e-01          0.9998
#> beta1_3[4]  0.000000e+00          0.0474
#> sigma1_3    4.110034e+00              NA
#> pi1[1]      5.502025e+01              NA
#> pi1[2]      3.100684e+01              NA
#> pi1[3]      2.240092e+01              NA
#> beta2_1[1]  5.017672e+01          1.0000
#> beta2_1[2]  1.813398e+00          1.0000
```

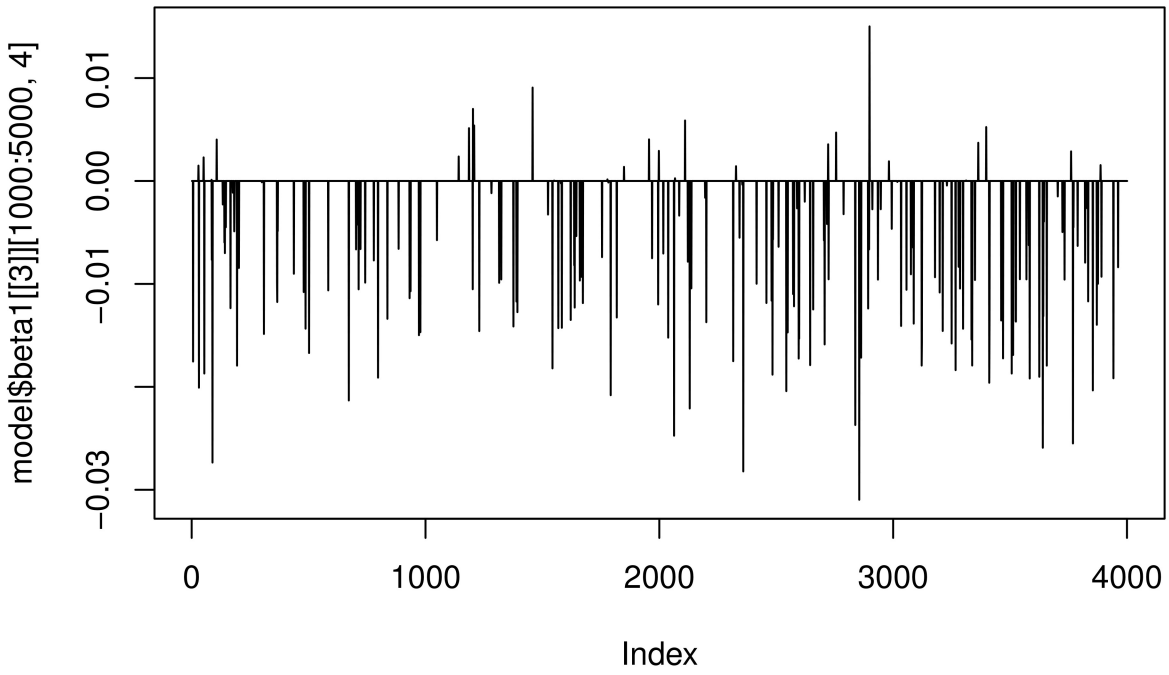
```
#> beta2_1[3] -4.762394e-01          1.0000
#> beta2_1[4]  0.000000e+00          0.0356
#> sigma2_1    6.076935e+00              NA
#> beta2_2[1]  1.010091e+02          1.0000
#> beta2_2[2] -2.869771e+01          1.0000
#> beta2_2[3]  3.317657e+00          1.0000
#> beta2_2[4] -2.881500e-01          0.9996
#> sigma2_2    6.121289e+00              NA
#> beta2_3[1]  9.038732e+01          1.0000
#> beta2_3[2]  4.744573e-01          1.0000
#> beta2_3[3]  2.253819e-04          0.0296
#> beta2_3[4]  0.000000e+00          0.0008
#> sigma2_3    6.069870e+00              NA
#> pi2[1]      3.376711e+01              NA
#> pi2[2]      3.329302e+01              NA
#> pi2[3]      4.184821e+01              NA
#> 1->2,1->1   3.109538e+01              NA
#> 1->2,1->2   4.658389e+01              NA
#> 1->2,1->3   3.444202e+01              NA
#> 1->2,2->1   2.474966e+01              NA
#> 1->2,2->2   1.687495e+01              NA
#> 1->2,2->3   7.256067e+01              NA
#> 1->2,3->1   6.286873e+01              NA
#> 1->2,3->2   3.029508e+01              NA
#> 1->2,3->3   2.557834e+01              NA
#> 2->1,1->1   5.110973e+01              NA
#> 2->1,1->2   2.279810e+01              NA
#> 2->1,1->3   4.159844e+01              NA
#> 2->1,2->1   7.738673e+01              NA
#> 2->1,2->2   1.572901e+01              NA
#> 2->1,2->3   2.010423e+01              NA
#> 2->1,3->1   4.574489e+01              NA
#> 2->1,3->2   5.401148e+01              NA
#> 2->1,3->3   1.326508e+01              NA
#> 1,1         1.621852e+01              NA
#> 1,2         2.450925e+01              NA
#> 1,3         1.814053e+01              NA
#> 2,1         7.211636e+00              NA
#> 2,2         4.844930e+00              NA
#> 2,3         2.151103e+01              NA
#> 3,1         1.321187e+01              NA
#> 3,2         6.191092e+00              NA
#> 3,3         5.168354e+00              NA
```

The notation for the transitions and joint probabilities requires some understanding. 1->2,i->j is $\Pr(c_2 = j|c_1 = i)$ and 2->1,i->j is $\Pr(c_1 = j|c_2 = i)$. The final columns i, j denote the joint probabilities $\Pr(c_1 = i \cap c_2 = j)$.

MCMC samples for variables selected out of the model.

The MCMC samples can be plotted to see how the variable selection works. In the example below, we see that all but about 5% of the draws for the cubic coefficient in Series 1 Group 3 are set ot zero, effectively selecting this parameter out of the model.

```
plot(model$beta1[[3]][1000:5000,4],type='l')
```



Checking for Label Switching and Local Modes

One issue with GBTMs is the tendency for estimation routines to find a local mode which is not globally optimal. This is a problem for GBTMs estimated using maximum likelihood as well. To increase the probability that we are in a global optimum rather than a local optimum, we often run the Gibbs sampler using several seeds and print out the likelihood at the posterior mean:

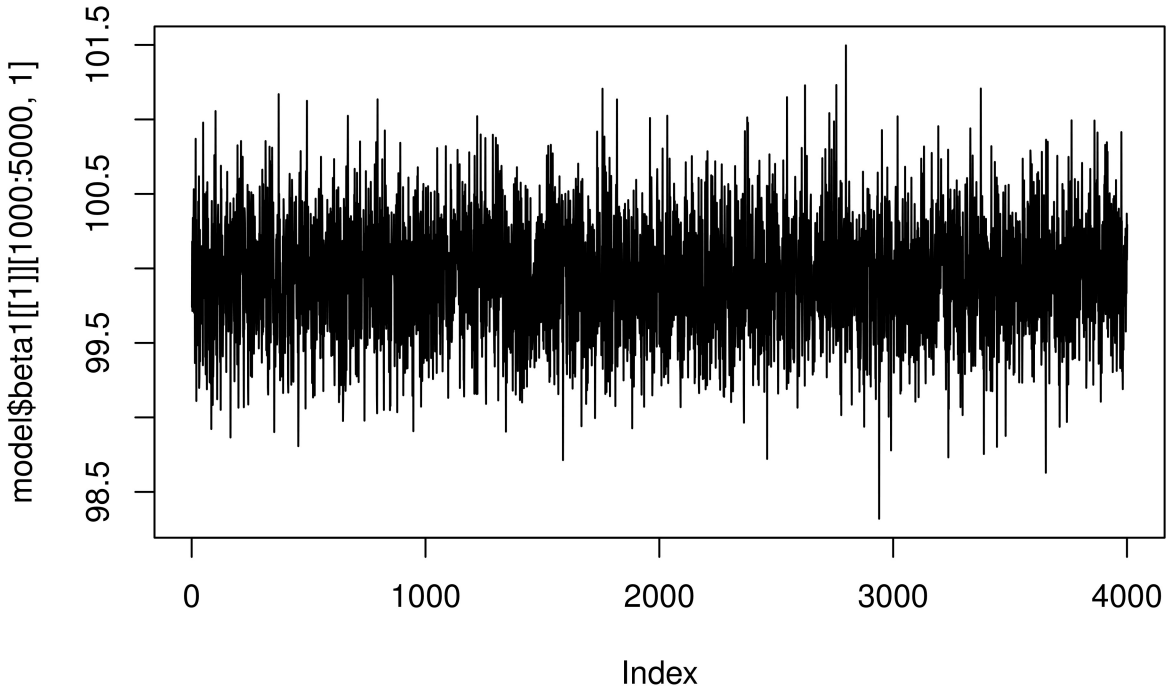
```
print(summary$log.likelihood)
#> [1] -56140.15
```

We then use the seed which maximize the likelihood. This solution has no optimality guarantees, but we have found that we can often reach better optimas this way than other existing packages using maximum likelihood.

The main drawback of the Bayesian approach is the tendency for label-switching and mode-switching. In the label-switching problem, the group labels switch in the middle of the algorithm. As a consequence, the group labeled "1" for the first 1000 draws may be labeled "2" in the second 1000 draws and vice versa. This would render and posterior summary of these coefficients meaningless. In our experience, label switching has not been a problem. However, switching between local-modes during the sampling process has occasionally been an issue.

There is no consensus for the best way to deal with label and mode switching. Either problem can be easily observed by plotting the draws sequentially and checking for sudden and sustained breaks in the trend. For example, the plot below looks consistent throughout the post-burn-in samples:

```r
plot(model$beta1[[1]][1000:5000,1],type='l')
```



This indicates that neither label-switching nor mode-siwtching occured.

If we do observe a sudden break, there are multiple possible solutions. From our experience, we usually find that re-estimating the model using a different seed will solve the problem with least amount of effort.