Functional ANOVA Example

November 09, 2018

Simulate Three-Group One-Way Functional ANOVA Data

Three-group one-way functional ANOVA data were simulated using methods described in the paper "Fast Function-on-Scalar Regression with Penalized Basis Expansions" from Reiss et al:

$$
\begin{aligned}
y_i(t) &= \mu(t) + \beta_{gp(i)}(t) + \epsilon_i(t) \\
\mu(t) &= 0.4 \arctan(10x - 5) + 0.6 \\
\beta_1(t) &= -0.5e^{-10t} - 0.04\sin(8t) - 0.3t + 0.5 \\
\beta_2(t) &= -(t - 0.5)^2 - 0.15\sin(13t) \\
\beta_3(t) &= -\beta_1(t) - \beta_2(t)
\end{aligned}
$$

where $t = m/200$ for $m = 0, \ldots, 200$ and the error functions, $\epsilon_i(t)$, were simulated from a mean-zero Gaussian process with coveriance $V(s,t) = \sigma_1 0.15^{|s-t|} + \sigma_2 \delta_{st}$ where $\delta_{st} = 1$ if $s = t$ and 0 otherwise. For the simulations, N=10 curves were simulated for each of the 3 groups, $\sigma_1 = 0.15$, and $\sigma_2 = 0.05$. So, we have 6030 observations from 30 curves with 10 curves per group.

```
################################
## Create the "True" Dataset ##
################################

## Model
## y_i(t) = mu(t) + b_gp(i)(t) + e_i(t) for t in [0,1]

## Values
t = 0:200/200     ## Evaluation time points
g = 3             ## Number of groups
N = 10            ## Number of curves per group, N = 10 or 60

## Coefficients from the paper
mu.t = 0.4*atan(10*t - 5) + 0.6
b1.t = -0.5*exp(-10*t) - 0.04*sin(8*t) - 0.3*t + 0.5
b2.t = -(t - 0.5)^2 - 0.15*sin(13*t)
b3.t = -b1.t - b2.t

## Mean functions by group
mean_1 = mu.t + b1.t
mean_2 = mu.t + b2.t
mean_3 = mu.t + b3.t

## Create the "true" dataset
true.data = data.frame(Y_True = c(mean_1, mean_2, mean_3),
                       Time = rep(t, 3),
                       Group = as.factor(c(rep(1, length(t)),
                                           rep(2, length(t)),
                                           rep(3, length(t)))) )
```

Note that the X variable, Time, was centered and scaled.

```
############################
## Generate Simulated Data ##
############################

## Create a stacked dataframe for the simulated data
dat = data.frame(Y_True = c(rep(mean_1, N),
                            rep(mean_2, N),
                            rep(mean_3, N)),
                 Time = rep(t, N*g),
                 Group = as.factor( c(rep(1, N*length(t)),
                                      rep(2, N*length(t)),
                                      rep(3, N*length(t))) ),
                 ID = factor( rep(1:(N*g),
                                  length(t))[ order(rep(1:(N*g), length(t))) ] ) )
rm(mean_1, mean_2, mean_3)

## Define values
sigma_1 = 0.15    ## Simulations used 0.05 and 0.15
sigma_2 = 0.05    ## 0.05 for all simulations

## Calculate the covariance matrix
grid = expand.grid(t, t)
cov.matrix = apply(X = grid, MARGIN = 1,
                   FUN = function(x){ ((sigma_1^2)*0.15^(abs(x[1] - x[2]) )) +
                       (sigma_2^2)*as.numeric(x[1] == x[2]) } )
cov.matrix = matrix(data = cov.matrix, ncol = length(t), nrow = length(t))
rm(grid)

## Generate errors
seed = 87654321
set.seed(seed)
errors = mvrnorm(n = N*g, mu = rep(0, length(t)), Sigma = cov.matrix)

## Add to the dataframe
dat$Errors = c( errors[1:(N*g),] )

## Calculate simulated Y's
dat$Y_Sim = (dat$Y_True + dat$Errors)
rm(cov.matrix, errors, sigma_1, sigma_2)

## Pre-smooth data
dat$Y_Smooth = rep(NA, N*g*length(t))
for(i in 1:(N*g)){
  start = (i-1)*length(t) + 1
  end = i*length(t)
  dat$Y_Smooth[start:end] = with(dat,
                                 smooth.spline(Time[start:end], Y_Sim[start:end],
                                               nknots = 20))$y
}
rm(i, start, end, t, mu.t)

## Center and scale the time variable
dat$Time_Standardized = scale(dat$Time)
```

```
## Calculate higher-order time terms
dat$Time_Stan_2 = dat$Time_Standardized^2
dat$Time_Stan_3 = dat$Time_Standardized^3
dat$Time_Stan_4 = dat$Time_Standardized^4
dat$Time_Stan_5 = dat$Time_Standardized^5
```

Figure 1: The true effect functions along with the simulated data.

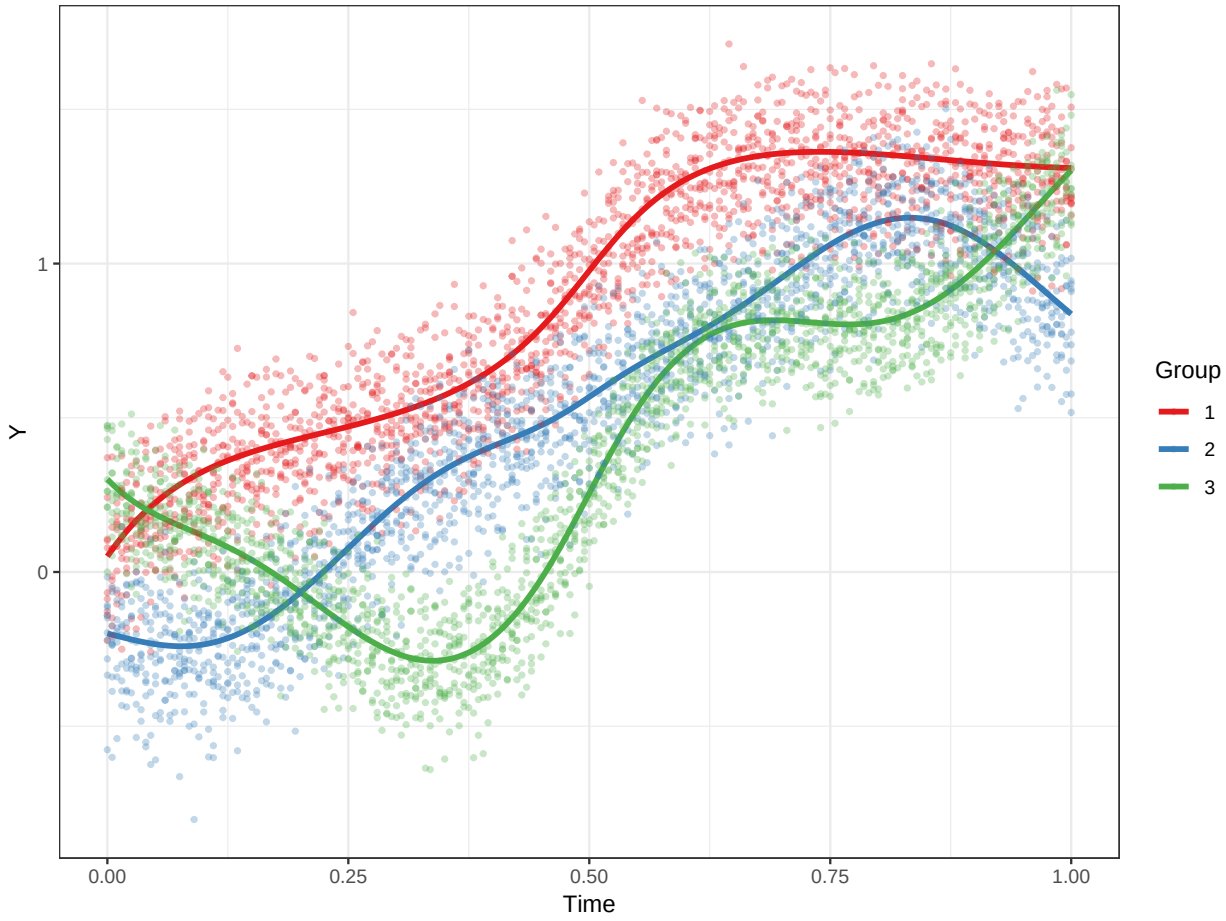


Figure 2: The smoothed subject-specific curves from the simulated data.

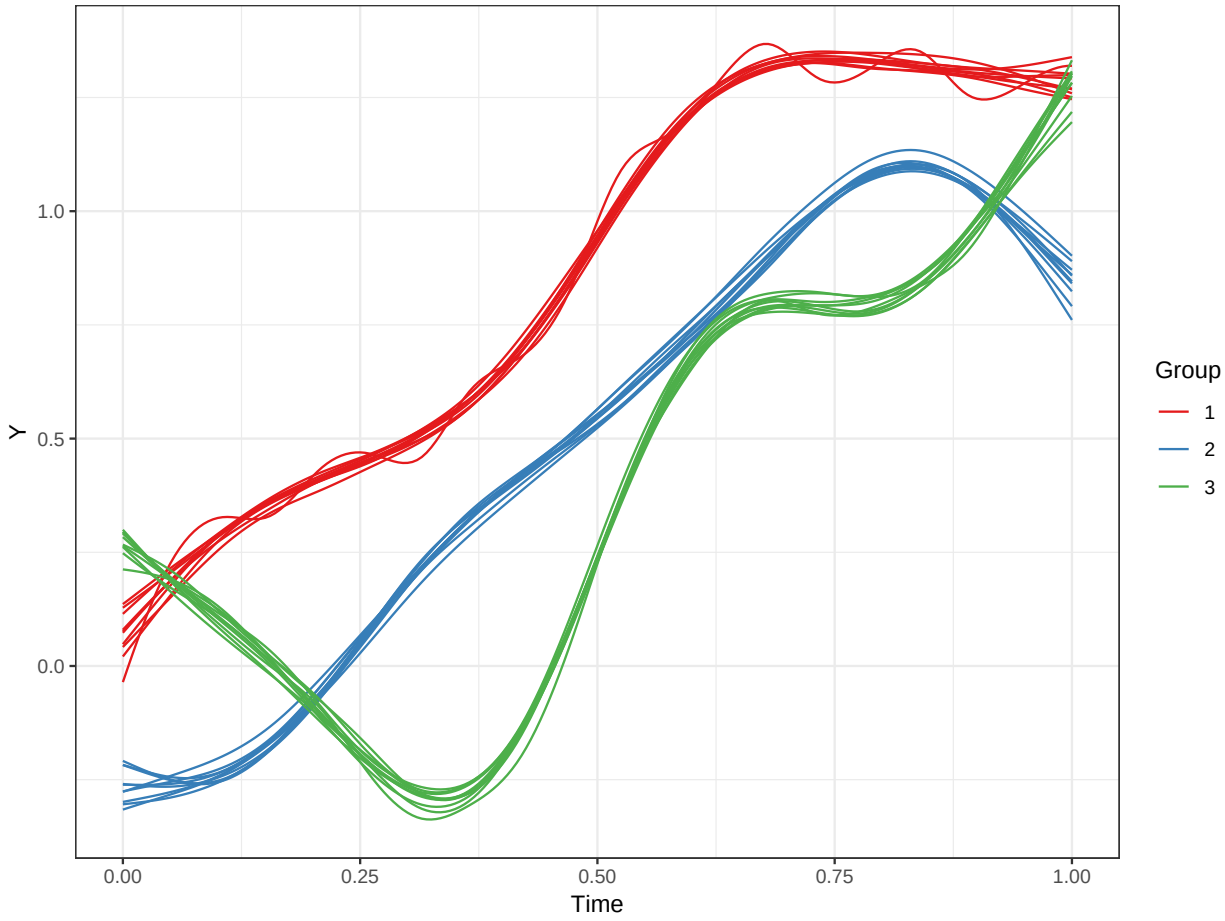
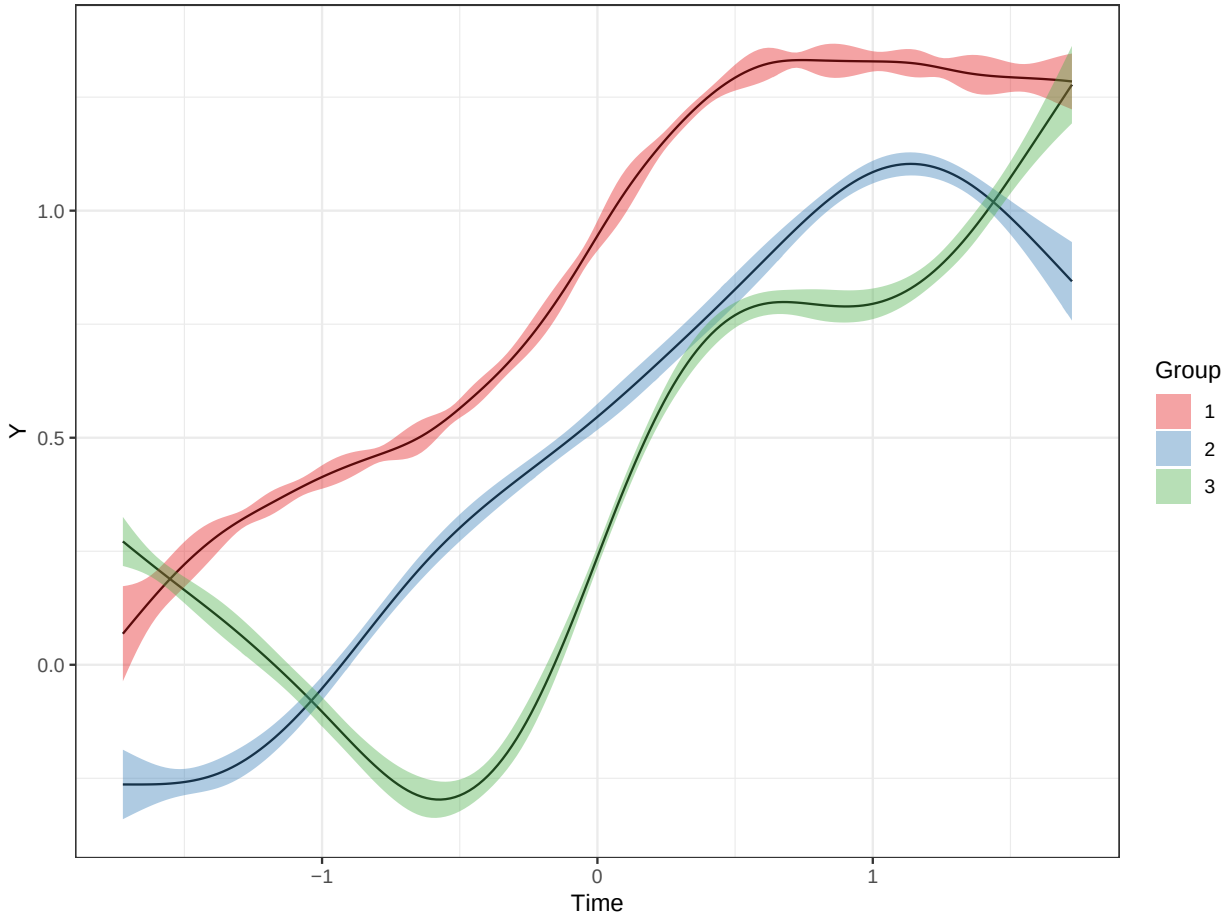


Figure 3: The group-level mean functions for the smoothed simulated data ± 2 point-wise standard deviations.

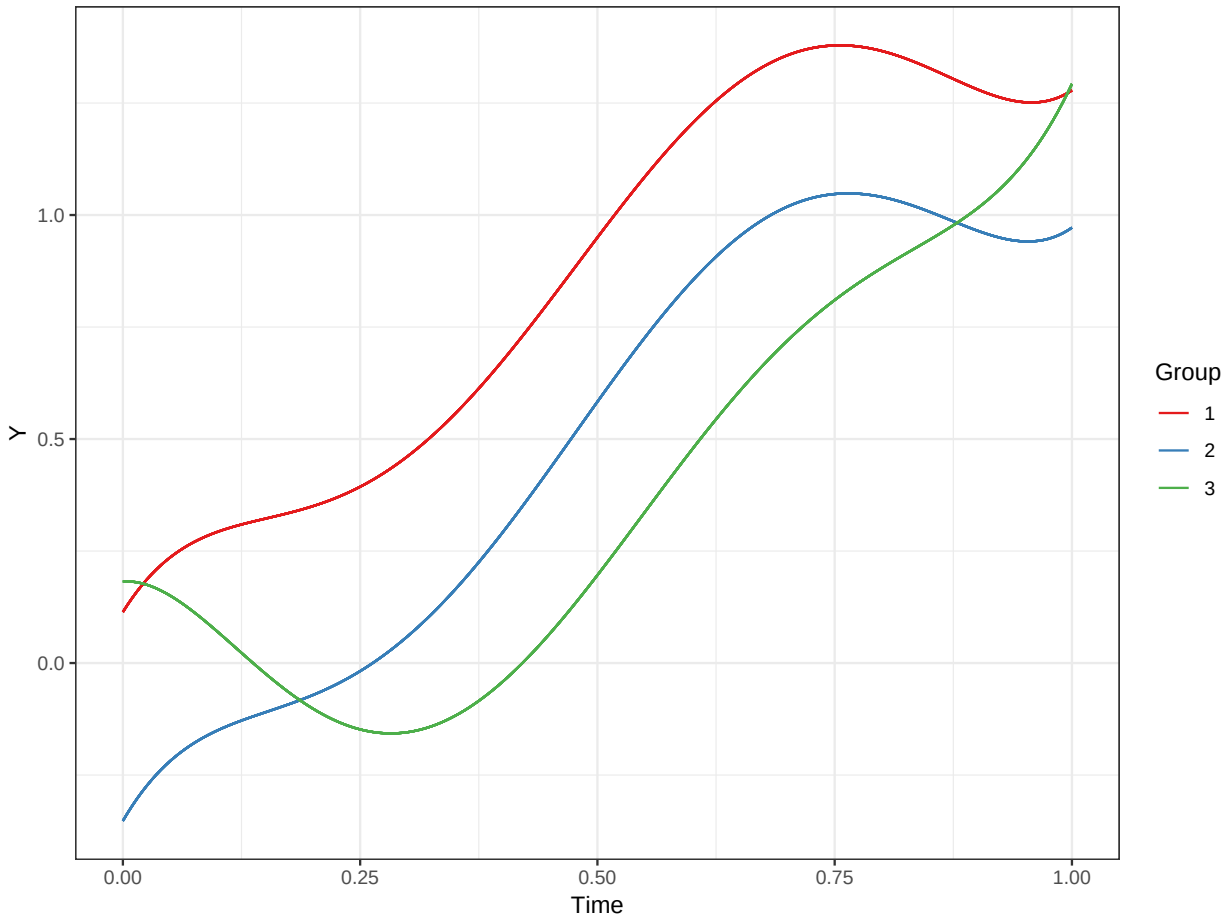


Analyze Using lm()

Fit a 5th-order linear model with time-by-group and Time^2-by-group interactions using lm():

```
## Fit the model
mod = lm(Y_Smooth ~ Group + Time_Standardized + Time_Stan_2 + Time_Stan_3 +
           Time_Stan_4 + Time_Stan_5 +
           Group*Time_Standardized + Group*Time_Stan_2, data = dat)

## Calculate the fitted values
dat$Y_Smooth_Fitted_lm = fitted(mod)
```

Figure 4: The group-level mean functions from lm() using a 5th-order linear model with time-by-group and Time^2-by-group interactions.

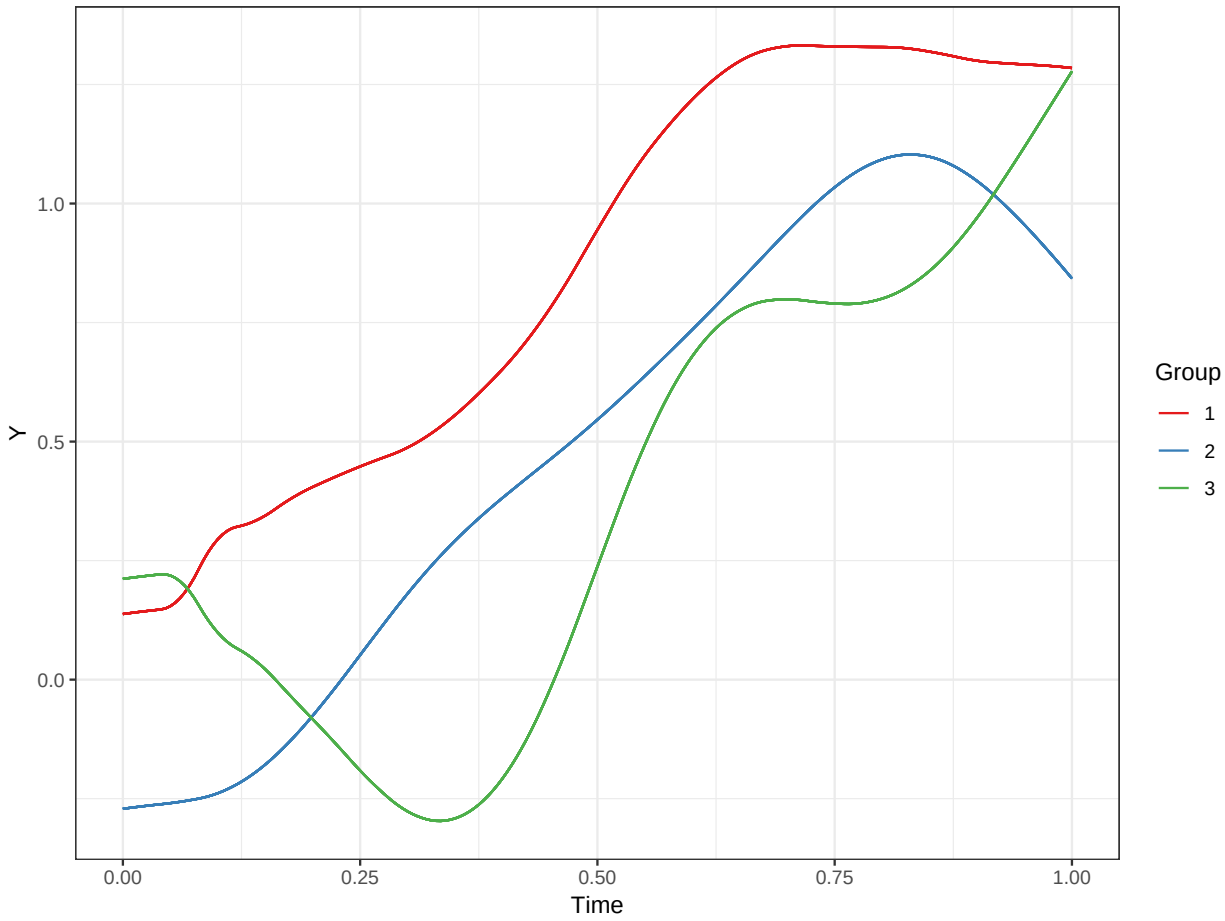


Analyze Using Penalized Splines and lme()

Fit a penalized spline using lme():

```
## Penalized basis functions
K = 24
qtiles = seq(0, 1, length = K + 2)[-c(1, K + 2)]
knots = quantile(dat$Time_Standardized, qtiles)
random.basis = cbind(sapply(knots,
                            function(k){
                              I((dat$Time_Standardized - k)^2)*(dat$Time_Standardized - k > 0)
                              }
                            )
                     )
rm(qtiles)

## Define the fixed basis
fixed.basis = cbind(1,
                    as.numeric(dat$Group == "2"),
                    as.numeric(dat$Group == "3"),
                    dat$Time_Standardized,
                    dat$Time_Stan_2)

## Partition the random.basis matrix
random.basis.1 = as.numeric(dat$Group == "1")*random.basis
random.basis.2 = as.numeric(dat$Group == "2")*random.basis
random.basis.3 = as.numeric(dat$Group == "3")*random.basis

## Define Y
Y = dat$Y_Smooth

## Fit the model
group.1 = group.2 = group.3 = rep(1, nrow(dat))
mod = lme(Y ~ fixed.basis - 1, random = list(group.1 = pdIdent(~ random.basis.1 - 1),
                                             group.2 = pdIdent(~ random.basis.2 - 1),
                                             group.3 = pdIdent(~ random.basis.3 - 1)) )
rm(group.1, group.2, group.3)

## Extract the model coefficients
coefs = c( unlist(mod$coefficients$fixed), unlist(mod$coefficients$random) )

## Extract the lambda estimates
#var.e2 = mod$sigma^2
#var.u2 = getVarCov(mod)[1,1]

## Calculate the fitted values
dat$Y_Smooth_Fitted_lme = as.vector( cbind(fixed.basis,
                                           random.basis.1,
                                           random.basis.2,
                                           random.basis.3) %*% coefs )

## Remove R objects
rm(fixed.basis, random.basis, coefs, knots, Y)
```

Figure 5: The group-level mean functions using penalized splines and lme().



Analyze Using JAGS

Fit a 5th-order linear model with time-by-group and Time^2-by-group interactions using JAGS:

```
## Define values
n = nrow(dat)
Y = dat$Y_Smooth
X = c(dat$Time_Standardized)
X_2 = c(dat$Time_Stan_2)
X_3 = c(dat$Time_Stan_3)
X_4 = c(dat$Time_Stan_4)
X_5 = c(dat$Time_Stan_5)
G.2 = as.numeric(dat$Group == "2")
G.3 = as.numeric(dat$Group == "3")

## Create the model
model_string <- "model{

# Likelihood
for(i in 1:n){
Y[i]   ~ dnorm(mu[i], inv.var)
mu[i] <- beta[1] + beta[2]*X[i] + beta[3]*X_2[i] + beta[4]*X_3[i] + beta[5]*X_4[i] + beta[6]*X_5[i] + b
}

# Prior for beta
for(j in 1:12){
beta[j] ~ dnorm(0,0.0001)
}

# Prior for the inverse variance
inv.var   ~ dgamma(0.01, 0.01)
sigma     <- 1/sqrt(inv.var)

}"

## Compile the model
model = jags.model(textConnection(model_string),
                   data = list(Y = Y,
                               n = n,
                               X = X,
                               X_2 = X_2,
                               X_3 = X_3,
                               X_4 = X_4,
                               X_5 = X_5,
                               G.2 = G.2,
                               G.3 = G.3),
                   n.chains = 3,
                   n.adapt  = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6030
##    Unobserved stochastic nodes: 13
```

```
##    Total graph size: 50880
##
## Initializing model
```

```
## Burn-in samples
update(model,
       1000,
       progress.bar = "none")

## Draw additional samples
samp = coda.samples(model,
                    variable.names = c("beta","sigma"),
                    thin = 3,
                    n.iter = 5000,
                    progress.bar = "none")
```

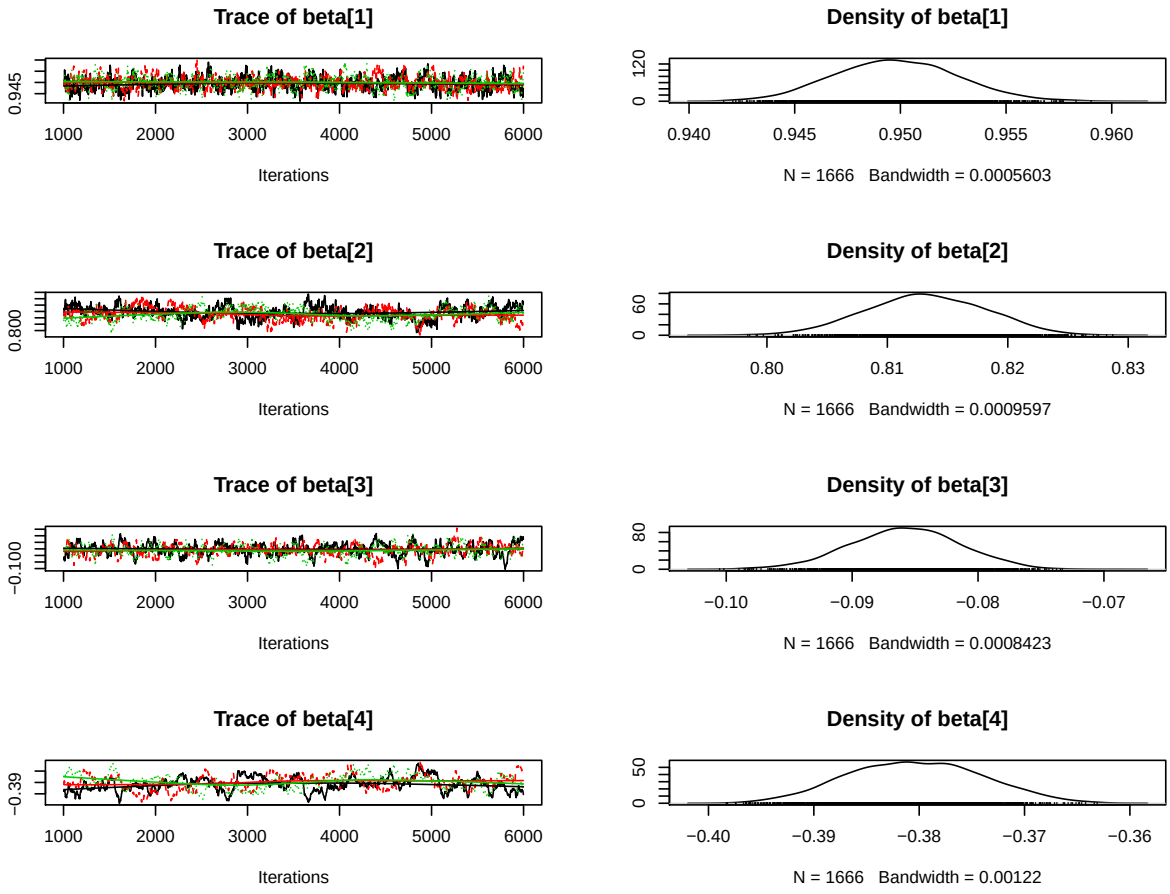
Investigate the fit of the model and extract the fitted values:

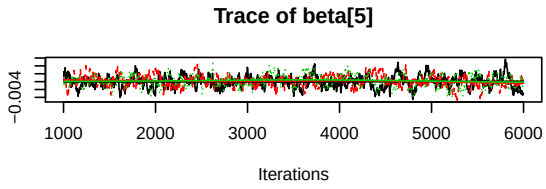
```
## Investigate the model
summary(samp)
```

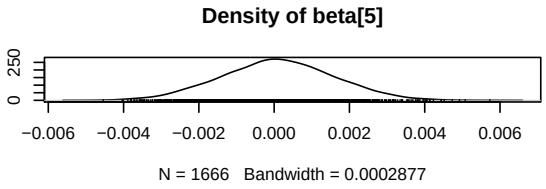
```
##
## Iterations = 1003:5998
## Thinning interval = 3
## Number of chains = 3
## Sample size per chain = 1666
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean       SD  Naive SE Time-series SE
## beta[1]   9.498e-01 0.002903 4.106e-05      1.340e-04
## beta[2]   8.132e-01 0.004973 7.034e-05      4.719e-04
## beta[3]  -8.580e-02 0.004365 6.174e-05      2.753e-04
## beta[4]  -3.804e-01 0.006321 8.941e-05      7.817e-04
## beta[5]   9.643e-05 0.001491 2.109e-05      8.517e-05
## beta[6]   7.417e-02 0.001845 2.610e-05      2.193e-04
## beta[7]  -3.666e-01 0.003712 5.250e-05      1.040e-04
## beta[8]  -7.538e-01 0.003636 5.144e-05      1.072e-04
## beta[9]   4.628e-02 0.002483 3.512e-05      4.559e-05
## beta[10] -1.558e-02 0.002521 3.565e-05      4.562e-05
## beta[11] -6.656e-03 0.002768 3.915e-05      7.937e-05
## beta[12]  2.676e-01 0.002693 3.810e-05      7.608e-05
## sigma     7.858e-02 0.000711 1.006e-05      1.077e-05
##
## 2. Quantiles for each variable:
##
##               2.5%        25%        50%       75%     97.5%
## beta[1]   0.944260  0.9477764  0.9497352  0.951737  0.955594
## beta[2]   0.803412  0.8098141  0.8131476  0.816708  0.822589
## beta[3]  -0.094575 -0.0887183 -0.0857545 -0.082858 -0.077459
## beta[4]  -0.392439 -0.3849208 -0.3805110 -0.375950 -0.368040
## beta[5]  -0.002822 -0.0009153  0.0000842  0.001093  0.003028
## beta[6]   0.070567  0.0728579  0.0742003  0.075519  0.077659
## beta[7]  -0.373809 -0.3691779 -0.3665755 -0.364060 -0.359382
## beta[8]  -0.760932 -0.7561885 -0.7538406 -0.751333 -0.746719
```

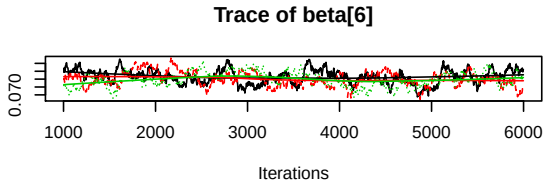
```
## beta[9]   0.041386  0.0445910  0.0462803  0.047979  0.051110
## beta[10] -0.020537 -0.0172463 -0.0155750 -0.013855 -0.010602
## beta[11] -0.011910 -0.0085735 -0.0067087 -0.004766 -0.001247
## beta[12]  0.262336  0.2658226  0.2676153  0.269416  0.272976
## sigma     0.077191  0.0781095  0.0785591  0.079062  0.080005
```

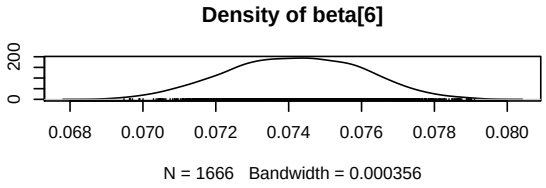
```
plot(samp)
```

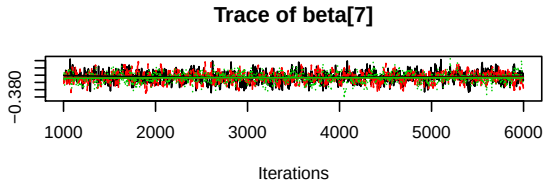


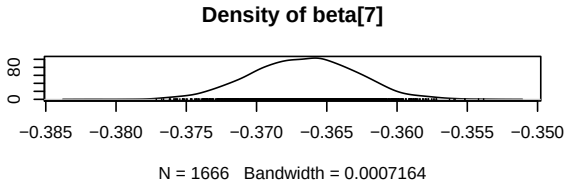

Trace of beta[5]
−0.004
1000
2000
3000
4000
5000
6000
Iterations

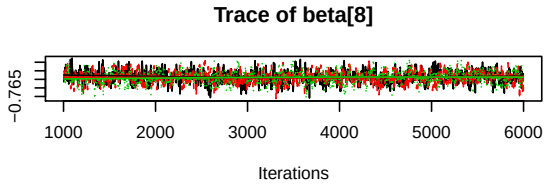

Density of beta[5]
0
250
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006
N = 1666 Bandwidth = 0.0002877

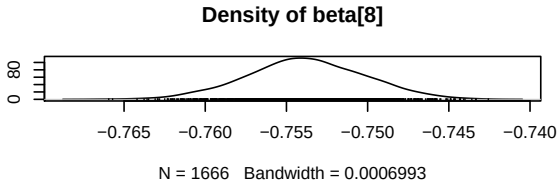

Trace of beta[6]
0.070
1000
2000
3000
4000
5000
6000
Iterations

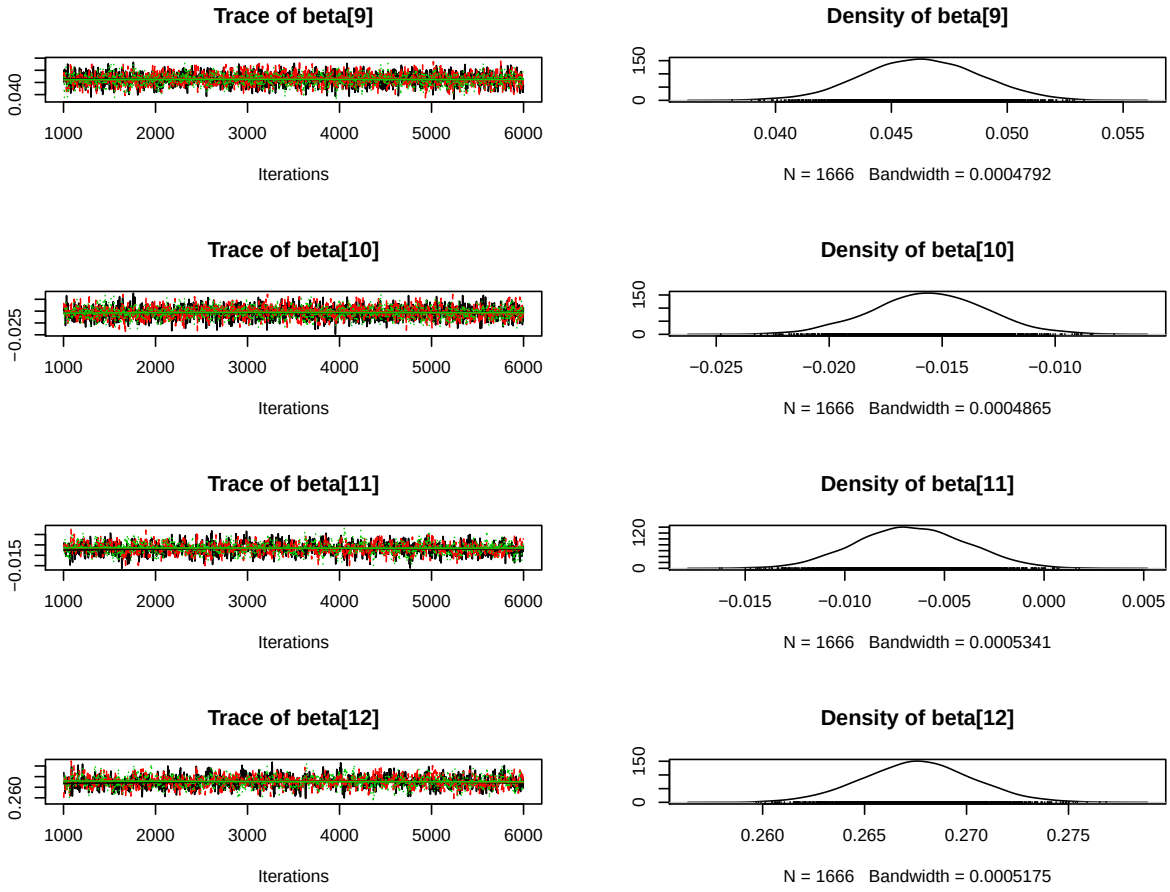

Density of beta[6]
0
200
0.068
0.070
0.072
0.074
0.076
0.078
0.080
N = 1666 Bandwidth = 0.000356

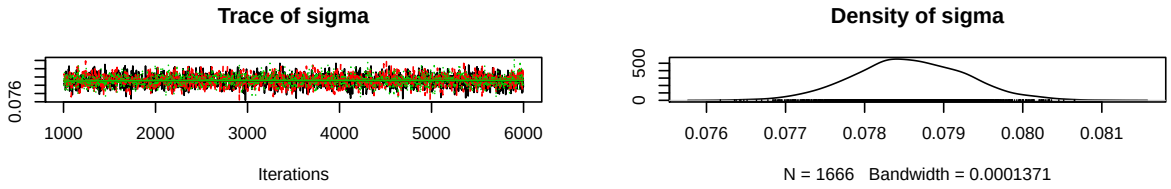

Trace of beta[7]
−0.380
1000
2000
3000
4000
5000
6000
Iterations


Density of beta[7]
0
80
−0.385
−0.380
−0.375
−0.370
−0.365
−0.360
−0.355
−0.350
N = 1666 Bandwidth = 0.0007164


Trace of beta[8]
−0.765
1000
2000
3000
4000
5000
6000
Iterations

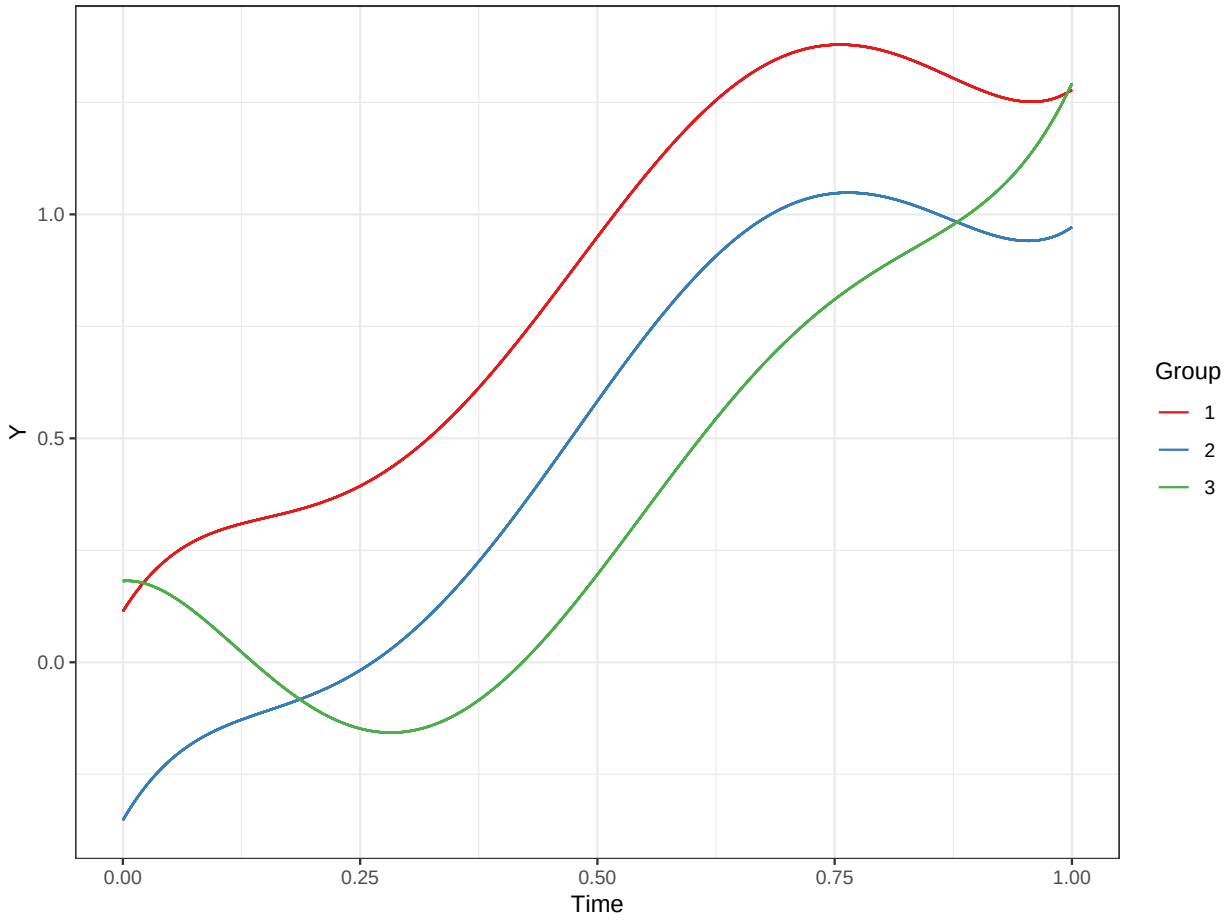

Density of beta[8]
0
80
−0.765
−0.760
−0.755
−0.750
−0.745
−0.740
N = 1666 Bandwidth = 0.0006993


Trace of beta[9]
0.040
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[9]
0
150
0.040
0.045
0.050
0.055
N = 1666 Bandwidth = 0.0004792
Trace of beta[10]
−0.025
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[10]
0
150
−0.025
−0.020
−0.015
−0.010
N = 1666 Bandwidth = 0.0004865
Trace of beta[11]
−0.015
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[11]
0
120
−0.015
−0.010
−0.005
0.000
0.005
N = 1666 Bandwidth = 0.0005341
Trace of beta[12]
0.260
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[12]
0
150
0.260
0.265
0.270
0.275
N = 1666 Bandwidth = 0.0005175




```
## Save the coefficients from the model
coefs = data.frame( summary(samp)$statistics )

## Extract the beta estimates
betas = coefs$Mean[ row.names(coefs) %in% paste0( paste0("beta[",1:12), "]") ]

## Calculate the fitted values
dat$Y_Smooth_Fitted_JAGS_5 = c(t(betas) %*% t(cbind(1, X, X_2, X_3, X_4, X_5,
                                                    G.2, G.3, X*G.2, X*G.3,
                                                    X_2*G.2, X_2*G.3)) )

## Remove R objects
rm(model, samp, coefs, model_string, n, X_3, X_4, X_5, betas)
```

Figure 6: The group-level mean functions from JAGS using a 5th-order linear model with time-by-group and Time^2-by-group interactions.


Y
1.0
0.5
0.0
0.00
0.25
0.50
0.75
1.00
Time
Group
1
2
3

Analyze Using Penalized Splines and JAGS

Fit a penalized spline using JAGS:

```
# Define values
n = nrow(dat)

## Create the model
model_string <- "model{

# Likelihood
for(i in 1:n){
Y[i]   ~ dnorm(mu[i], inv.var)
mu[i] <- beta[1] + beta[2]*G.2[i] + beta[3]*G.3[i] + beta[4]*X[i] + beta[5]*X_2[i] + alpha[1]*random.bas
}

# Prior for beta
for(j in 1:5){
beta[j] ~ dnorm(0,0.0001)
}

# Prior for alpha
for(k in 1:(3*K)){
alpha[k] ~ dnorm(0,0.0001)
}

# Prior for the inverse variance
inv.var    ~ dgamma(0.01, 0.01)
sigma     <- 1/sqrt(inv.var)

}"

## Compile the model
model = jags.model(textConnection(model_string),
                   data = list(Y = Y,
                               n = n,
                               X = X,
                               X_2 = X_2,
                               G.2 = G.2,
                               G.3 = G.3,
                               K = K,
                               random.basis.1 = random.basis.1,
                               random.basis.2 = random.basis.2,
                               random.basis.3 = random.basis.3),
                   n.chains = 3,
                   n.adapt  = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 6030
##    Unobserved stochastic nodes: 78
##    Total graph size: 472577
##
```

```
## Initializing model
```

```
## Burn-in samples
update(model,
       1000,
       progress.bar = "none")

## Draw additional samples
samp = coda.samples(model,
                    variable.names = c("beta","alpha","sigma"),
                    thin = 3,
                    n.iter = 5000,
                    progress.bar = "none")
```

Investigate the fit of the model and extract the fitted values:

```
## Investigate the model
summary(samp)
```

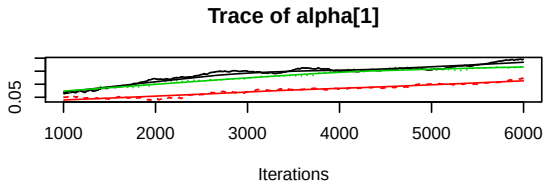
```
##
## Iterations = 1003:5998
## Thinning interval = 3
## Number of chains = 3
## Sample size per chain = 1666
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean        SD  Naive SE Time-series SE
## alpha[1]   0.1162596  0.039824 5.633e-04       0.011363
## alpha[2]   0.0812388  0.048139 6.809e-04       0.004754
## alpha[3]   0.0023003  0.019449 2.751e-04       0.005334
## alpha[4]  -0.0055698  0.060813 8.602e-04       0.010247
## alpha[5]  -0.0273928  0.052383 7.409e-04       0.013810
## alpha[6]  -0.1074374  0.062800 8.883e-04       0.024469
## alpha[7]  -0.0857162  0.052892 7.482e-04       0.011453
## alpha[8]  -0.0888489  0.055029 7.784e-04       0.009588
## alpha[9]   0.0628334  0.103071 1.458e-03       0.045371
## alpha[10]  0.0537579  0.104428 1.477e-03       0.034318
## alpha[11]  0.1209571  0.124411 1.760e-03       0.053709
## alpha[12] -0.1005982  0.076320 1.080e-03       0.025305
## alpha[13] -0.1147686  0.157147 2.223e-03       0.020615
## alpha[14] -0.3559387  0.293318 4.149e-03       0.074946
## alpha[15] -0.6977843  0.309648 4.380e-03       0.109194
## alpha[16] -0.2365078  0.351442 4.971e-03       0.071223
## alpha[17]  0.2518026  0.377191 5.335e-03       0.176725
## alpha[18]  0.9691836  0.623452 8.819e-03       0.371438
## alpha[19]  0.5521195  0.440448 6.230e-03       0.164400
## alpha[20] -0.7565838  0.635987 8.996e-03       0.263575
## alpha[21] -0.7066760  1.033065 1.461e-02       0.418911
## alpha[22]  1.9469904  1.694540 2.397e-02       0.507226
## alpha[23] -1.5070931  2.175567 3.077e-02       0.383041
## alpha[24]  0.7286818  2.890781 4.089e-02       0.340278
## alpha[25]  0.1361500  0.024920 3.525e-04       0.009748
## alpha[26]  0.1018134  0.026775 3.787e-04       0.007929
## alpha[27]  0.1047418  0.032269 4.564e-04       0.005176
```

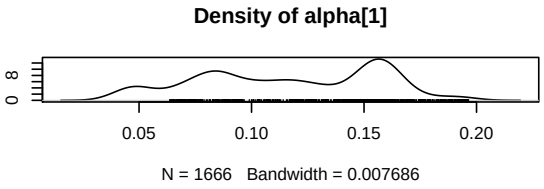
```
## alpha[28]  0.1125364  0.036978 5.231e-04       0.011159
## alpha[29] -0.0003799  0.061069 8.638e-04       0.021194
## alpha[30] -0.1202876  0.088043 1.245e-03       0.024056
## alpha[31] -0.2448329  0.130929 1.852e-03       0.035674
## alpha[32] -0.3086444  0.148085 2.095e-03       0.032618
## alpha[33] -0.4091620  0.120622 1.706e-03       0.012744
## alpha[34] -0.2056296  0.087731 1.241e-03       0.023211
## alpha[35] -0.0729451  0.197812 2.798e-03       0.104340
## alpha[36]  0.2465318  0.223348 3.159e-03       0.098220
## alpha[37]  0.5902134  0.259238 3.667e-03       0.111290
## alpha[38]  0.6073840  0.136093 1.925e-03       0.044130
## alpha[39]  0.3855411  0.421311 5.959e-03       0.230972
## alpha[40] -0.6544915  0.777551 1.100e-02       0.381821
## alpha[41] -0.8970813  0.547178 7.740e-03       0.192690
## alpha[42] -1.3821884  0.843964 1.194e-02       0.402015
## alpha[43]  0.9203881  1.770704 2.505e-02       1.170407
## alpha[44]  0.8516159  1.740964 2.463e-02       0.613072
## alpha[45] -0.6824377  4.336988 6.135e-02       3.568719
## alpha[46] -0.7176114  5.464382 7.729e-02       2.848422
## alpha[47]  1.5311469  5.907083 8.356e-02       1.990404
## alpha[48] -0.3481189  6.199916 8.770e-02       1.509658
## alpha[49] -0.2321412  0.051681 7.310e-04       0.021598
## alpha[50] -0.2491400  0.052588 7.439e-04       0.023419
## alpha[51] -0.1898735  0.031862 4.507e-04       0.011501
## alpha[52] -0.0842308  0.015683 2.218e-04       0.006170
## alpha[53]  0.0719569  0.063655 9.004e-04       0.026415
## alpha[54]  0.3564686  0.137660 1.947e-03       0.058662
## alpha[55]  0.5892549  0.138223 1.955e-03       0.057087
## alpha[56]  0.6609249  0.155523 2.200e-03       0.056209
## alpha[57]  0.7518156  0.101651 1.438e-03       0.033975
## alpha[58]  0.5843744  0.185954 2.630e-03       0.051680
## alpha[59] -0.0131337  0.318004 4.498e-03       0.150344
## alpha[60] -1.1045172  0.497080 7.031e-03       0.219930
## alpha[61] -1.7771111  0.496498 7.023e-03       0.192190
## alpha[62] -2.0265421  0.280319 3.965e-03       0.081375
## alpha[63] -1.3960528  0.819941 1.160e-02       0.466706
## alpha[64]  0.8676368  1.466609 2.075e-02       0.714003
## alpha[65]  3.5038016  1.366324 1.933e-02       0.479456
## alpha[66]  2.4426347  2.129385 3.012e-02       1.240093
## alpha[67] -0.4661451  3.931032 5.560e-02       2.084203
## alpha[68] -5.4036842  3.288187 4.651e-02       1.211755
## alpha[69]  3.4742624  9.947726 1.407e-01       7.158253
## alpha[70]  2.9356767 13.156251 1.861e-01       9.221146
## alpha[71] -5.3850237 12.647719 1.789e-01       4.954127
## alpha[72]  3.3137875 12.139663 1.717e-01       3.390474
## beta[1]    0.6460561  0.015556 2.200e-04       0.002956
## beta[2]   -0.4664607  0.006809 9.632e-05       0.001112
## beta[3]   -0.1514627  0.027548 3.897e-04       0.009521
## beta[4]    0.5352347  0.030282 4.283e-04       0.005607
## beta[5]    0.1783599  0.013081 1.850e-04       0.001684
## sigma      0.0589147  0.009865 1.395e-04       0.003908
##
## 2. Quantiles for each variable:
##
```

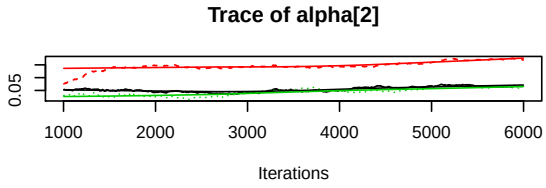
```
##                2.5%       25%      50%       75%     97.5%
## alpha[1]    0.045356  0.083573  0.11690  0.154397  0.180397
## alpha[2]    0.023718  0.045906  0.06073  0.140195  0.171210
## alpha[3]   -0.040805 -0.010746  0.00299  0.020331  0.029549
## alpha[4]   -0.087839 -0.059293 -0.02453  0.053840  0.099543
## alpha[5]   -0.096341 -0.065354 -0.03781  0.006937  0.080334
## alpha[6]   -0.227597 -0.147116 -0.10914 -0.067687  0.005001
## alpha[7]   -0.190432 -0.127076 -0.08006 -0.037251 -0.013479
## alpha[8]   -0.180552 -0.129174 -0.10214 -0.039827  0.008526
## alpha[9]   -0.116091 -0.004498  0.05696  0.151537  0.247037
## alpha[10]  -0.135636 -0.032897  0.04528  0.144181  0.216750
## alpha[11]  -0.109230  0.029832  0.13153  0.235318  0.302429
## alpha[12]  -0.278322 -0.145475 -0.09807 -0.048810  0.061336
## alpha[13]  -0.357321 -0.224031 -0.15635  0.059170  0.143776
## alpha[14]  -0.820816 -0.621699 -0.42031 -0.094360  0.150425
## alpha[15]  -1.246077 -0.979621 -0.59420 -0.415041 -0.272633
## alpha[16]  -0.853179 -0.439832 -0.28562  0.132783  0.291712
## alpha[17]  -0.265358 -0.042101  0.17689  0.495916  1.047093
## alpha[18]   0.216146  0.553463  0.78888  1.136239  2.566605
## alpha[19]  -0.264410  0.167792  0.61363  0.936035  1.229411
## alpha[20]  -2.338403 -1.019136 -0.67321 -0.322675  0.294799
## alpha[21]  -2.373039 -1.453474 -0.78759 -0.192181  1.753285
## alpha[22]  -1.503879  0.717670  2.08781  2.980428  5.296444
## alpha[23]  -6.270523 -2.943541 -1.33563  0.041807  2.373063
## alpha[24]  -4.726074 -1.243160  0.55863  2.583534  6.873914
## alpha[25]   0.086152  0.117259  0.14049  0.154272  0.175279
## alpha[26]   0.057906  0.079487  0.09738  0.125432  0.144357
## alpha[27]   0.044452  0.075157  0.11158  0.127663  0.159324
## alpha[28]   0.051123  0.077539  0.11042  0.143907  0.179292
## alpha[29]  -0.104154 -0.061513  0.01313  0.049958  0.090115
## alpha[30]  -0.290187 -0.192100 -0.12221 -0.028235  0.012155
## alpha[31]  -0.490805 -0.334624 -0.22907 -0.154518 -0.035970
## alpha[32]  -0.588353 -0.390995 -0.30008 -0.208547 -0.048246
## alpha[33]  -0.538268 -0.505850 -0.47530 -0.278700 -0.174719
## alpha[34]  -0.346636 -0.265466 -0.22301 -0.158171  0.018745
## alpha[35]  -0.399913 -0.231278 -0.07103  0.089074  0.290542
## alpha[36]  -0.208545  0.102589  0.25865  0.406552  0.607714
## alpha[37]  -0.009379  0.424504  0.67222  0.747283  0.971265
## alpha[38]   0.273276  0.533626  0.63998  0.713633  0.778495
## alpha[39]  -0.411962  0.042463  0.42137  0.752345  1.045167
## alpha[40]  -1.770126 -1.394220 -0.68583  0.059298  0.644791
## alpha[41]  -1.676082 -1.255027 -0.97532 -0.610532  0.382092
## alpha[42]  -2.714890 -1.987659 -1.49938 -0.898482  0.343801
## alpha[43]  -1.929442 -0.909437  1.21517  2.409380  3.600694
## alpha[44]  -2.784873 -0.339084  0.86908  2.086568  4.026108
## alpha[45]  -6.438545 -4.745384 -1.78345  3.776050  6.394502
## alpha[46]  -9.385634 -5.963812 -0.58666  4.551358  7.188324
## alpha[47] -12.173726 -2.756472  2.26161  6.645158 10.110995
## alpha[48]  -9.497878 -4.907350 -1.26930  2.729232 15.272586
## alpha[49]  -0.322173 -0.271945 -0.24154 -0.193163 -0.135631
## alpha[50]  -0.343206 -0.283708 -0.24960 -0.215650 -0.143012
## alpha[51]  -0.227814 -0.218099 -0.19964 -0.166614 -0.125416
## alpha[52]  -0.105421 -0.096480 -0.08736 -0.074870 -0.045038
## alpha[53]  -0.034085  0.013992  0.07077  0.126697  0.178567
```

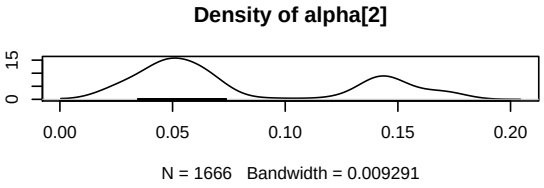
```
## alpha[54]   0.070070   0.250878  0.36222  0.457720  0.603026
## alpha[55]   0.321802   0.488002  0.61784  0.701736  0.797696
## alpha[56]   0.336480   0.555561  0.68967  0.755108  0.932524
## alpha[57]   0.504245   0.698743  0.75312  0.840324  0.877574
## alpha[58]   0.222848   0.455992  0.58706  0.730047  0.896404
## alpha[59]  -0.595758  -0.265845 -0.03326  0.237671  0.562808
## alpha[60]  -1.872235  -1.528806 -1.20354 -0.729657 -0.139014
## alpha[61]  -2.453618  -2.187461 -1.85613 -1.480568 -0.683856
## alpha[62]  -2.511045  -2.250225 -2.01467 -1.856963 -1.404509
## alpha[63]  -2.638813  -2.207031 -1.43015 -0.654191 -0.009962
## alpha[64]  -2.090824  -0.334844  1.21859  2.193449  2.748953
## alpha[65]  -0.103778   3.090716  3.91727  4.220405  5.311936
## alpha[66]  -1.804263   0.583423  2.88977  4.198703  5.344830
## alpha[67]  -5.436112  -3.726139 -1.85489  2.741530  7.084632
## alpha[68] -10.098362  -8.396288 -6.03258 -2.590422  1.005256
## alpha[69] -13.931502  -6.312723  7.69026 12.160052 14.733726
## alpha[70] -14.900675  -9.899244  1.57862 16.304196 23.495720
## alpha[71] -23.692213 -18.093769 -4.55522  5.119609 21.277819
## alpha[72] -29.935682  -3.293444  4.04375 13.271258 19.861001
## beta[1]     0.609042   0.639129  0.64964  0.657965  0.666336
## beta[2]    -0.478260  -0.471477 -0.46690 -0.462041 -0.452504
## beta[3]    -0.213004  -0.170179 -0.14411 -0.129690 -0.114210
## beta[4]     0.466262   0.520393  0.53982  0.560134  0.575471
## beta[5]     0.151870   0.169014  0.17891  0.188783  0.199944
## sigma       0.046991   0.050512  0.05633  0.065673  0.081440
```

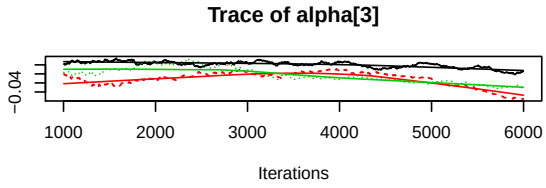
```
plot(samp)
```

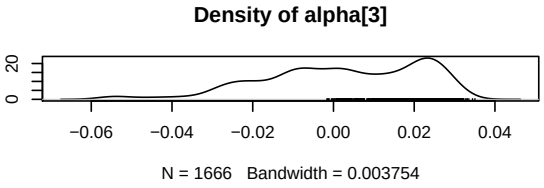

Trace of alpha[1]
0.05
1000
2000
3000
4000
5000
6000
Iterations

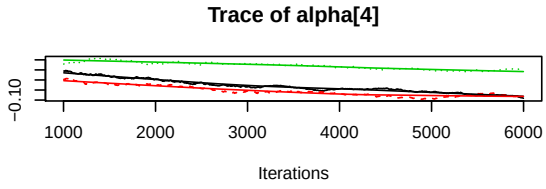

Density of alpha[1]
0
8
0.05
0.10
0.15
0.20
N = 1666 Bandwidth = 0.007686

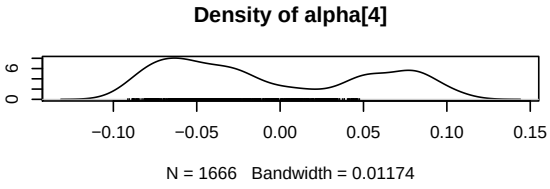

Trace of alpha[2]
0.05
1000
2000
3000
4000
5000
6000
Iterations

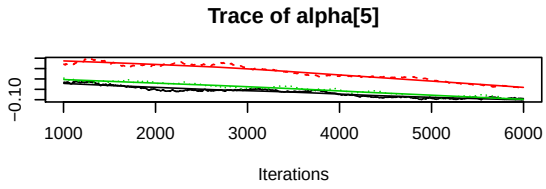

Density of alpha[2]
0
15
0.00
0.05
0.10
0.15
0.20
N = 1666 Bandwidth = 0.009291

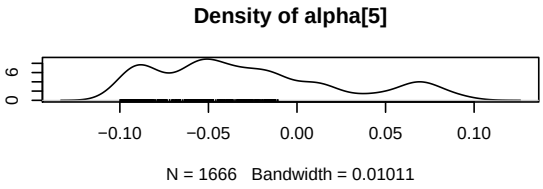

Trace of alpha[3]
−0.04
1000
2000
3000
4000
5000
6000
Iterations

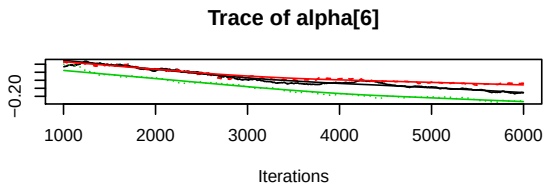

Density of alpha[3]
0
20
−0.06
−0.04
−0.02
0.00
0.02
0.04
N = 1666 Bandwidth = 0.003754

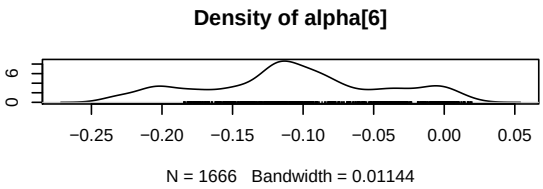

Trace of alpha[4]
−0.10
1000
2000
3000
4000
5000
6000
Iterations

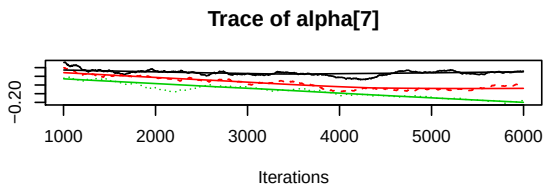

Density of alpha[4]
0
6
−0.10
−0.05
0.00
0.05
0.10
0.15
N = 1666 Bandwidth = 0.01174

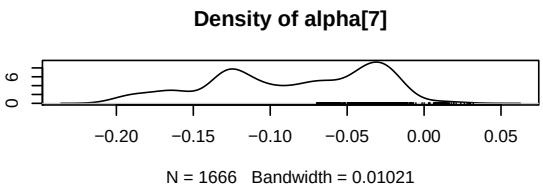

Trace of alpha[5]
−0.10
1000
2000
3000
4000
5000
6000
Iterations

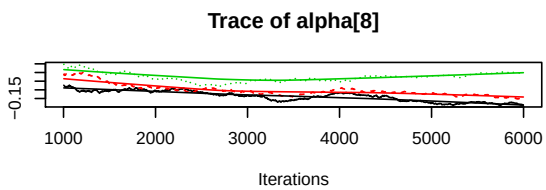

Density of alpha[5]
0 6
−0.10
−0.05
0.00
0.05
0.10
N = 1666 Bandwidth = 0.01011

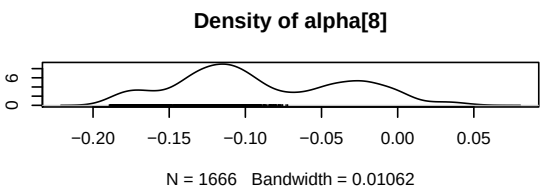

Trace of alpha[6]
−0.20
1000
2000
3000
4000
5000
6000
Iterations

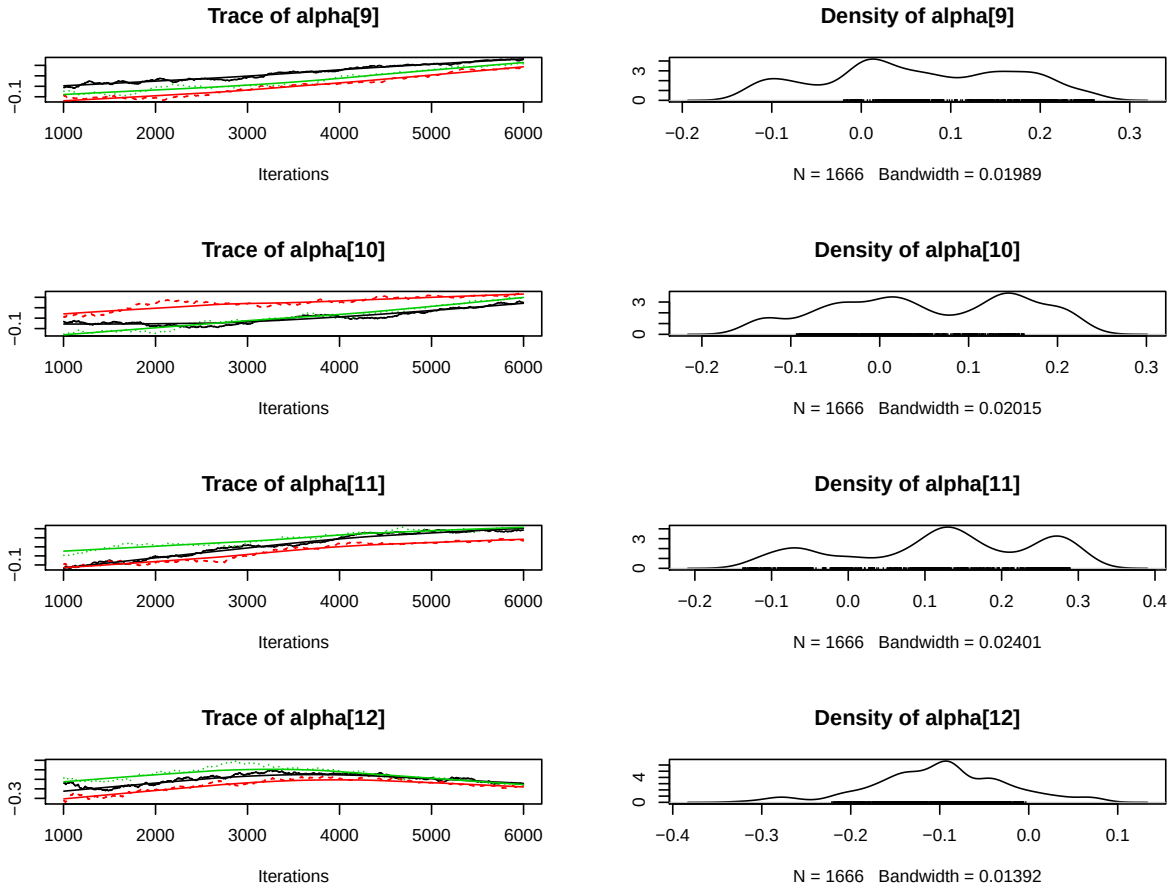

Density of alpha[6]
0 6
−0.25
−0.20
−0.15
−0.10
−0.05
0.00
0.05
N = 1666 Bandwidth = 0.01144

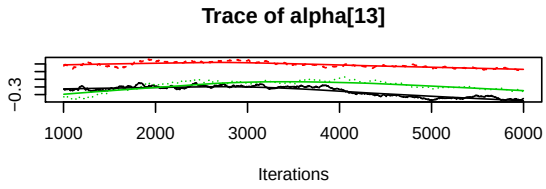

Trace of alpha[7]
−0.20
1000
2000
3000
4000
5000
6000
Iterations

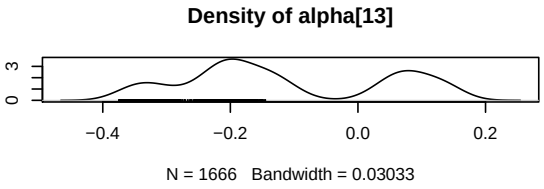

Density of alpha[7]
0 6
−0.20
−0.15
−0.10
−0.05
0.00
0.05
N = 1666 Bandwidth = 0.01021

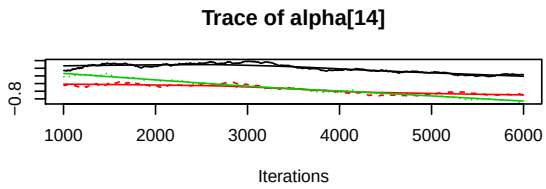

Trace of alpha[8]
−0.15
1000
2000
3000
4000
5000
6000
Iterations

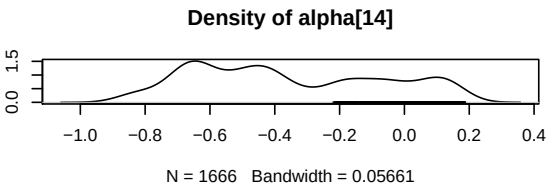

Density of alpha[8]
0 6
−0.20
−0.15
−0.10
−0.05
0.00
0.05
N = 1666 Bandwidth = 0.01062

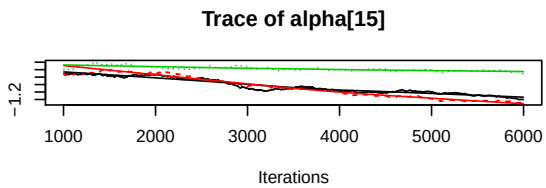

Trace of alpha[9]
−0.1
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[9]
0
3
−0.2
−0.1
0.0
0.1
0.2
0.3
N = 1666 Bandwidth = 0.01989
Trace of alpha[10]
−0.1
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[10]
0
3
−0.2
−0.1
0.0
0.1
0.2
0.3
N = 1666 Bandwidth = 0.02015
Trace of alpha[11]
−0.1
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[11]
0
3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
N = 1666 Bandwidth = 0.02401
Trace of alpha[12]
−0.3
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[12]
0
4
−0.4
−0.3
−0.2
−0.1
0.0
0.1
N = 1666 Bandwidth = 0.01392

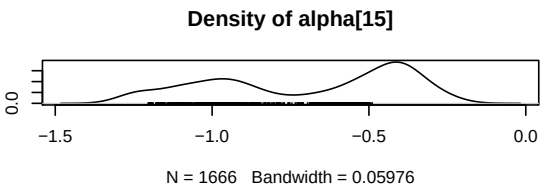

Trace of alpha[13]
−0.3
1000
2000
3000
4000
5000
6000
Iterations

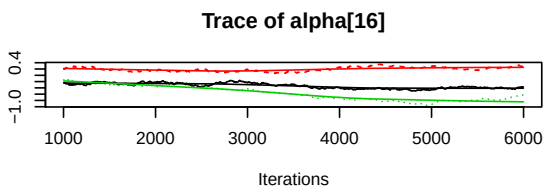

Density of alpha[13]
0
3
−0.4
−0.2
0.0
0.2
N = 1666 Bandwidth = 0.03033

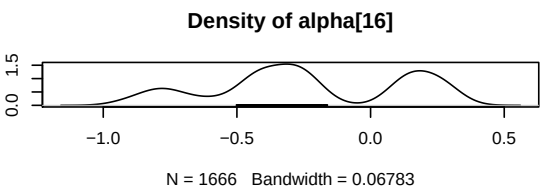

Trace of alpha[14]
−0.8
1000
2000
3000
4000
5000
6000
Iterations

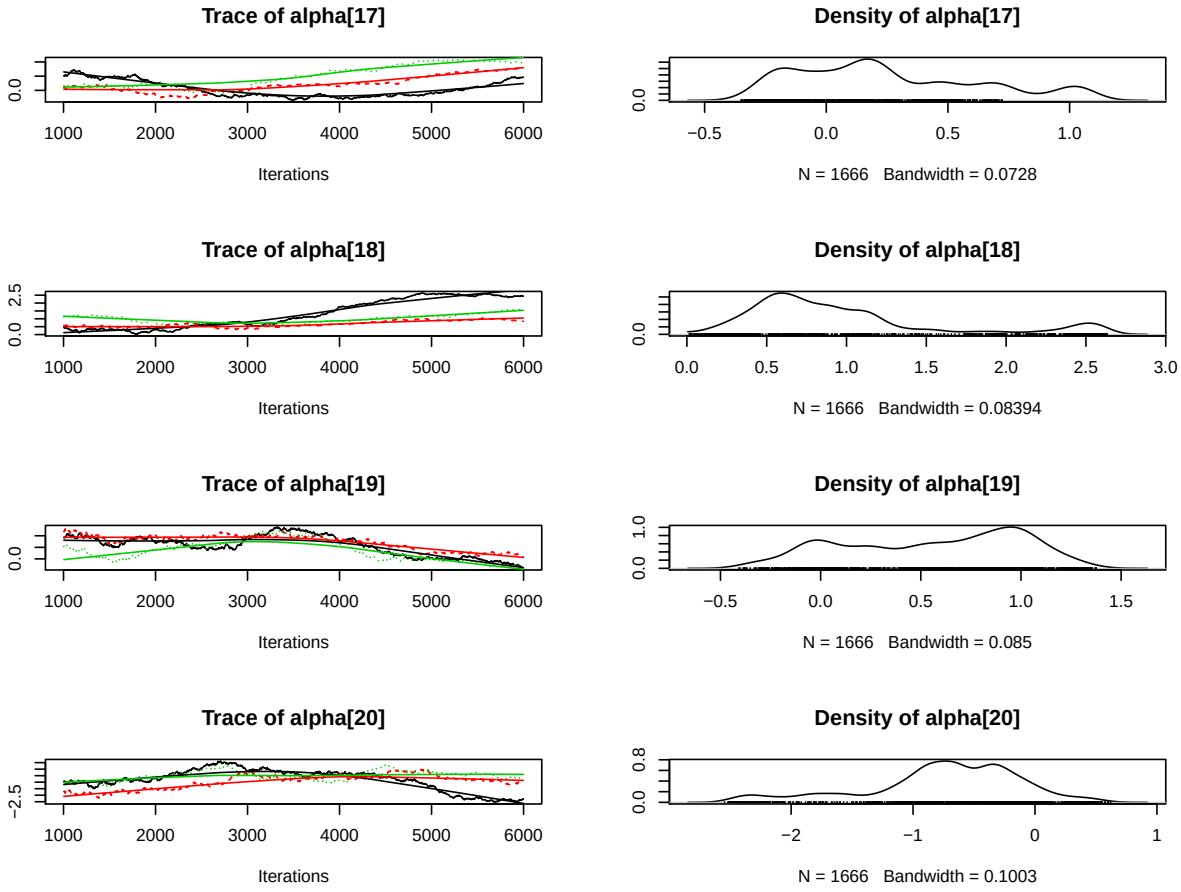

Density of alpha[14]
0.0
1.5
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
N = 1666 Bandwidth = 0.05661

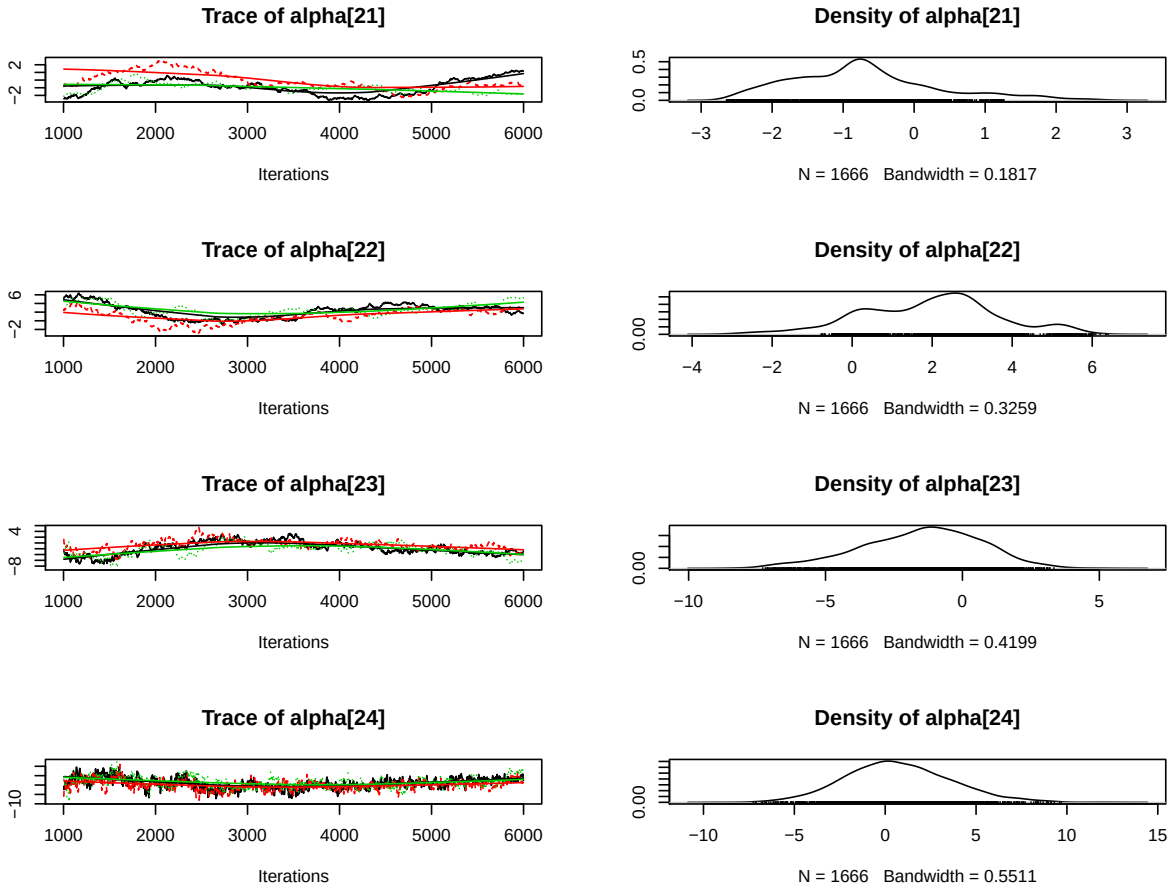

Trace of alpha[15]
−1.2
1000
2000
3000
4000
5000
6000
Iterations

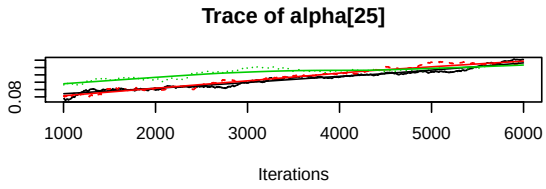

Density of alpha[15]
0.0
−1.5
−1.0
−0.5
0.0
N = 1666 Bandwidth = 0.05976

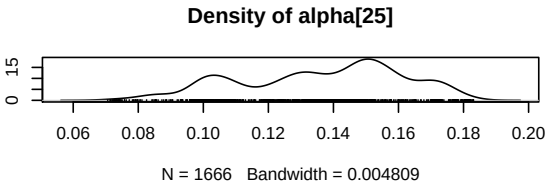

Trace of alpha[16]
−1.0
0.4
1000
2000
3000
4000
5000
6000
Iterations

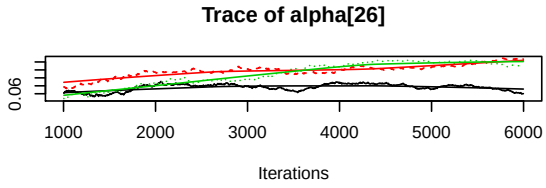

Density of alpha[16]
0.0
1.5
−1.0
−0.5
0.0
0.5
N = 1666 Bandwidth = 0.06783

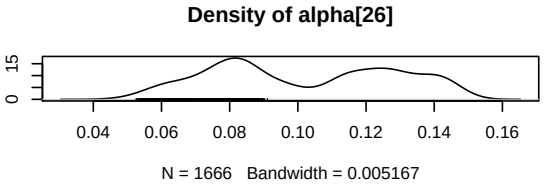

Trace of alpha[17]
Density of alpha[17]
Iterations
N = 1666 Bandwidth = 0.0728
Trace of alpha[18]
Density of alpha[18]
Iterations
N = 1666 Bandwidth = 0.08394
Trace of alpha[19]
Density of alpha[19]
Iterations
N = 1666 Bandwidth = 0.085
Trace of alpha[20]
Density of alpha[20]
Iterations
N = 1666 Bandwidth = 0.1003

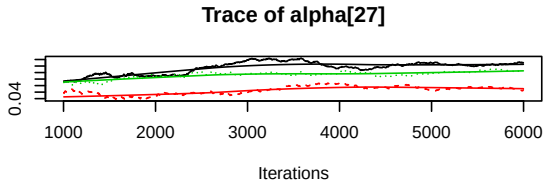

Trace of alpha[21]
−2 2
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[21]
0.0 0.5
−3 −2 −1 0 1 2 3
N = 1666 Bandwidth = 0.1817
Trace of alpha[22]
−2 6
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[22]
0.00
−4 −2 0 2 4 6
N = 1666 Bandwidth = 0.3259
Trace of alpha[23]
−8 4
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[23]
0.00
−10 −5 0 5
N = 1666 Bandwidth = 0.4199
Trace of alpha[24]
−10
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[24]
0.00
−10 −5 0 5 10 15
N = 1666 Bandwidth = 0.5511

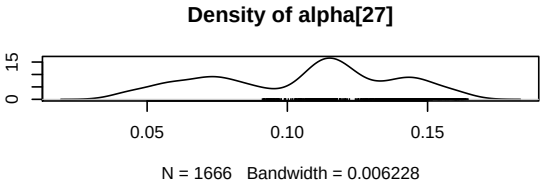

Trace of alpha[25]
0.08
1000
2000
3000
4000
5000
6000
Iterations

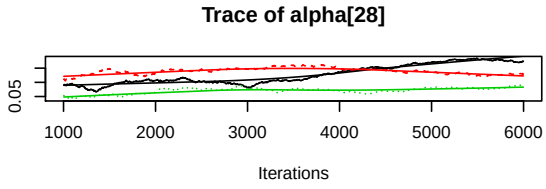

Density of alpha[25]
0
15
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
N = 1666 Bandwidth = 0.004809

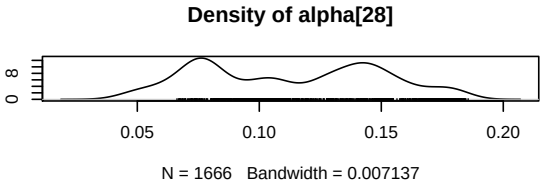

Trace of alpha[26]
0.06
1000
2000
3000
4000
5000
6000
Iterations

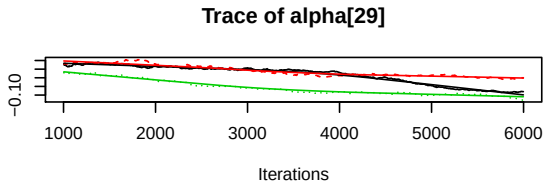

Density of alpha[26]
0
15
0.04
0.06
0.08
0.10
0.12
0.14
0.16
N = 1666 Bandwidth = 0.005167

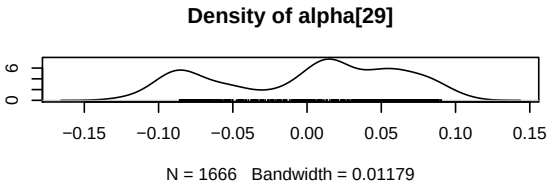

Trace of alpha[27]
0.04
1000
2000
3000
4000
5000
6000
Iterations

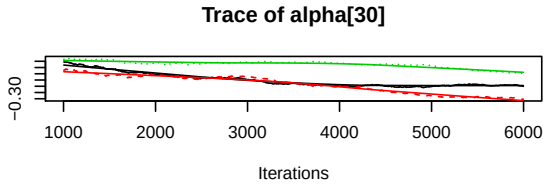

Density of alpha[27]
0
15
0.05
0.10
0.15
N = 1666 Bandwidth = 0.006228

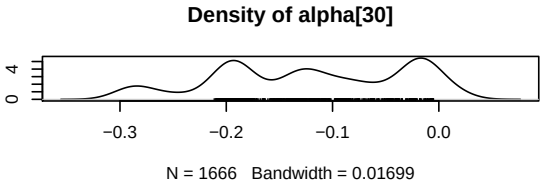

Trace of alpha[28]
0.05
1000
2000
3000
4000
5000
6000
Iterations

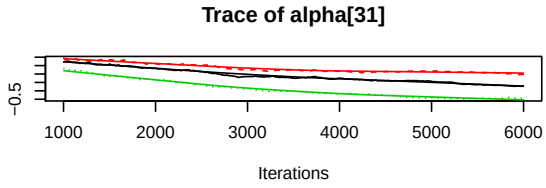

Density of alpha[28]
0
8
0.05
0.10
0.15
0.20
N = 1666 Bandwidth = 0.007137

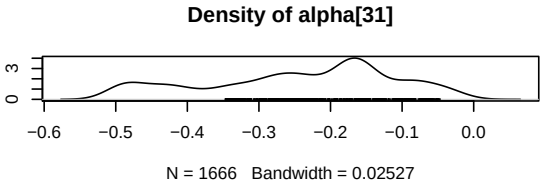

Trace of alpha[29]
−0.10
1000
2000
3000
4000
5000
6000
Iterations

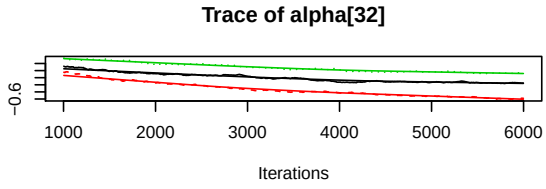

Density of alpha[29]
0
6
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
N = 1666 Bandwidth = 0.01179

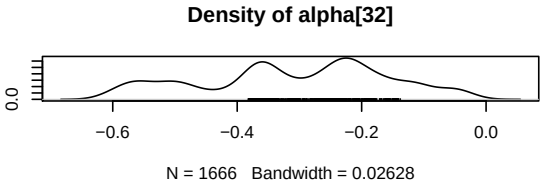

Trace of alpha[30]
−0.30
1000
2000
3000
4000
5000
6000
Iterations

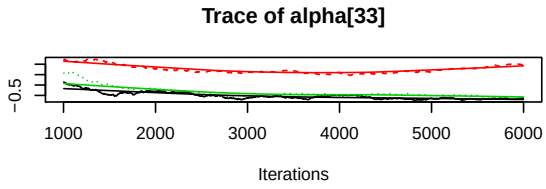

Density of alpha[30]
0
4
−0.3
−0.2
−0.1
0.0
N = 1666 Bandwidth = 0.01699

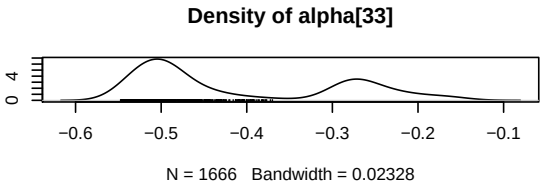

Trace of alpha[31]
−0.5
1000
2000
3000
4000
5000
6000
Iterations

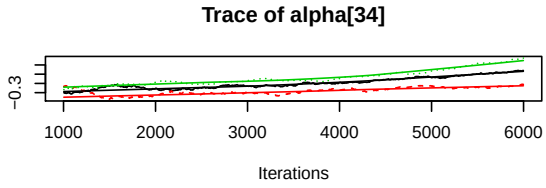

Density of alpha[31]
0
3
−0.6
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
N = 1666 Bandwidth = 0.02527

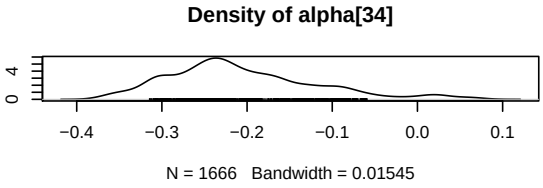

Trace of alpha[32]
−0.6
1000
2000
3000
4000
5000
6000
Iterations

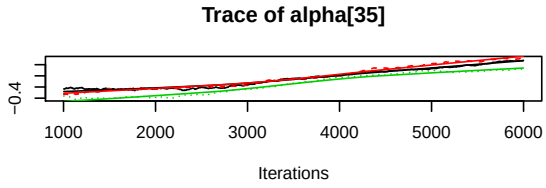

Density of alpha[32]
0.0
−0.6
−0.4
−0.2
0.0
N = 1666 Bandwidth = 0.02628

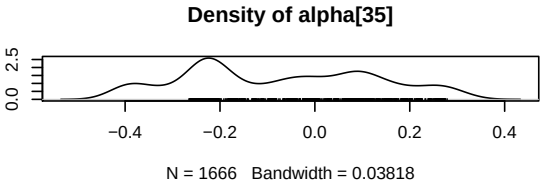

Trace of alpha[33]
−0.5
1000
2000
3000
4000
5000
6000
Iterations

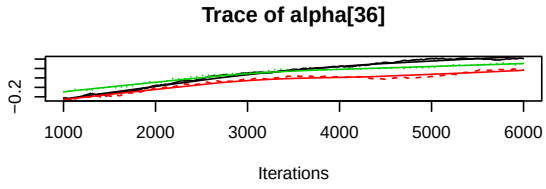

Density of alpha[33]
0
4
−0.6
−0.5
−0.4
−0.3
−0.2
−0.1
N = 1666 Bandwidth = 0.02328

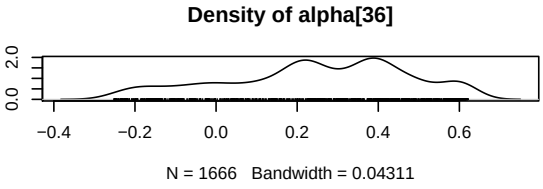

Trace of alpha[34]
−0.3
1000
2000
3000
4000
5000
6000
Iterations

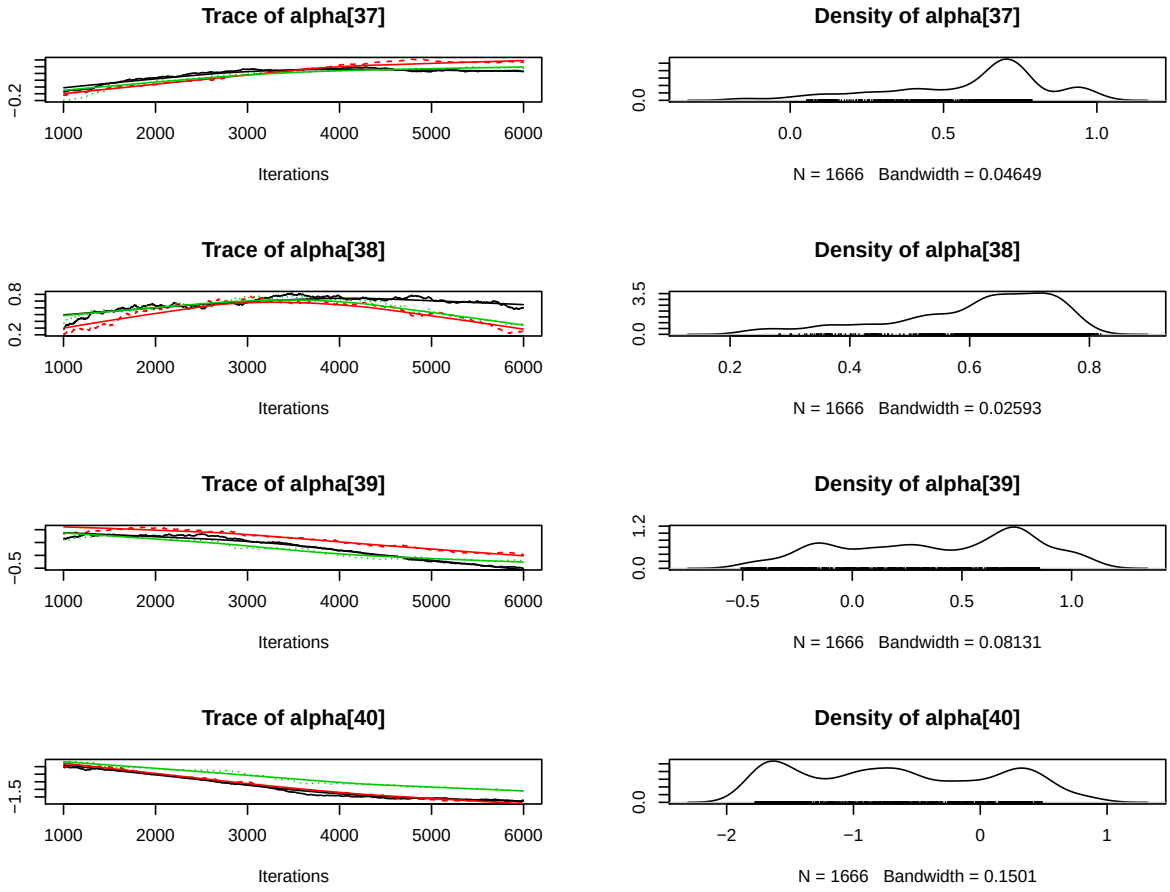

Density of alpha[34]
0
4
−0.4
−0.3
−0.2
−0.1
0.0
0.1
N = 1666 Bandwidth = 0.01545

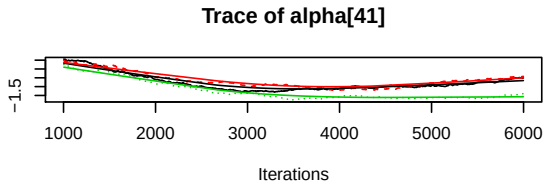

Trace of alpha[35]
−0.4
1000
2000
3000
4000
5000
6000
Iterations

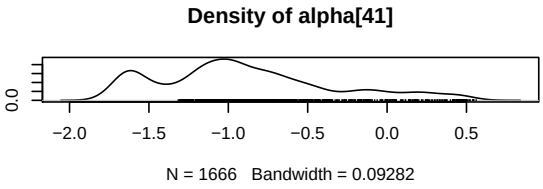

Density of alpha[35]
0.0
2.5
−0.4
−0.2
0.0
0.2
0.4
N = 1666 Bandwidth = 0.03818

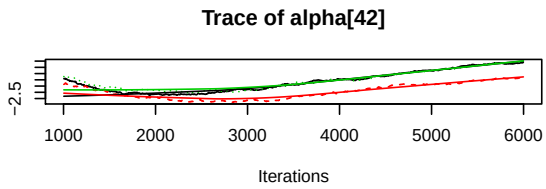

Trace of alpha[36]
−0.2
1000
2000
3000
4000
5000
6000
Iterations

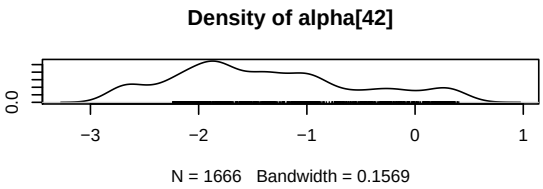

Density of alpha[36]
0.0
2.0
−0.4
−0.2
0.0
0.2
0.4
0.6
N = 1666 Bandwidth = 0.04311

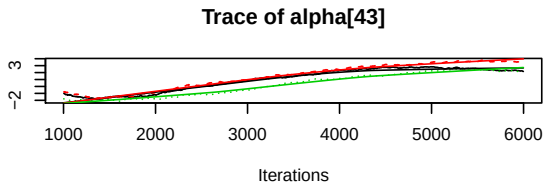

Trace of alpha[37]
−0.2
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[37]
0.0
0.0
0.5
1.0
N = 1666 Bandwidth = 0.04649
Trace of alpha[38]
0.2
0.8
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[38]
0.0
3.5
0.2
0.4
0.6
0.8
N = 1666 Bandwidth = 0.02593
Trace of alpha[39]
−0.5
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[39]
0.0
1.2
−0.5
0.0
0.5
1.0
N = 1666 Bandwidth = 0.08131
Trace of alpha[40]
−1.5
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[40]
0.0
−2
−1
0
1
N = 1666 Bandwidth = 0.1501

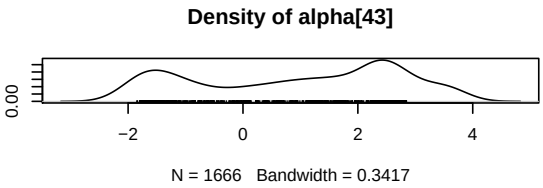

Trace of alpha[41]
−1.5
1000
2000
3000
4000
5000
6000
Iterations

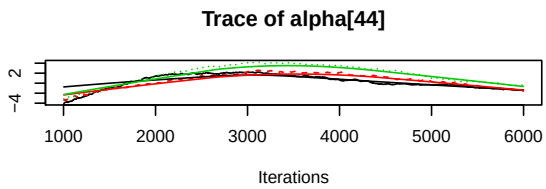

Density of alpha[41]
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
N = 1666 Bandwidth = 0.09282

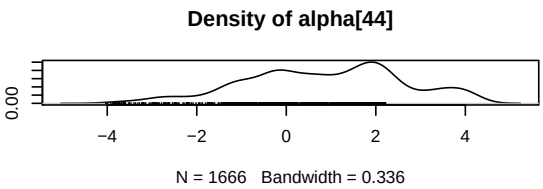

Trace of alpha[42]
−2.5
1000
2000
3000
4000
5000
6000
Iterations

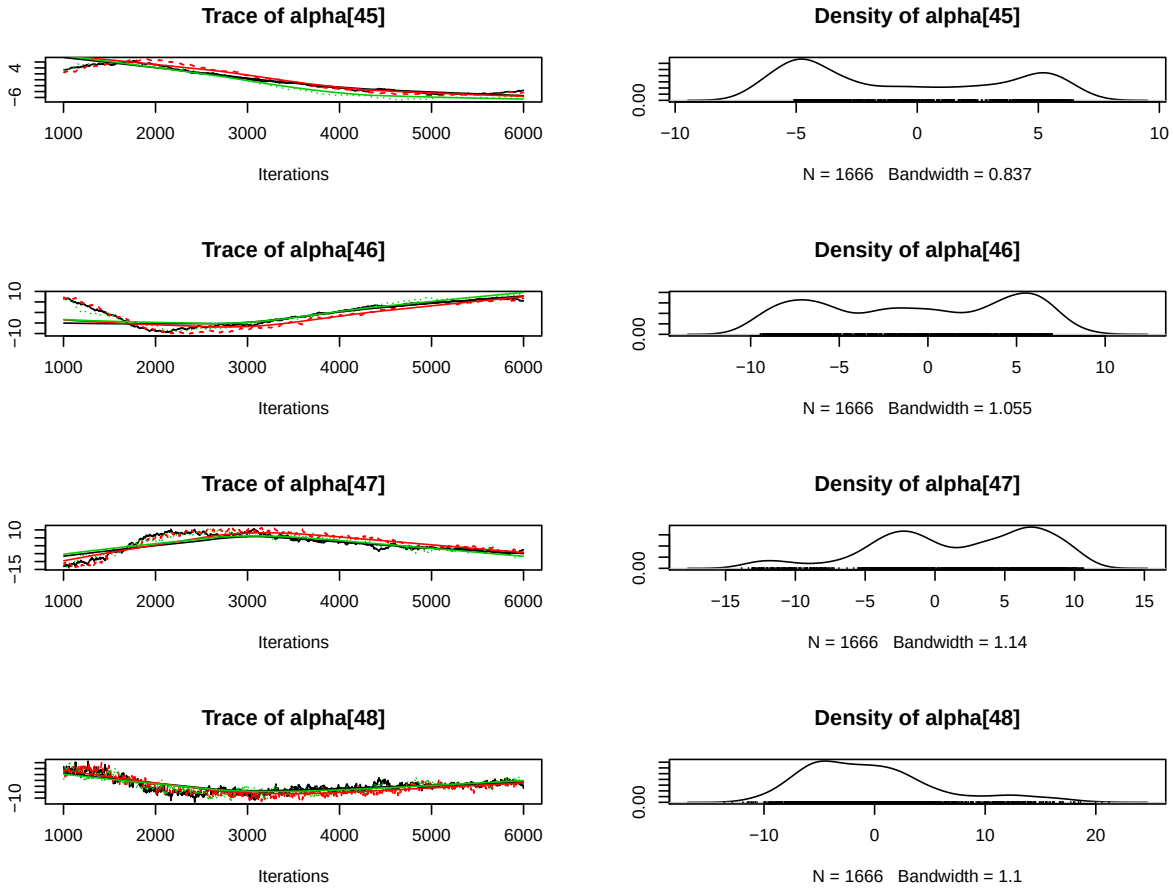

Density of alpha[42]
0.0
−3
−2
−1
0
1
N = 1666 Bandwidth = 0.1569

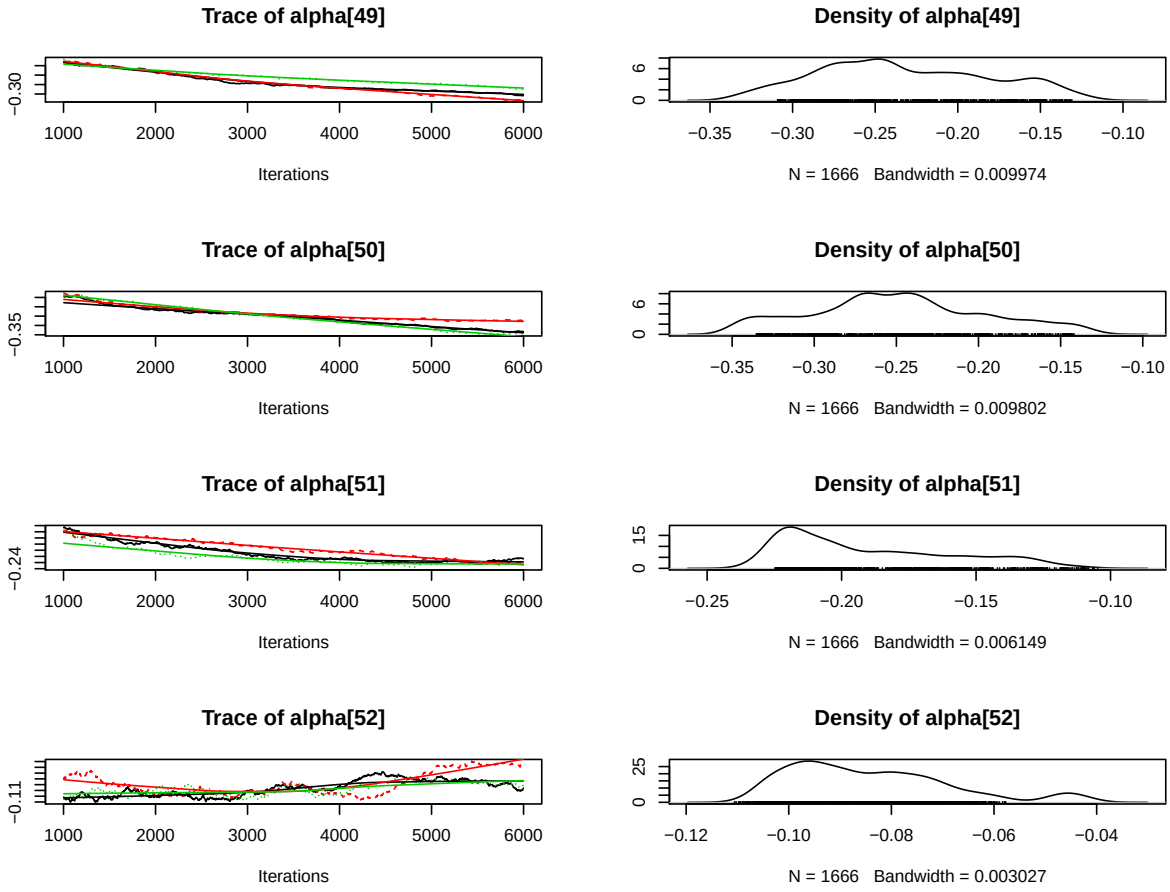

Trace of alpha[43]
−2 3
1000
2000
3000
4000
5000
6000
Iterations

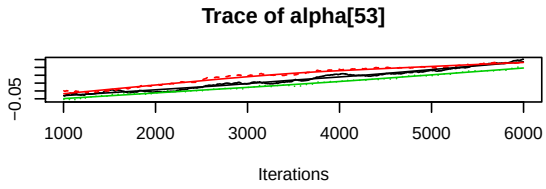

Density of alpha[43]
0.00
−2
0
2
4
N = 1666 Bandwidth = 0.3417

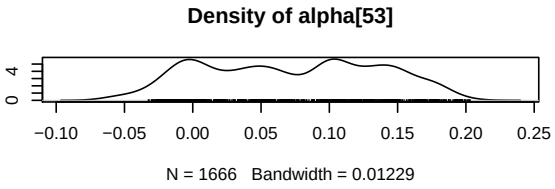

Trace of alpha[44]
−4 2
1000
2000
3000
4000
5000
6000
Iterations

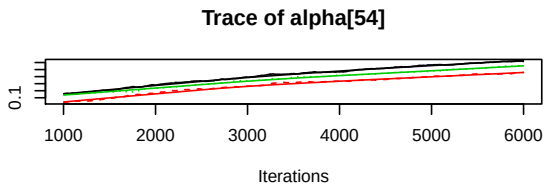

Density of alpha[44]
0.00
−4
−2
0
2
4
N = 1666 Bandwidth = 0.336

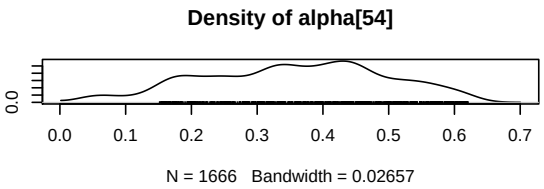

Trace of alpha[45]
−6 4
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[45]
0.00
−10 −5 0 5 10
N = 1666 Bandwidth = 0.837
Trace of alpha[46]
−10 10
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[46]
0.00
−10 −5 0 5 10
N = 1666 Bandwidth = 1.055
Trace of alpha[47]
−15 10
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[47]
0.00
−15 −10 −5 0 5 10 15
N = 1666 Bandwidth = 1.14
Trace of alpha[48]
−10
1000 2000 3000 4000 5000 6000
Iterations
Density of alpha[48]
0.00
−10 0 10 20
N = 1666 Bandwidth = 1.1

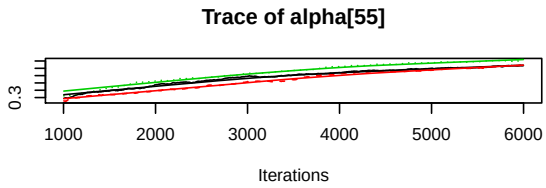

Trace of alpha[49]
−0.30
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[49]
0
6
−0.35
−0.30
−0.25
−0.20
−0.15
−0.10
N = 1666 Bandwidth = 0.009974
Trace of alpha[50]
−0.35
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[50]
0
6
−0.35
−0.30
−0.25
−0.20
−0.15
−0.10
N = 1666 Bandwidth = 0.009802
Trace of alpha[51]
−0.24
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[51]
0
15
−0.25
−0.20
−0.15
−0.10
N = 1666 Bandwidth = 0.006149
Trace of alpha[52]
−0.11
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[52]
0
25
−0.12
−0.10
−0.08
−0.06
−0.04
N = 1666 Bandwidth = 0.003027

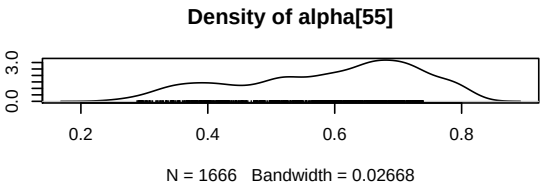

Trace of alpha[53]
−0.05
1000
2000
3000
4000
5000
6000
Iterations

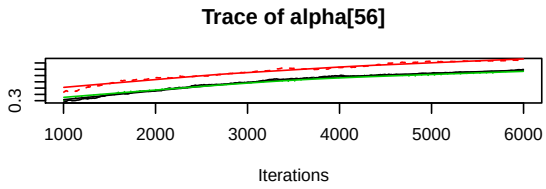

Density of alpha[53]
0
4
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20
0.25
N = 1666 Bandwidth = 0.01229

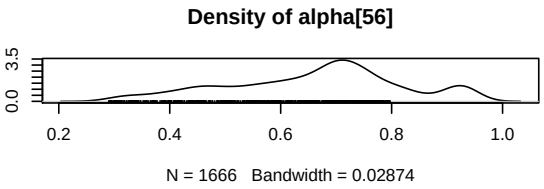

Trace of alpha[54]
0.1
1000
2000
3000
4000
5000
6000
Iterations

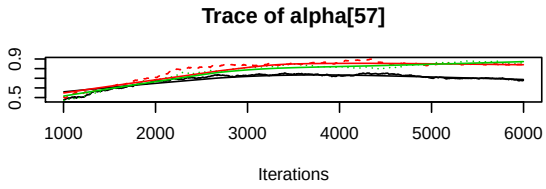

Density of alpha[54]
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
N = 1666 Bandwidth = 0.02657

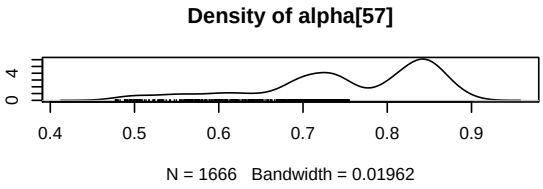

Trace of alpha[55]
0.3
1000
2000
3000
4000
5000
6000
Iterations

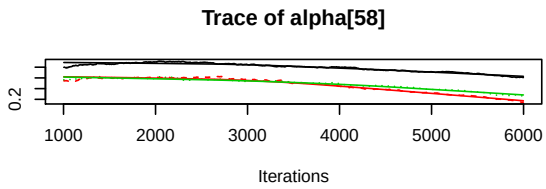

Density of alpha[55]
0.0
3.0
0.2
0.4
0.6
0.8
N = 1666 Bandwidth = 0.02668

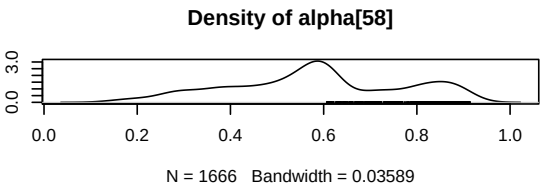

Trace of alpha[56]
0.3
1000
2000
3000
4000
5000
6000
Iterations

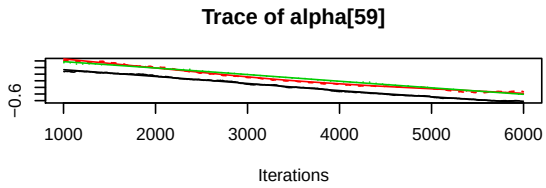

Density of alpha[56]
0.0
3.5
0.2
0.4
0.6
0.8
1.0
N = 1666 Bandwidth = 0.02874

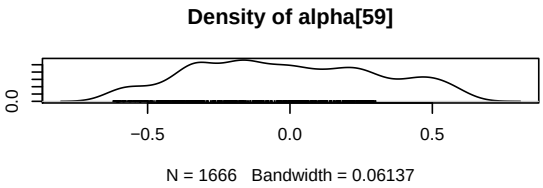

Trace of alpha[57]
0.5
0.9
1000
2000
3000
4000
5000
6000
Iterations

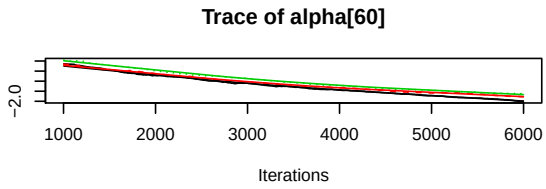

Density of alpha[57]
0
4
0.4
0.5
0.6
0.7
0.8
0.9
N = 1666 Bandwidth = 0.01962

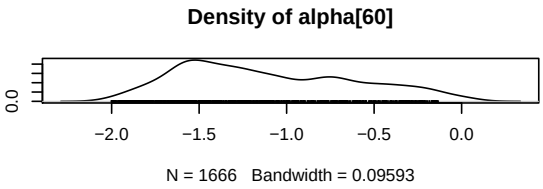

Trace of alpha[58]
0.2
1000
2000
3000
4000
5000
6000
Iterations

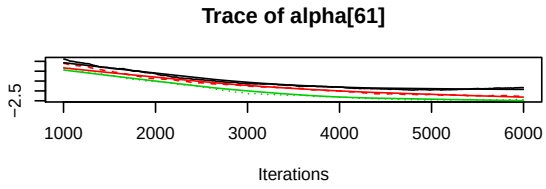

Density of alpha[58]
0.0
3.0
0.0
0.2
0.4
0.6
0.8
1.0
N = 1666 Bandwidth = 0.03589

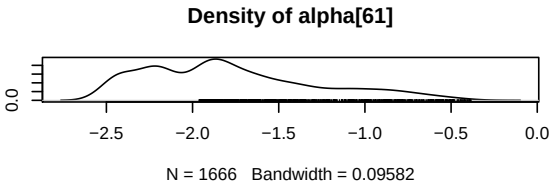

Trace of alpha[59]
−0.6
1000
2000
3000
4000
5000
6000
Iterations

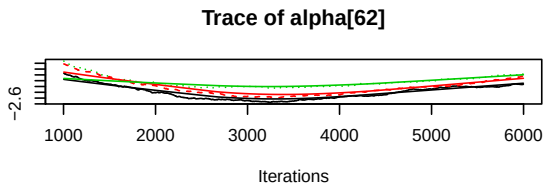

Density of alpha[59]
0.0
−0.5
0.0
0.5
N = 1666 Bandwidth = 0.06137

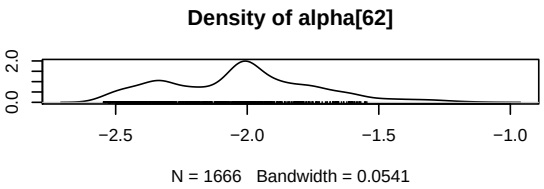

Trace of alpha[60]
−2.0
1000
2000
3000
4000
5000
6000
Iterations

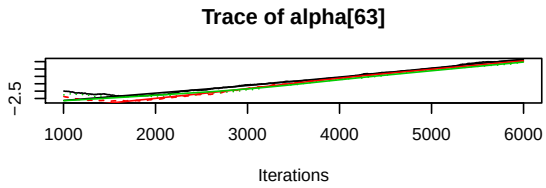

Density of alpha[60]
0.0
−2.0
−1.5
−1.0
−0.5
0.0
N = 1666 Bandwidth = 0.09593

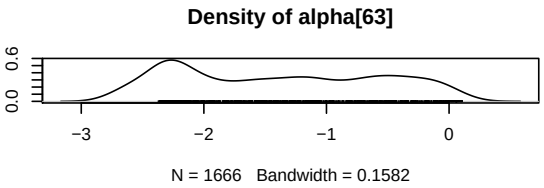

Trace of alpha[61]
−2.5
1000
2000
3000
4000
5000
6000
Iterations

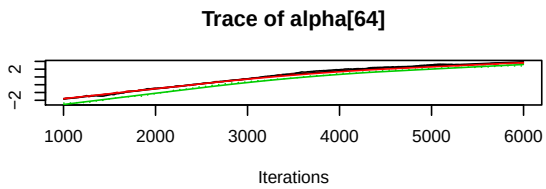

Density of alpha[61]
0.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
N = 1666 Bandwidth = 0.09582

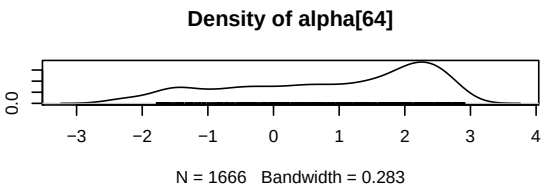

Trace of alpha[62]
−2.6
1000
2000
3000
4000
5000
6000
Iterations

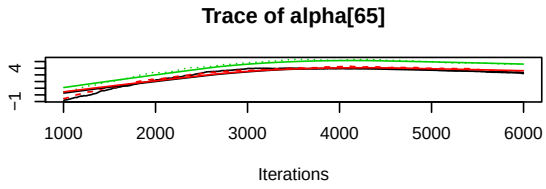

Density of alpha[62]
0.0
2.0
−2.5
−2.0
−1.5
−1.0
N = 1666 Bandwidth = 0.0541

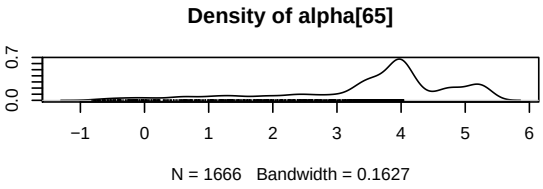

Trace of alpha[63]
−2.5
1000
2000
3000
4000
5000
6000
Iterations

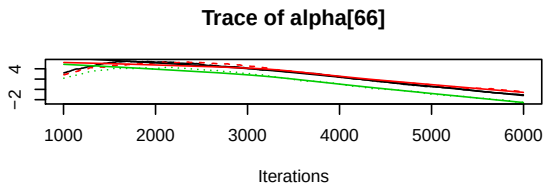

Density of alpha[63]
0.0
0.6
−3
−2
−1
0
N = 1666 Bandwidth = 0.1582

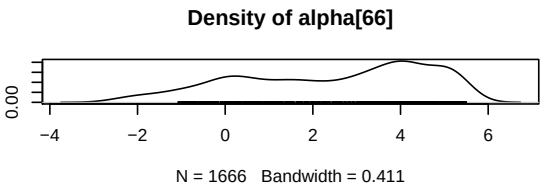

Trace of alpha[64]
−2
2
1000
2000
3000
4000
5000
6000
Iterations

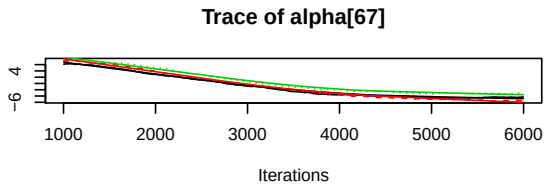

Density of alpha[64]
0.0
−3
−2
−1
0
1
2
3
4
N = 1666 Bandwidth = 0.283

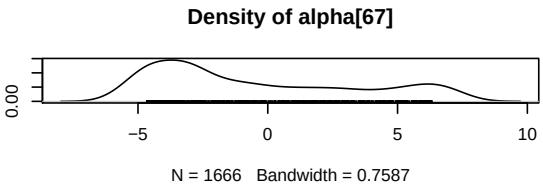

Trace of alpha[65]
−1 4
1000 2000 3000 4000 5000 6000
Iterations

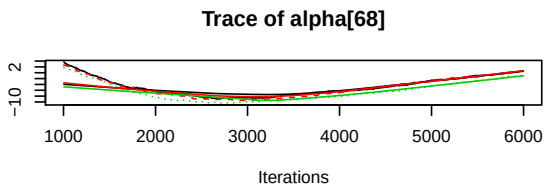

Density of alpha[65]
0.0 0.7
−1 0 1 2 3 4 5 6
N = 1666 Bandwidth = 0.1627

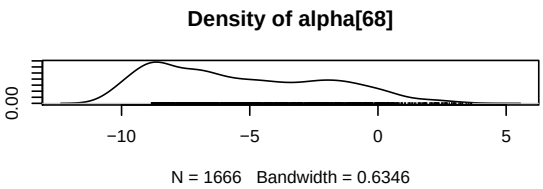

Trace of alpha[66]
−2 4
1000 2000 3000 4000 5000 6000
Iterations

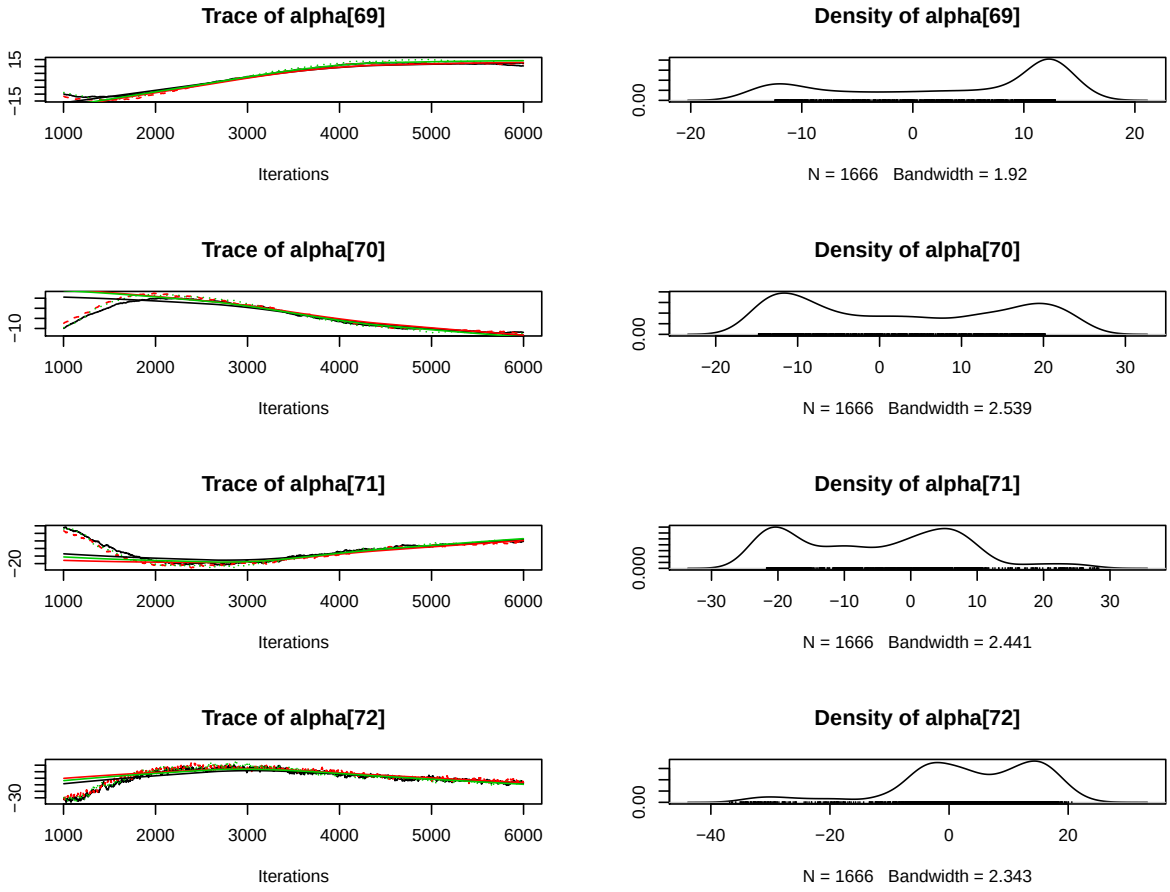

Density of alpha[66]
0.00
−4 −2 0 2 4 6
N = 1666 Bandwidth = 0.411

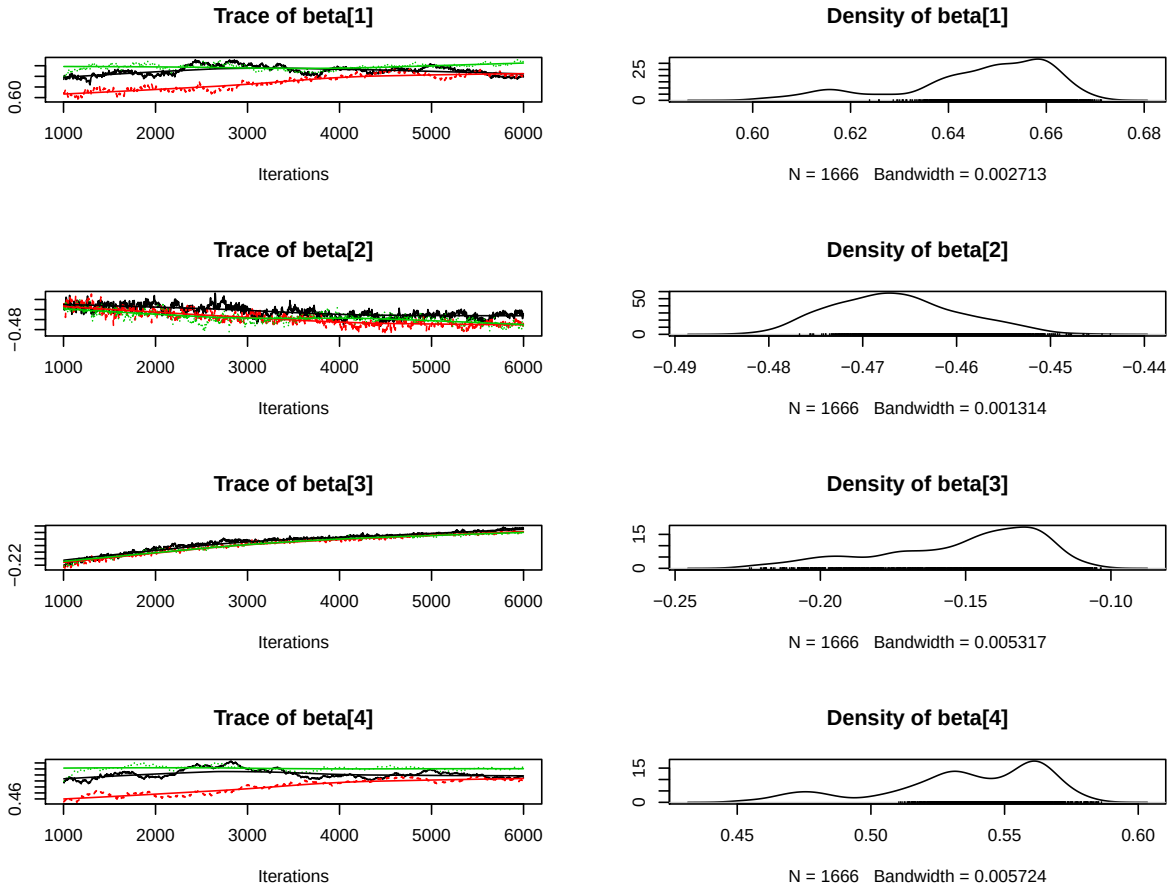

Trace of alpha[67]
−6 4
1000 2000 3000 4000 5000 6000
Iterations

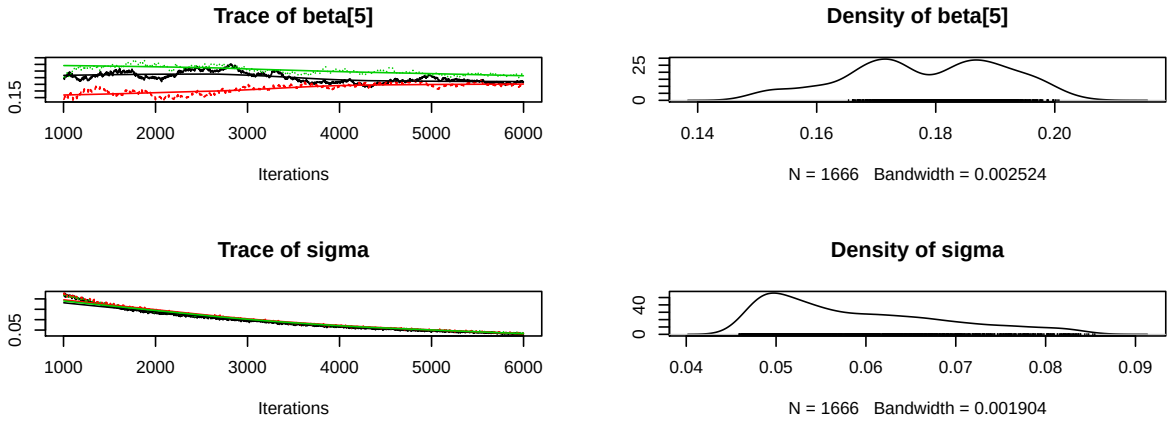

Density of alpha[67]
0.00
−5 0 5 10
N = 1666 Bandwidth = 0.7587


Trace of alpha[68]
−10 2
1000 2000 3000 4000 5000 6000
Iterations


Density of alpha[68]
0.00
−10 −5 0 5
N = 1666 Bandwidth = 0.6346


Trace of alpha[69]
−15
15
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[69]
0.00
−20
−10
0
10
20
N = 1666 Bandwidth = 1.92
Trace of alpha[70]
−10
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[70]
0.00
−20
−10
0
10
20
30
N = 1666 Bandwidth = 2.539
Trace of alpha[71]
−20
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[71]
0.000
−30
−20
−10
0
10
20
30
N = 1666 Bandwidth = 2.441
Trace of alpha[72]
−30
1000
2000
3000
4000
5000
6000
Iterations
Density of alpha[72]
0.00
−40
−20
0
20
N = 1666 Bandwidth = 2.343


Trace of beta[1]
0.60
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[1]
0
25
0.60
0.62
0.64
0.66
0.68
N = 1666 Bandwidth = 0.002713
Trace of beta[2]
−0.48
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[2]
0
50
−0.49
−0.48
−0.47
−0.46
−0.45
−0.44
N = 1666 Bandwidth = 0.001314
Trace of beta[3]
−0.22
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[3]
0
15
−0.25
−0.20
−0.15
−0.10
N = 1666 Bandwidth = 0.005317
Trace of beta[4]
0.46
1000
2000
3000
4000
5000
6000
Iterations
Density of beta[4]
0
15
0.45
0.50
0.55
0.60
N = 1666 Bandwidth = 0.005724




```
## Save the coefficients from the model
coefs = data.frame( summary(samp)$statistics )

## Save the lambda estimates
#var.e4
#var.u4

## Extract the parameter estimates and order them c(fixed, random)
params = c( coefs$Mean[ row.names(coefs) %in% paste0( paste0("beta[",1:5),"]") ],
            coefs$Mean[ row.names(coefs) %in% paste0( paste0("alpha[",1:72),"]") ] )

## Calculate the fitted values
dat$Y_Smooth_Fitted_JAGS_Spline = c(t(params) %*% t(cbind(1,
                                                          G.2,
                                                          G.3,
                                                          X,
                                                          X_2,
                                                          random.basis.1,
                                                          random.basis.2,
                                                          random.basis.3)) )
rm(coefs, model, samp, model_string, random.basis.1, random.basis.2,
   random.basis.3, n, G.2, G.3, K, params)
```

Figure 7: The group-level mean functions using penalized splines and JAGS.

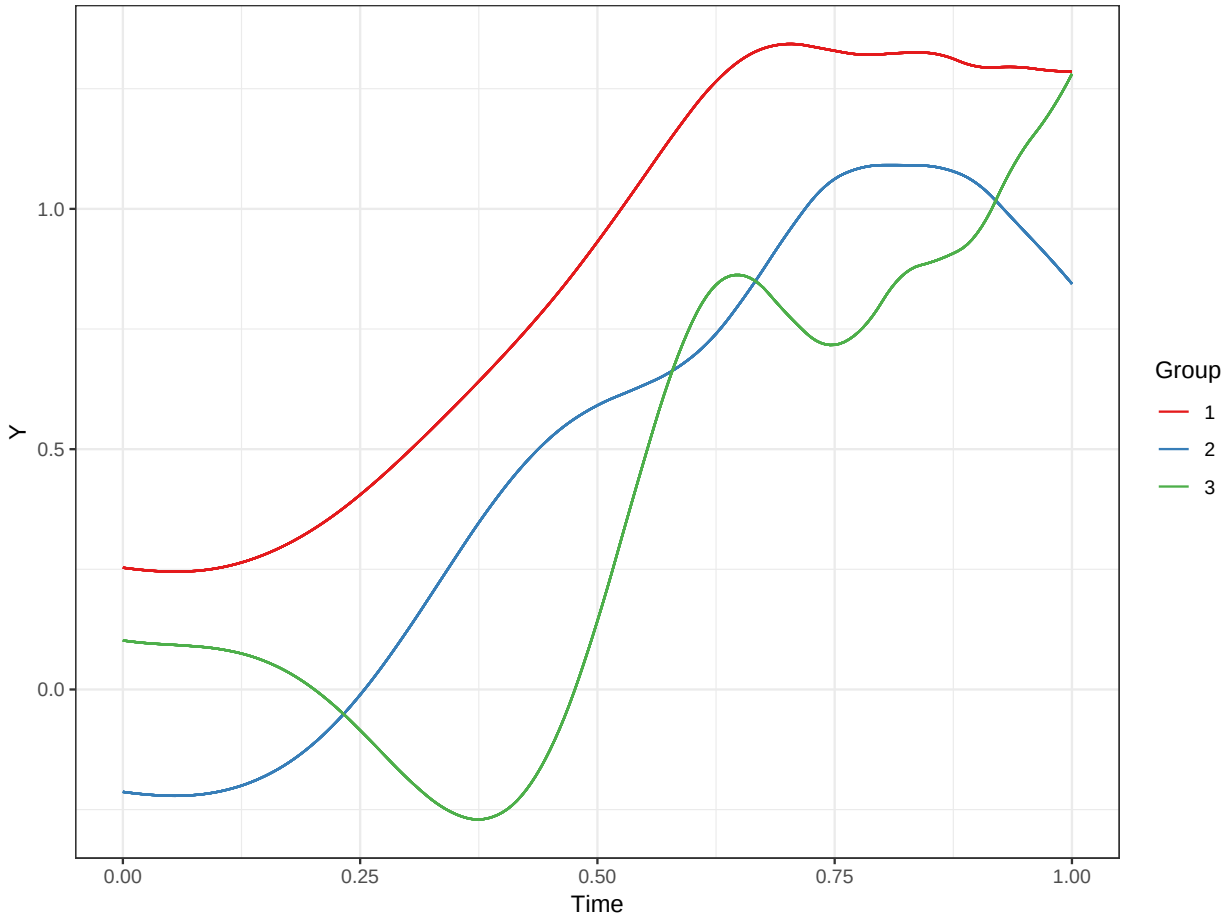
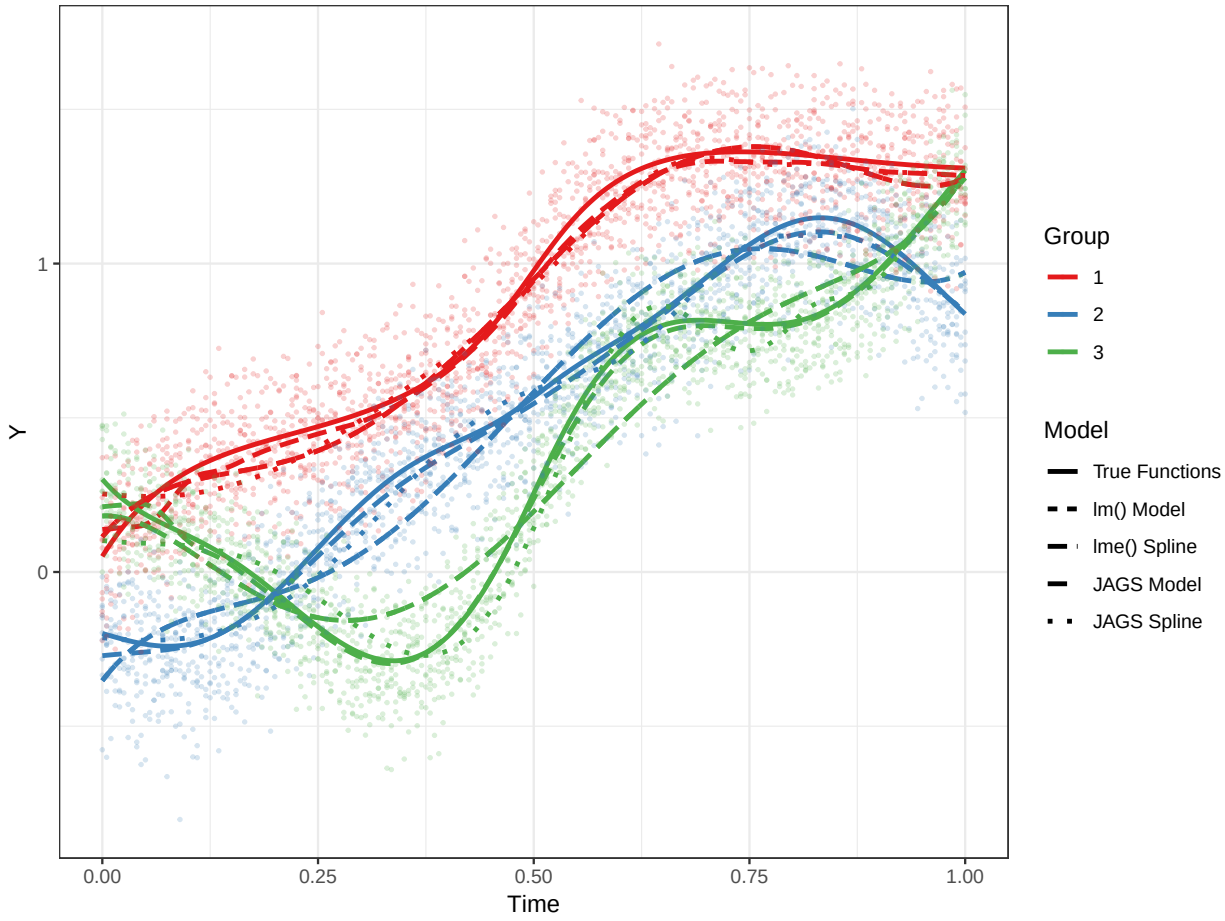


Figure 8: A comparison of the fitted effect functions from each model.



Session Info

A summary of the R session used for the analysis.

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 7 x64 (build 7601) Service Pack 1
##
## Matrix products: default
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] rjags_4-8          coda_0.19-2        lme4_1.1-18-1
## [4] Matrix_1.2-14      nlme_3.1-137       RColorBrewer_1.1-2
## [7] ggplot2_3.1.0      knitr_1.20         MASS_7.3-51
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.19     nloptr_1.2.1     pillar_1.3.0     compiler_3.5.1
##  [5] plyr_1.8.4       bindr_0.1.1      tools_3.5.1      digest_0.6.18
##  [9] evaluate_0.12    tibble_1.4.2     gtable_0.2.0     lattice_0.20-35
## [13] pkgconfig_2.0.2  rlang_0.3.0.1    rstudioapi_0.8   yaml_2.2.0
## [17] xfun_0.4         bindrcpp_0.2.2   stringr_1.3.1    withr_2.1.2
## [21] dplyr_0.7.7      rprojroot_1.3-2  grid_3.5.1       tidyselect_0.2.5
## [25] glue_1.3.0       R6_2.3.0         rmarkdown_1.10   minqa_1.2.4
## [29] purrr_0.2.5      magrittr_1.5     splines_3.5.1    backports_1.1.2
## [33] scales_1.0.0     htmltools_0.3.6  rsconnect_0.8.8  assertthat_0.2.0
## [37] colorspace_1.3-2 labeling_0.3     tinytex_0.9      stringi_1.2.4
## [41] lazyeval_0.2.1   munsell_0.5.0    crayon_1.3.4
```