Hierarchical Compartmental reserving model using PK/PD

Hierarchical Compartmental Models

Similar to a PK/PD model, Jake describes the state-space process of claims developments with a set of ordinary differential equations (ODE), using EX= Exposure, OS= Outstanding claims, PD= Paid claims (these are the compartments):

dEX/dt=−ker⋅EXdOS/dt=ker⋅RLR⋅EX−kp⋅OSdPD/dt=kp⋅RRF⋅OS

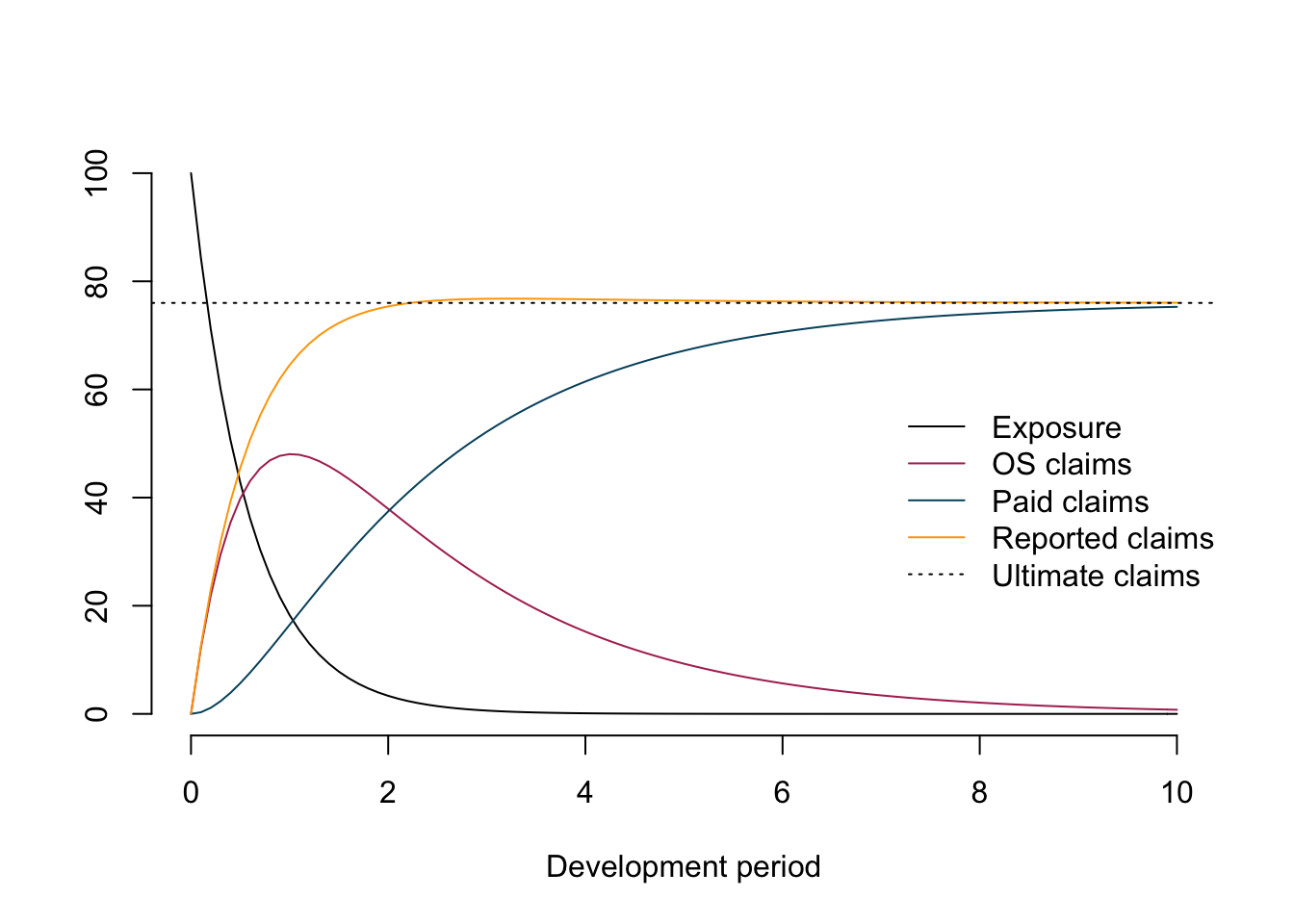
with initial conditions EX(0)=Π (ultimate earned premiums), OS(0)=0,PD(0)=0.

The parameters describe:

* ker the rate at which claim events occur and are subsequently reported to the insurer
* RLR the reported loss ratio
* RRF the reserve robustness factor, the proportion of outstanding claims that eventually are paid
* kp the rate of payment, i.e. the rate at which outstanding claims are paid

Note, the product of RLR and RRF describes the ultimate loss ratio ULR= ultimate loss / premium, which is a key metric in insurance to assess performance and profitability.

Setting the parameters ker=1.7, RLR=0.8, kp=0.5, RRF=0.95 produces the following output.



The chart illustrates nicely the different compartments (processes) for a policy over time.

The autonomous system of ODEs above can be solved analytically:

EX(t)=Π⋅exp⁡(−kert)OS(t)=Π⋅RLR⋅kerker−kp⋅(exp⁡(−kpt)−exp⁡(−kert))PD(t)=Π⋅RLR⋅RRFker−kp(ker⋅(1−exp⁡(−kpt)−kp⋅(1−exp⁡(−kert))

In this post I will focus on the last two equations, as they describe the observable data of outstanding claims and cumulative paid claims over time. However, one could expand the model, e.g. to allow for different premium earning patterns.

Get example data

In his paper Jake uses data from the CAS Reserving Database, namely company 337 from the worker’s compensation file.

The following function downloads the data and reshapes it into a more useful format.

**library**(data.table)

CASdata <- fread("http://www.casact.org/research/reserve\_data/wkcomp\_pos.csv")

createLossData2 <- **function**(CASdata, company\_code){

compData <- CASdata[GRCODE==company\_code,

c("EarnedPremDIR\_D", "AccidentYear", "DevelopmentLag",

"IncurLoss\_D", "CumPaidLoss\_D", "BulkLoss\_D")]

setnames(compData, names(compData),

c("premium", "accident\_year", "dev",

"incurred\_loss", "paid\_loss", "bulk\_loss"))

compData <- compData[, `:=`(

origin = accident\_year - min(accident\_year) + 1)]

compData0 <- compData[dev==1]

compData0 <- compData0[, `:=`(dev = 0, incurred\_loss = 0,

paid\_loss = 0, bulk\_loss = 0)]

compData <- rbindlist(list(compData0, compData))

compData <- compData[, cal := origin + dev - 1][order(origin, dev)]

compData <- compData[, `:=`(

paid\_train = ifelse(cal <= max(origin), paid\_loss, NA),

paid\_test = ifelse(cal > max(origin), paid\_loss, NA),

os\_train = ifelse(cal <= max(origin), incurred\_loss - paid\_loss, NA),

os\_test = ifelse(cal > max(origin), incurred\_loss - paid\_loss, NA))]

traintest <- rbindlist(list(compData[cal <= max(origin)],

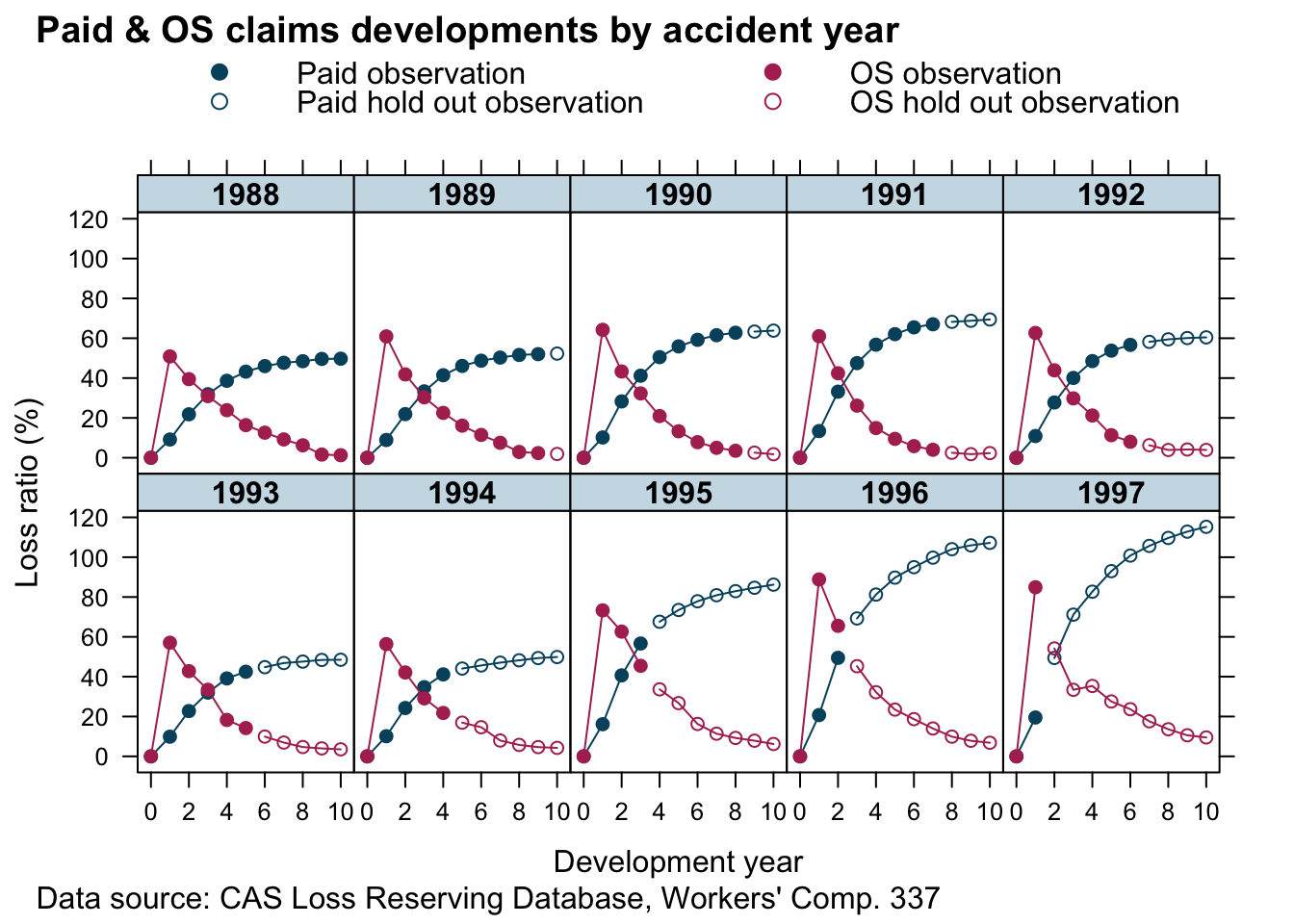
compData[cal > max(origin)]))

**return**(traintest)

}

lossData <- createLossData2(CASdata, company\_code = 337)

Let’s plot the data of outstanding and paid loss ratio against development years (t=dev) for the different accident years.



The data looks similar to the example output from our model above. That’s a good start.

Re-organising data

In order to fit the two curves simultaneously I use a little trick. I stack the paid and outstanding claims into a single column and add another indicator column (delta), which is either 0 for outstanding claims or 1 for paid claims. This turns the multivariate data back into a univariate set.

Additionally, I remove the artificial data for dev=0 again, add another column of the indicator variable as a factor (deltaf) and divide all measurement variables by the premium of the oldest accident year. The last step normalises the data and should make modelling a little easier as everything is in the ballpark of 1.

lossData0 <-

rbindlist(

list(lossData[, list(accident\_year, dev, loss\_train=os\_train,

loss\_test=os\_test, delta=0, premium=premium)],

lossData[,list(accident\_year, dev,loss\_train=paid\_train,

loss\_test=paid\_test, delta=1, premium=premium)]

))[order(accident\_year, dev)]

premind <- lossData0[accident\_year==min(accident\_year) & dev==min(dev) & delta==1, premium]

lossData0 <- lossData0[, `:=`(premium = premium/premind,

loss\_train = loss\_train/premind,

loss\_test = loss\_test/premind,

deltaf = factor(delta, labels = c("os", "paid")),

cal=accident\_year + dev - 1)][dev>0]

Model fitting

Non-linear Least Squares

Before I build a complex Bayesian model I start with a simple non-linear least squares model. Or in other words, I believe the data is generated from a Normal distribution, with the mean described by an analytical function μ(t)=f(t,…) and constant variance σ2.

y(t)∼N(μ(t),σ2)μ(t)=f(t,Π,ker,kp,RLR,RRF,δ)=Π⋅[(1−δ)RLR⋅kerker−kp⋅(exp⁡(−kpt)−exp⁡(−kert))+δRLR⋅RRFker−kp(ker⋅(1−exp⁡(−kpt)−kp⋅(1−exp⁡(−kert))]δ={0 if y is outstanding claim1 if y is paid claim

To ensure all parameters stay positive I will use the same approach as Jake does in this paper, that is I reparameterize using the logarithm of the parameters.

my.f <- **function**(t, premium, lk\_er, lk\_p, lRLR, lRRF, delta){

k\_er <- exp(lk\_er)

k\_p <- exp(lk\_p)

RLR <- exp(lRLR)

RRF <- exp(lRRF)

os <- (RLR \* k\_er / (k\_er - k\_p) \* (exp(-k\_p \* t) - exp(-k\_er \* t)))

paid <- (RLR \* RRF / (k\_er - k\_p) \* (k\_er \* (1 - exp(-k\_p \* t)) -

k\_p \* (1 - exp(-k\_er \* t))))

**return**(premium \* (os \* (1 - delta) + paid \* delta))

}

Using the nls function in R the model can be fitted quickly.

n1 <- nls(loss\_train ~ my.f(dev, premium,

lk\_er=lker, lk\_p=lkp,

lRLR=lRLR, lRRF=lRRF, delta=delta),

data=lossData0[cal<=max(accident\_year)],

start = c(lker = log(1.5), lRLR = log(1),

lkp = log(0.75), lRRF = log(0.75)))

n1

**#***# Nonlinear regression model*

**#***# model: loss\_train ~ my.f(dev, premium, lk\_er = lker, lk\_p = lkp, lRLR = lRLR, lRRF = lRRF, delta = delta)*

**#***# data: lossData0[cal <= max(accident\_year)]*

**#***# lker lRLR lkp lRRF*

**#***# 0.8621 -0.1090 -0.8646 -0.4397*

**#***# residual sum-of-squares: 0.3505*

**#***#*

**#***# Number of iterations to convergence: 5*

**#***# Achieved convergence tolerance: 2.844e-06*

Let’s bring the coefficients back to the original scale:

n1par <- data.table(summary(n1)$coefficients)[, exp(Estimate + 0.5\*`Std. Error`^2)]

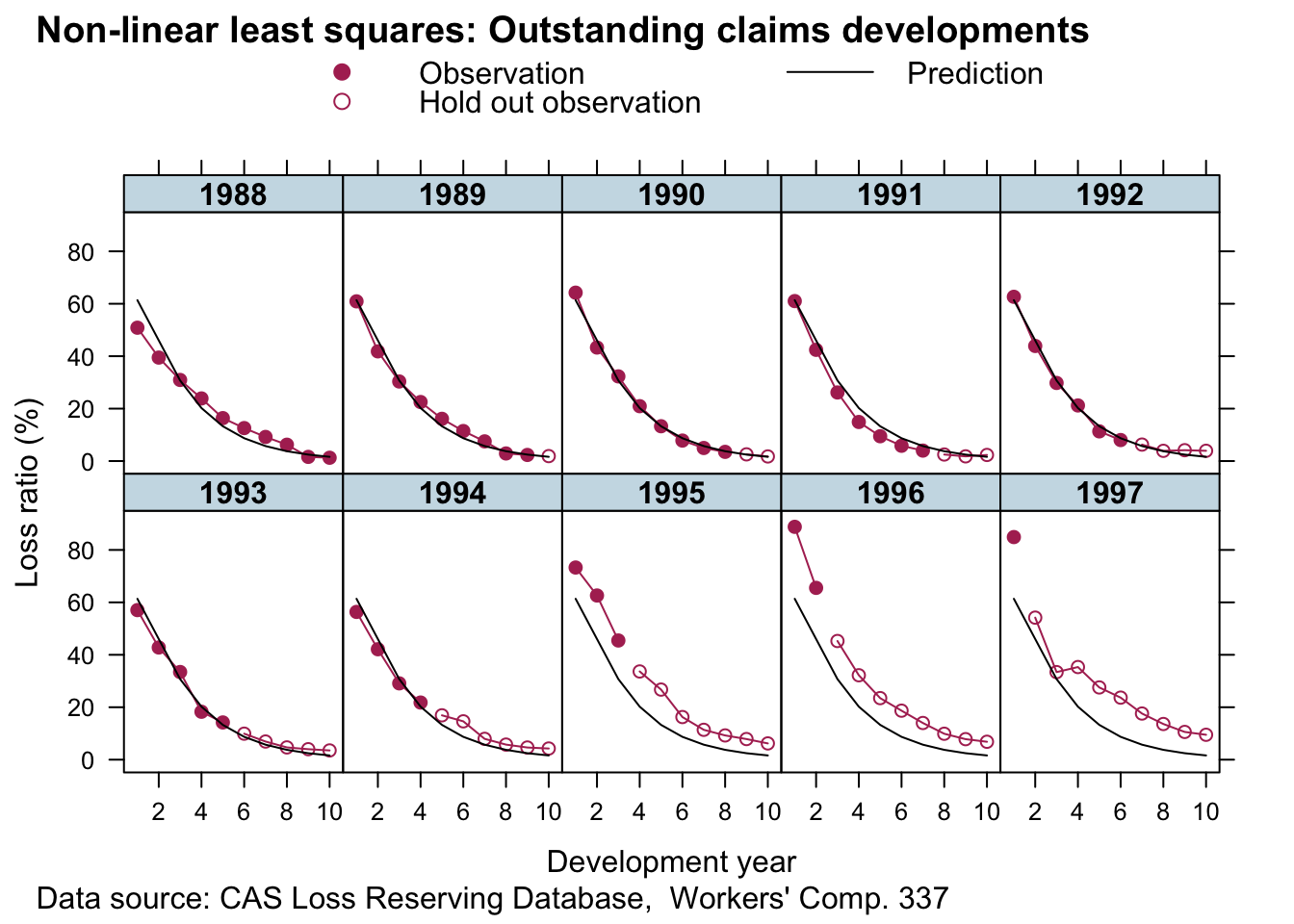
names(n1par) <- c("ker", "RLR", "kp", "RRF")

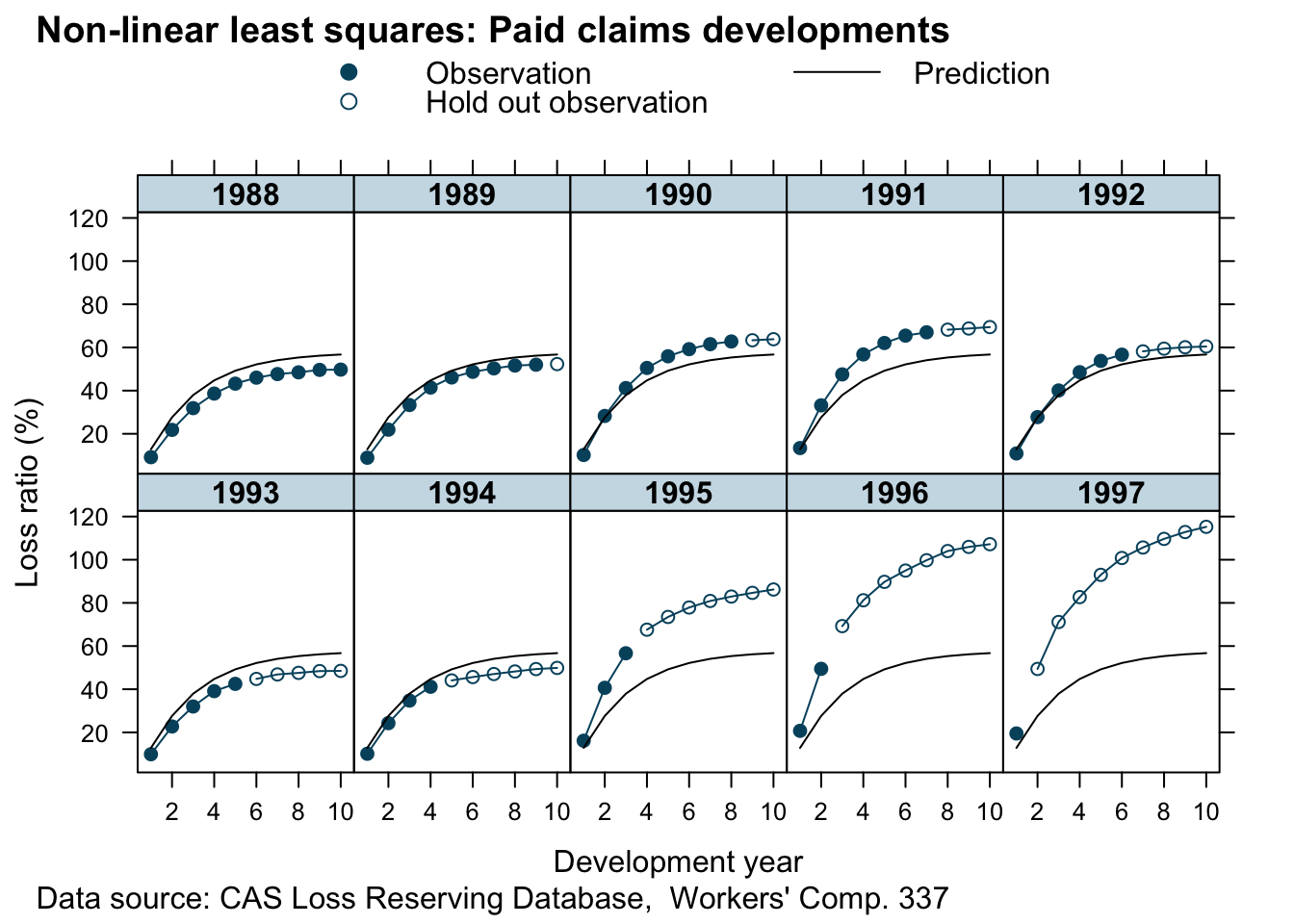
n1par

## **ker** **RLR** **kp** **RRF**

## 2.4375609 0.8985600 0.4231800 0.6466241

Assuming a constant variance across outstanding (delta=0) and paid claims (delta=1) doesn’t really make much sense, neither that the parameters RLR and RRF are the same across all accident years, which assumes the same ULR for all years: 57.8%.





Hierarchical non-linear model

To address the issues mentioned above, I allow for different variances depending on the compartment type (σy[δ]2) and allow the parameters RLR and RRF to vary across accident years.

y(t)∼N(f(t,Π,ker,kp,RLR[i],RRF[i]),σy[δ]2)(RLR[i]RRF[i])∼N((μRLRμRRF),(σRLR200σRRF2))

The parameters μRLR,μRRF describe the underlying means across all years i.

With the preparation done, I can fit the hierarchical non-linear model using the nlme package, assuming that the parameters lRLR and lRRF vary across accident years (random effects), while I treat the parameters lker and lkp as fixed effects.

**library**(nlme)

m1 <- nlme(loss\_train ~ my.f(dev, premium,

lk\_er=lker, lk\_p=lkp,

lRLR=lRLR, lRRF=lRRF, delta=delta),

data=lossData0[cal<=max(accident\_year)],

fixed = lker + lRLR + lkp + lRRF ~ 1,

random = pdDiag(lRLR + lRRF ~ 1),

groups = ~ accident\_year,

weights = varIdent(form = ~ 1 | deltaf),

start = c(lker = log(1.5), lRLR = log(1),

lkp = log(0.75), lRRF = log(0.75)),

control=list(msMaxIter=10000, pnlsMaxIter=10000,

pnlsTol=0.4))

summary(m1)

**#***# Nonlinear mixed-effects model fit by maximum likelihood*

**#***# Model: loss\_train ~ my.f(dev, premium, lk\_er = lker, lk\_p = lkp, lRLR = lRLR, lRRF = lRRF, delta = delta)*

**#***# Data: lossData0[cal <= max(accident\_year)]*

**#***# AIC BIC logLik*

**#***# -524.4347 -502.8309 270.2174*

**#***#*

**#***# Random effects:*

**#***# Formula: list(lRLR ~ 1, lRRF ~ 1)*

**#***# Level: accident\_year*

**#***# Structure: Diagonal*

**#***# lRLR lRRF Residual*

**#***# StdDev: 0.186949 0.1318405 0.03337559*

**#***#*

**#***# Variance function:*

**#***# Structure: Different standard deviations per stratum*

**#***# Formula: ~1 | deltaf*

**#***# Parameter estimates:*

**#***# os paid*

**#***# 1.0000000 0.1805809*

**#***# Fixed effects: lker + lRLR + lkp + lRRF ~ 1*

**#***# Value Std.Error DF t-value p-value*

**#***# lker 0.4102733 0.05624434 97 7.294481 0.0000*

**#***# lRLR 0.0225969 0.06751438 97 0.334698 0.7386*

**#***# lkp -0.7946096 0.03483365 97 -22.811551 0.0000*

**#***# lRRF -0.4049580 0.05558069 97 -7.285948 0.0000*

**#***# Correlation:*

**#***# lker lRLR lkp*

**#***# lRLR -0.375*

**#***# lkp -0.928 0.388*

**#***# lRRF 0.522 -0.282 -0.572*

**#***#*

**#***# Standardized Within-Group Residuals:*

**#***# Min Q1 Med Q3 Max*

**#***# -2.63652139 -0.52065601 0.03990089 0.61348869 2.01628548*

**#***#*

**#***# Number of Observations: 110*

**#***# Number of Groups: 10*

The parameters ker, μRLR, kp, μRRF on the original scale are:

m1\_fe <- data.table(summary(m1)$tTable)[, exp(Value + 0.5\*`Std.Error`^2)]

names(m1\_fe) <- c("ker", "RLR", "kp", "RRF")

m1\_fe

## **ker** **RLR** **kp** **RRF**

## 1.5096155 1.0251880 0.4520317 0.6680359

If you look at the p-Value of lRLR, we might as well assume μRLR=1.

The estimated ULR across accident years is 68.5%, and the median by accident year:

RLRRRF <- coef(m1)[,c(2,4)]

names(RLRRRF) <- c("RLR", "RRF")

round(exp(cbind(RLRRRF, ULR=apply(RLRRRF,1,sum))),3)

## **RLR** **RRF** **ULR**

## 1988 0.853 0.593 0.506

## 1989 0.925 0.577 0.534

## 1990 0.968 0.673 0.651

## 1991 0.910 0.798 0.727

## 1992 0.946 0.663 0.627

## 1993 0.899 0.562 0.505

## 1994 0.885 0.611 0.540

## 1995 1.232 0.724 0.892

## 1996 1.406 0.785 1.104

## 1997 1.382 0.734 1.014

The parameters σy[δ] are:

(sig\_delta <- sapply(split(resid(m1), lossData0[cal<=max(accident\_year)]$deltaf), sd))

**#***# os paid*

**#***# 0.032065014 0.005522208*

and σRLR and σRRF are:

VarCorr(m1)

**#***# accident\_year = pdDiag(list(lRLR ~ 1,lRRF ~ 1))*

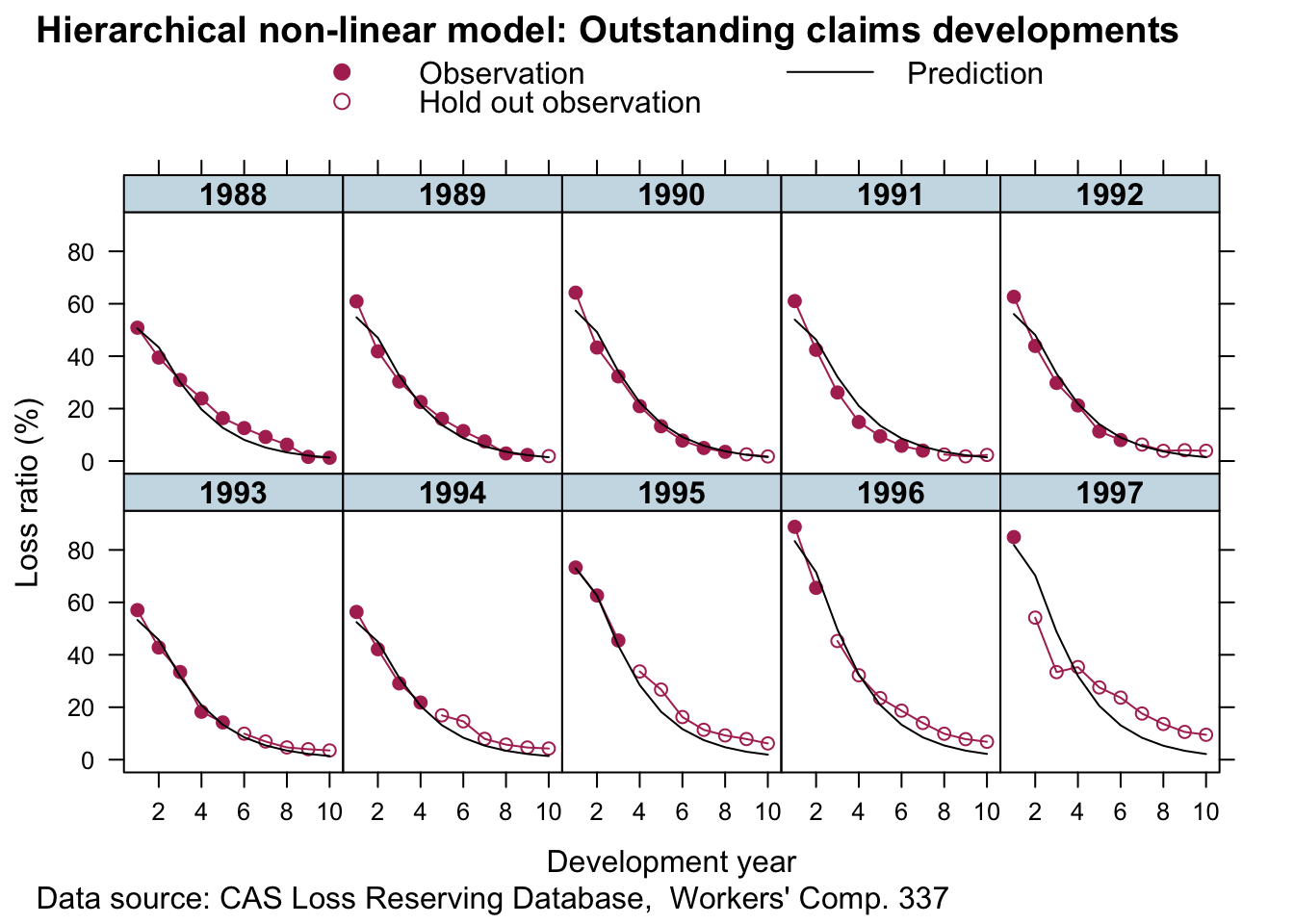
**#***# Variance StdDev*

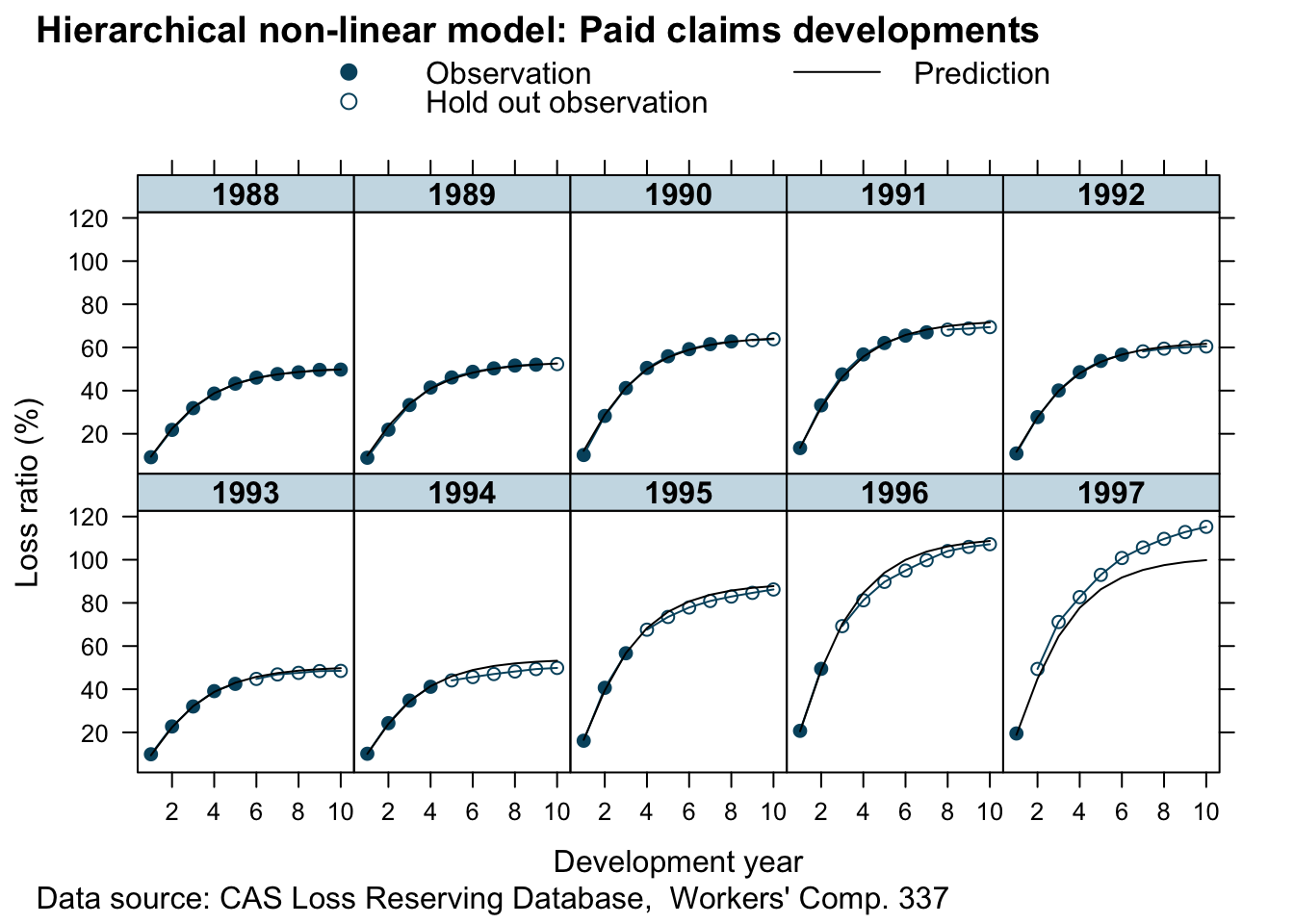
**#***# lRLR 0.03494992 0.18694896*

**#***# lRRF 0.01738192 0.13184052*

**#***# Residual 0.00111393 0.03337559*

Let’s looks at the predictions:





Looks good. This is a huge improvement to the simple non-linear least squares model. For all years but 1997 (here we had only one data point) the predictions look pretty reasonable. The more data we have the more credibility will be given to it and shift parameters RLR and RRF away from the population mean.

Bayesian reserving models with Stan

We can fit the same model with Stan, which allows us to make different assumptions on the data generating distribution and specify the priors more flexibly. For example, I’d like to assume Gamma priors over my parameters, this will ensure the parameters are all positive and I don’t have to reparameterise them as I did above.

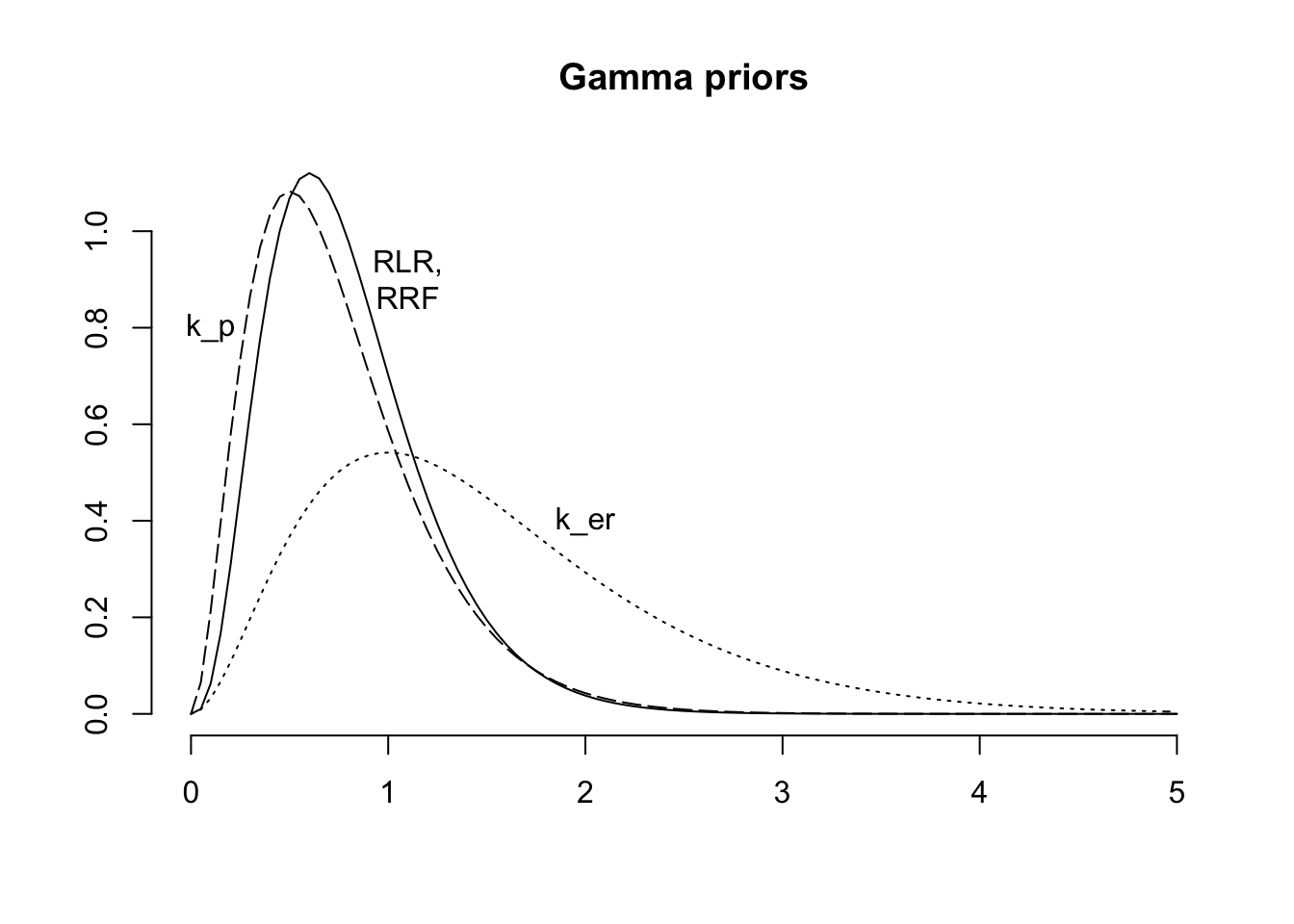
curve(dgamma(x, 4, 5), from = 0, 5, main="Gamma priors",

xlab="", ylab="", bty="n")

curve(dgamma(x, 3, 2), add = TRUE, lty=3)

curve(dgamma(x, 3, 4), add=TRUE, lty=5)

text(x = c(0.1, 1.1, 2), y = c(0.8, 0.9, 0.4), labels = c("k\_p", "RLR,\nRRF", "k\_er"))



Next I specify my model with brm, which will not only generate the Stan code, but also run the model and collect all the results. I continue to use a Gaussian distribution, as it will allow me to compare the output of brm with nlme.

**library**(rstan)

**library**(brms)

rstan\_options(auto\_write = TRUE)

options(mc.cores = parallel::detectCores())

fml <-

loss\_train ~ premium \* (

(RLR\*ker/(ker - kp) \* (exp(-kp\*dev) - exp(-ker\*dev))) \* (1 - delta) +

(RLR\*RRF/(ker - kp) \* (ker \*(1 - exp(-kp\*dev)) - kp\*(1 - exp(-ker\*dev)))) \* delta

)

b1 <- brm(bf(fml,

RLR ~ 1 + (1 | accident\_year),

RRF ~ 1 + (1 | accident\_year),

ker ~ 1,

kp ~ 1,

sigma ~ 0 + deltaf,

nl = TRUE),

data = lossData0[cal <= max(accident\_year)],

family = brmsfamily("gaussian", link\_sigma = "log"),

prior = c(prior(gamma(4, 5), nlpar = "RLR", lb=0),

prior(gamma(4, 5), nlpar = "RRF", lb=0),

prior(gamma(3, 2), nlpar = "ker", lb=0),

prior(gamma(3, 4), nlpar = "kp", lb=0)),

control = list(adapt\_delta = 0.999, max\_treedepth=15),

seed = 1234, iter = 1000)

b1

**#***# Family: gaussian*

**#***# Links: mu = identity; sigma = log*

**#***# Formula: loss\_train ~ premium \* ((RLR \* ker/(ker - kp) \* (exp(-kp \* dev) - exp(-ker \* dev))) \* (1 - delta) + (RLR \* RRF/(ker - kp) \* (ker \* (1 - exp(-kp \* dev)) - kp \* (1 - exp(-ker \* dev)))) \* delta)*

**#***# RLR ~ 1 + (1 | accident\_year)*

**#***# RRF ~ 1 + (1 | accident\_year)*

**#***# ker ~ 1*

**#***# kp ~ 1*

**#***# sigma ~ 0 + deltaf*

**#***# Data: lossData0[cal <= max(accident\_year)] (Number of observations: 110)*

**#***# Samples: 4 chains, each with iter = 1000; warmup = 500; thin = 1;*

**#***# total post-warmup samples = 2000*

**#***#*

**#***# Group-Level Effects:*

**#***# ~accident\_year (Number of levels: 10)*

**#***# Estimate Est.Error l-95% CI u-95% CI Rhat Bulk\_ESS Tail\_ESS*

**#***# sd(RLR\_Intercept) 0.25 0.08 0.14 0.45 1.01 630 954*

**#***# sd(RRF\_Intercept) 0.11 0.04 0.06 0.20 1.01 822 1178*

**#***#*

**#***# Population-Level Effects:*

**#***# Estimate Est.Error l-95% CI u-95% CI Rhat Bulk\_ESS Tail\_ESS*

**#***# RLR\_Intercept 1.02 0.09 0.83 1.19 1.01 522 573*

**#***# RRF\_Intercept 0.67 0.04 0.59 0.76 1.01 639 738*

**#***# ker\_Intercept 1.55 0.11 1.37 1.78 1.00 1021 1115*

**#***# kp\_Intercept 0.45 0.02 0.42 0.49 1.01 1144 958*

**#***# sigma\_deltafos -3.38 0.12 -3.61 -3.15 1.00 1301 1395*

**#***# sigma\_deltafpaid -5.07 0.13 -5.30 -4.80 1.00 1127 1026*

**#***#*

**#***# Samples were drawn using sampling(NUTS). For each parameter, Bulk\_ESS*

**#***# and Tail\_ESS are effective sample size measures, and Rhat is the potential*

**#***# scale reduction factor on split chains (at convergence, Rhat = 1).*

This looks similar to the output of nlme. Note, the link function for sigma was log, so to compare the output with nlme I have to transform it back to the original scale:

**library**(rstan)

X <- extract(b1$fit)

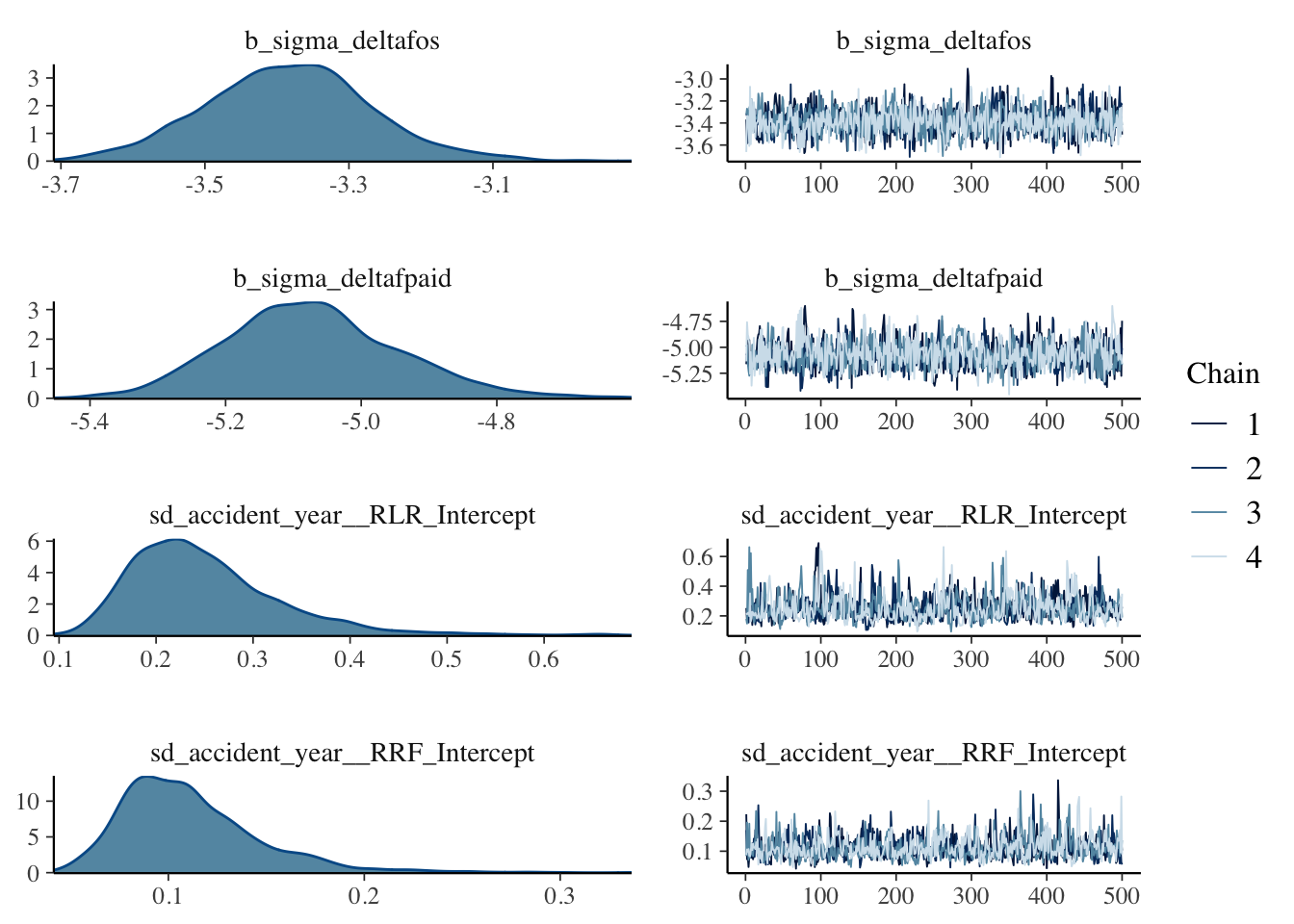
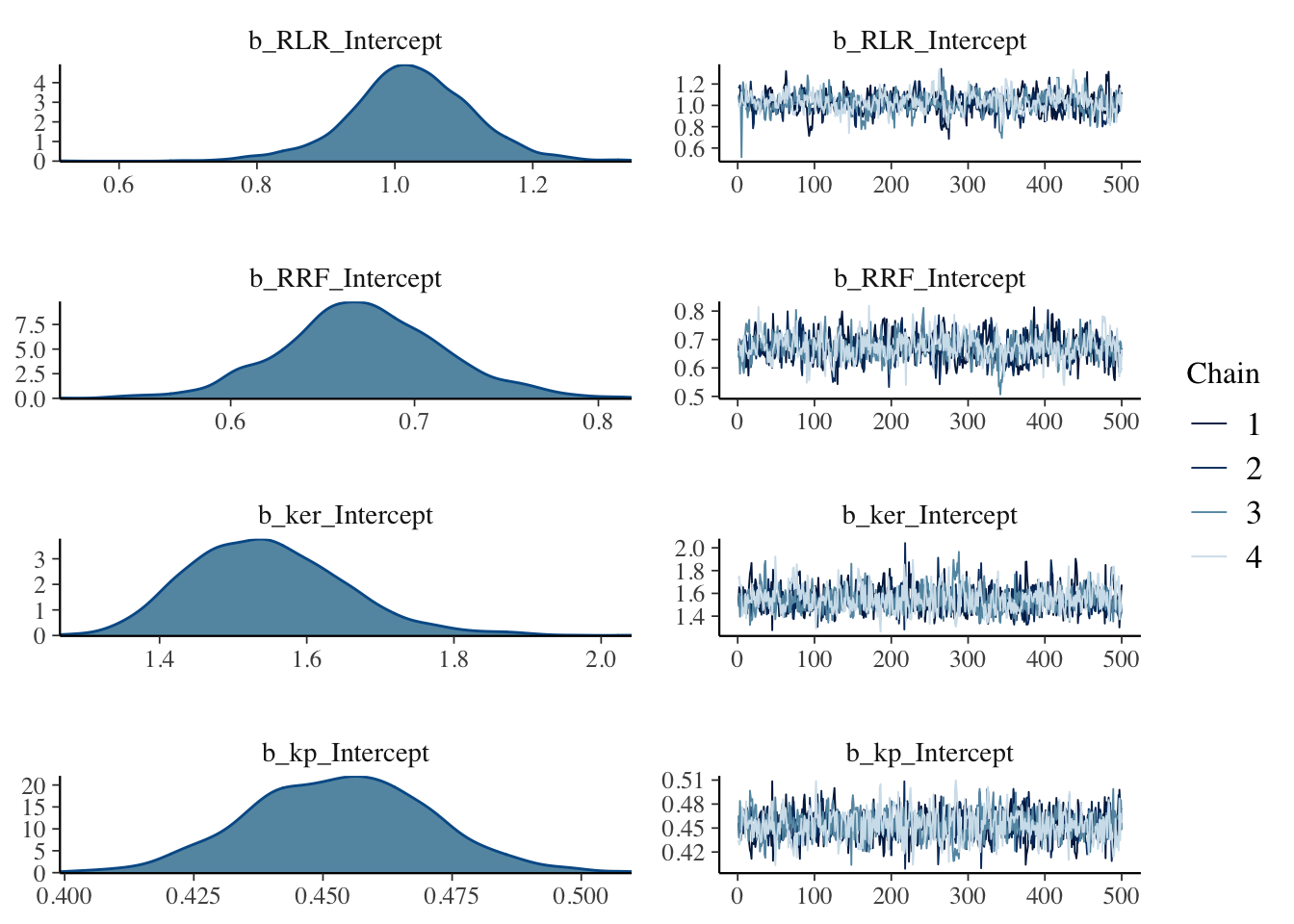
(sig\_delta\_brm <- apply(X$b\_sigma, 2, **function**(x) exp(mean(x)+0.5\*var(x))))

## [1] 0.03413337 0.00632486

Again, that’s pretty similar.

The simulations are well behaved as well:

plot(b1, N = 4, ask = FALSE)



Next I want to compare the estimated accident year ULRs with the ‘population’ level ULR.

RLR <- sweep(X$r\_1\_RLR\_1, 1, X$b\_RLR, "+")

RRF <- sweep(X$r\_2\_RRF\_1, 1, X$b\_RRF, "+")

ULR <- t(apply(RLR \* RRF, 2, quantile, c(0.025, 0.5, 0.975)))

matplot(unique(lossData0$accident\_year), ULR\*100,

t="l", ylim=c(0, 150), lty=1, col=c(1,2,4), bty="n",

ylab="Projected ULR (%)", xlab="Accident year")

legend("topleft", legend = c("2.5%ile","50%ile", "97.5%ile"),

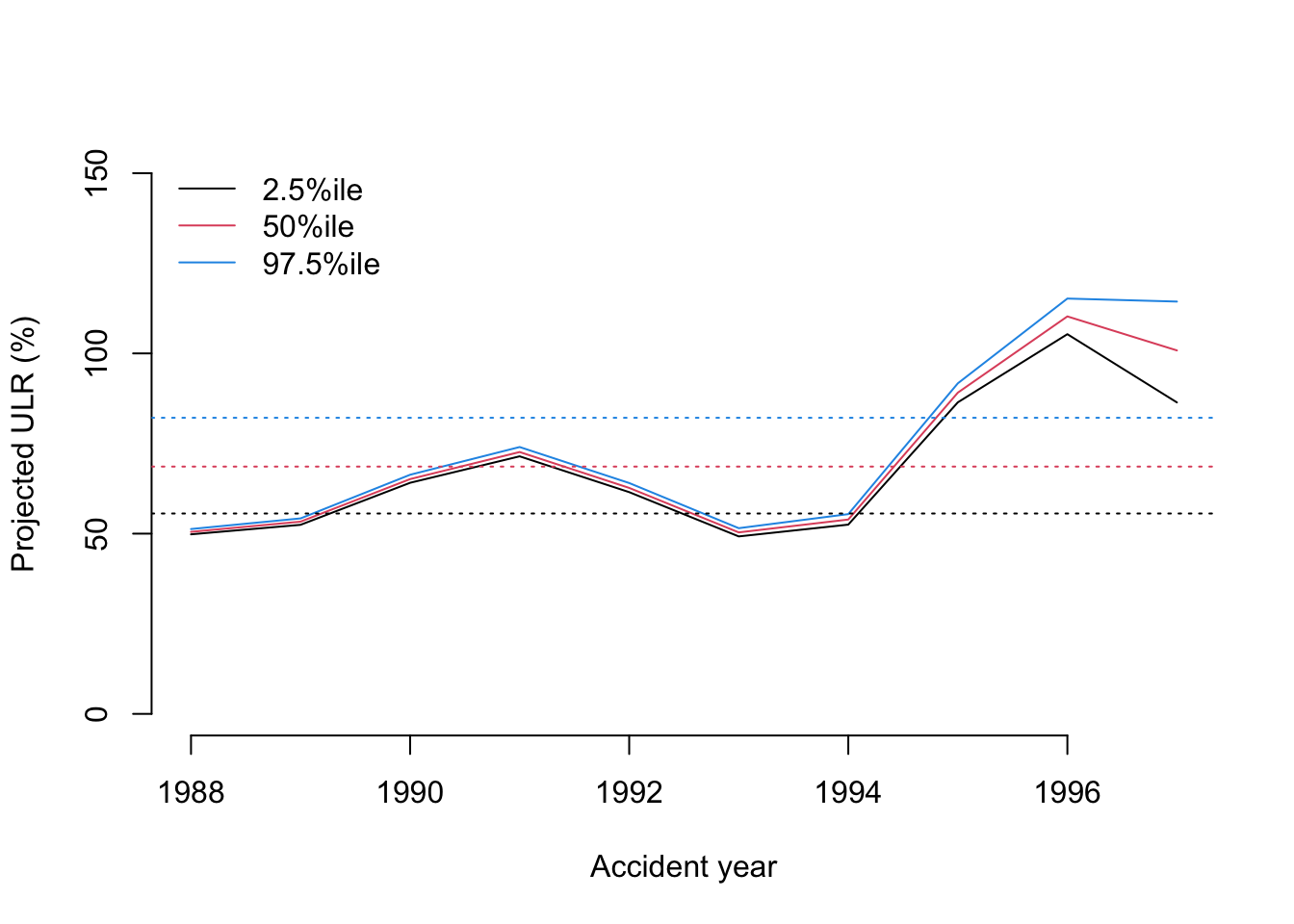
lty=1, col=c(1,2,4), bty="n")

baseULR <- X$b\_RLR \* X$b\_RRF

abline(h=quantile(baseULR, 0.025)\*100, col=1, lty=3)

abline(h=median(baseULR)\*100, col=2, lty=3)

abline(h=quantile(baseULR, 0.975)\*100, col=4, lty=3)



That’s interesting, as the softening casualty market (increasing loss ratio) of the late 1990’s is very visible. From this plot you would assume 1996 was the worst year and perhaps an outlier. However, as the data shows, 1997 was even less profitable than 1996.

Let’s plot the predictions of the model, with the plotDevBananas function.

predClaimsPred <- predict(b1, newdata = lossData0, method="predict")

plotDevBananas(`Q2.5`/premium\*100 + `Q97.5`/premium\*100 +

Estimate/premium\*100 + loss\_train/premium\*100 +

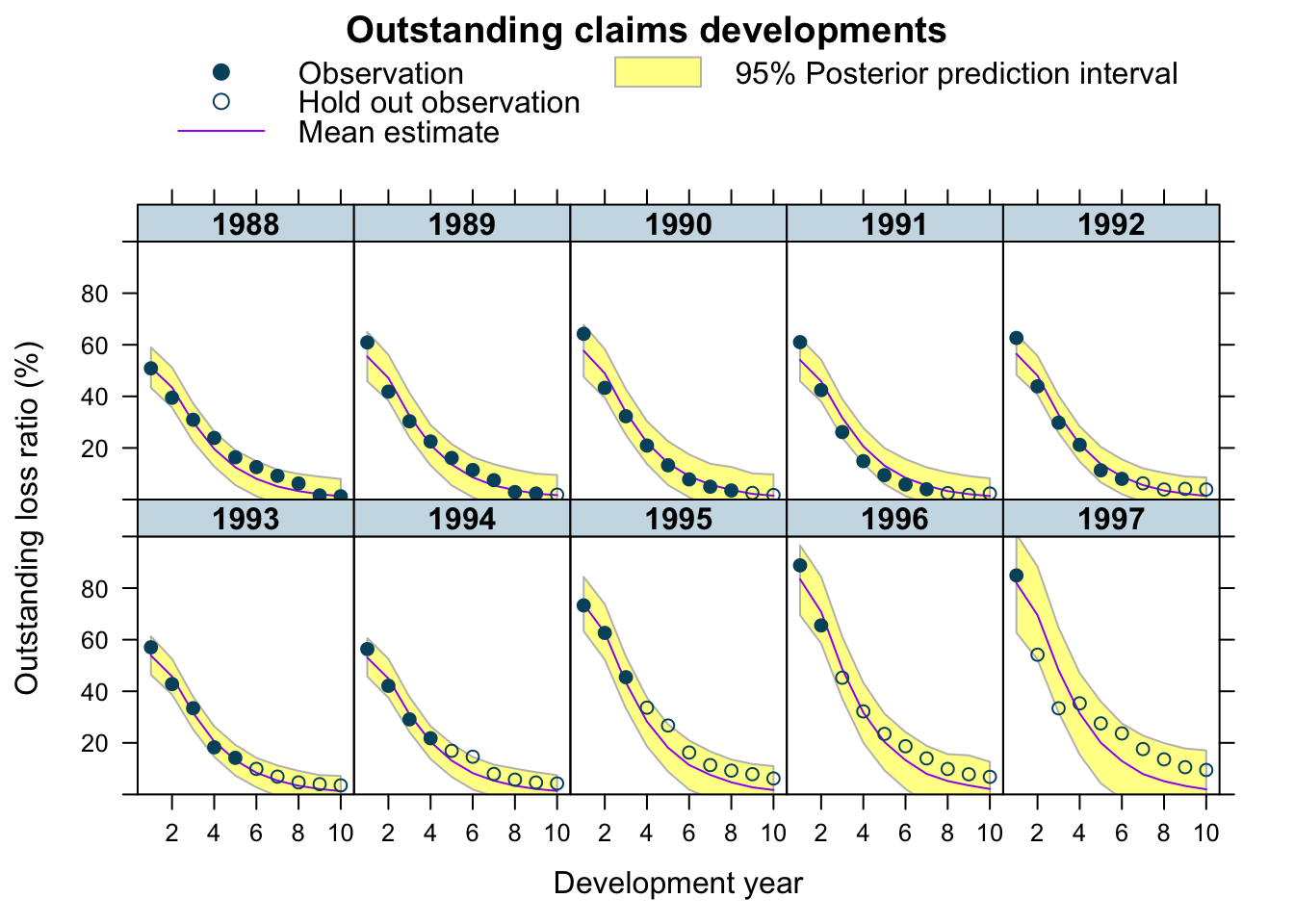
loss\_test/premium\*100 ~

dev | factor(accident\_year), ylim=c(0, 100),

data=cbind(lossData0, predClaimsPred)[delta==0],

main="Outstanding claims developments",

ylab="Outstanding loss ratio (%)")



plotDevBananas(`Q2.5`/premium\*100 + `Q97.5`/premium\*100 +

Estimate/premium\*100 + loss\_train/premium\*100 +

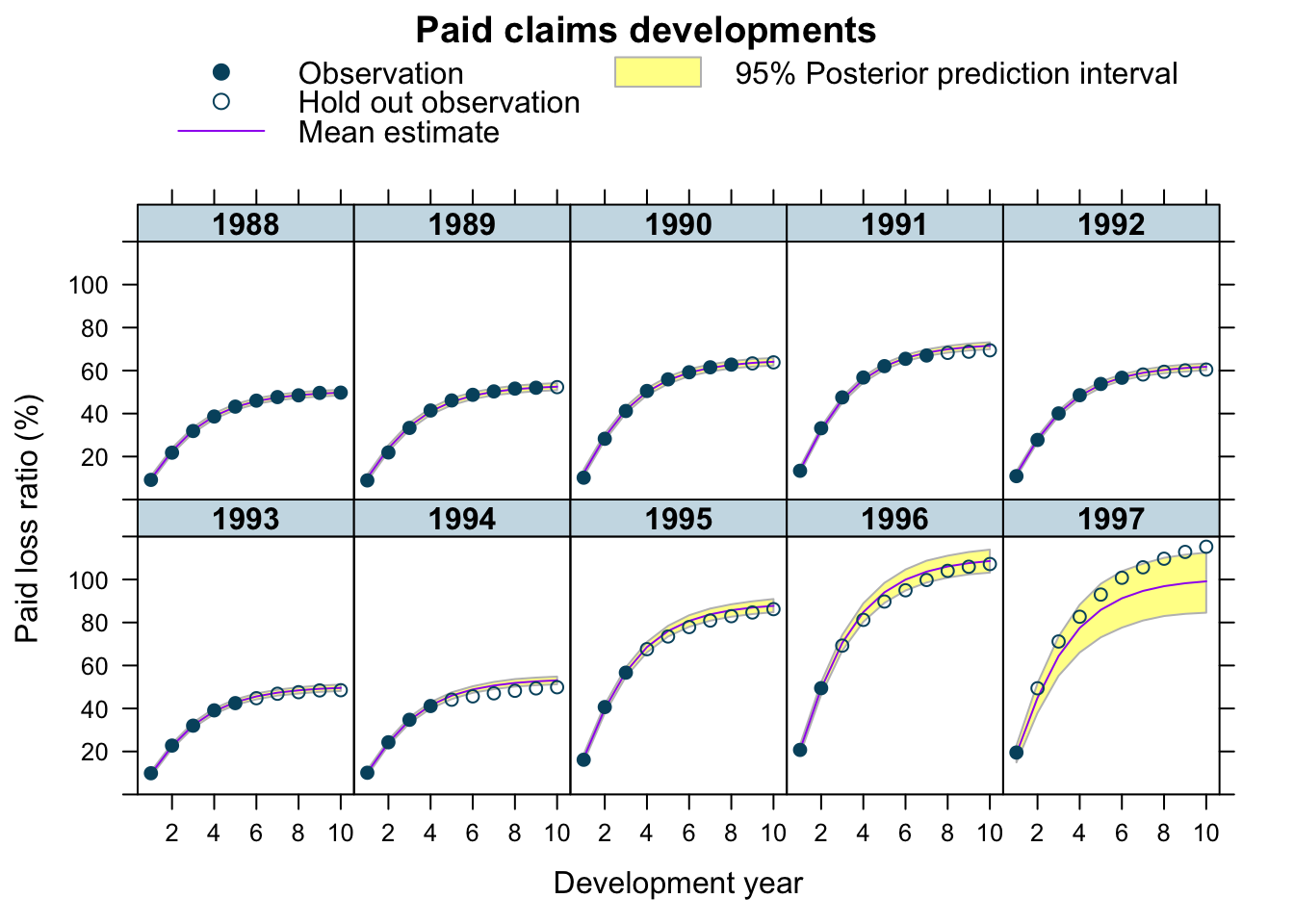
loss\_test/premium\*100 ~

dev | factor(accident\_year), ylim=c(0, 120),

data=cbind(lossData0, predClaimsPred)[delta==1],

main="Paid claims developments",

ylab="Paid loss ratio (%)")



The output looks very promising, apart form the 1997 accident year, which turned out even worse than 1996. Perhaps, I shouldn’t assume a Gaussian data generating distribution, but a distribution which is skewed to the right, such as Gamma, Weibull or Log-normal, all of which is doable with brm by changing the family argument.

The under-prediction for 1997 may arise because historically RRF < 100% i.e. over-reserving. However, over-reserving lessens in more recent accident years as the reserving cycle kicks in, and the latest accident year in particular appears to exhibit under-reserving. With only 2 data points from which to assess this, shrinkage drags the RRF parameter downwards. It would be an interesting exercise to try and incorporate a prior which captures the expectation of a deteriotating reserving cycle.

Additionally, I could test for auto-correlation, calendar year effects, etc. Jake’s paper has further details on this, he also shows how the model can be extended to cover time-dependent parameters and covariate sub-models.

PK/ PD is usually short for pharmacokinetic/ pharmacodynamic models, it could also be short for *Payment Kinetics/ Payment Dynamics Models* in the insurance context.

In this post I’d like to discuss an extension, which allows for a time varying parameter \(k\_{er}(t)\) describing the changing rate of earning and reporting and allows for correlation between \(RLR\), the reported loss ratio, and \(RRF\), the reserve robustness factor. A positive correlation would give evidence of a reserving cycle, i.e. in years with higher initial reported loss ratios a higher proportion of reported losses are paid.

**PK/ PD model**

First of all let’s take a look at my new set of differential equations, describing the dynamics of exposure (\(EX\)), outstanding (\(OS\)) and paid claims (\(PD\)) over time. I replaced the constant parameter \(k\_{er}\) with the linear function \(\beta\_{er} \cdot t\).

\[  
\begin{aligned}  
dEX/dt & = -\beta\_{er} \cdot t \cdot EX \\  
dOS/dt & = \beta\_{er} \cdot t \cdot RLR \cdot EX – k\_p \cdot OS \\  
dPD/dt & = k\_{p} \cdot RRF \cdot OS  
\end{aligned}  
\]

The dynamical system is no longer autonomous and initially I can’t be bothered to solve it analytically. Hence, I use an ODE solver instead, but I will get back to integrating the differential equations later.

**Integrating ODEs with Stan**

Fortunately, an ODE solver is part of the Stan language. The following code demonstrates how I can integrate the differential equations in Stan. You will notice that the model section is empty, as all I want to do for now is run the solver.

functions{

real[] claimsprocess(real t, real [] y, real [] theta,

real [] x\_r, int[] x\_i){

real dydt[3];

dydt[1] = - theta[1] \* t \* y[1];

dydt[2] = theta[1] \* t \* theta[3] \* y[1] - theta[2] \* y[2];

dydt[3] = theta[2] \* theta[4] \* y[2];

return dydt;

}

}

data {

real theta[4];

int N\_t;

real times[N\_t];

real C0[3];

}

parameters{

}

transformed parameters{

real C[N\_t, 3];

C = integrate\_ode\_rk45(claimsprocess, C0, 0, times, theta,

rep\_array(0.0, 0), rep\_array(1, 1));

}

model{

}

generated quantities{

real Exposure[N\_t];

real OS[N\_t];

real Paid[N\_t];

Exposure = C[, 1];

OS = C[, 2];

Paid = C[, 3];

}

To run the Stan code I provide as an input a list with the parameters, the number of data points I’d like to generate, the time line and initial starting values. Note also that I have to set the algorithm = "Fixed\_param" as I run a deterministic model and for that reason one iteration and one chain are sufficient.

library(rstan)

input\_data <- list(

theta=c(Ber=5, kp=0.5, RLR=0.8, RRF=0.9),

N\_t=100,

times=seq(from = 0.1, to = 10, length.out = 100),

C0=c(100, 0, 0))

samples <- sampling(ODEmodel, data=input\_data,

algorithm = "Fixed\_param",

iter = 1, chain = 1)

Let’s take a look at the output.

X <- extract(samples)

out <- data.frame(

Time = input\_data$times,

Exposure = c(X[["Exposure"]]),

Outstandings = c(X[["OS"]]),

Paid = c(X[["Paid"]])

)

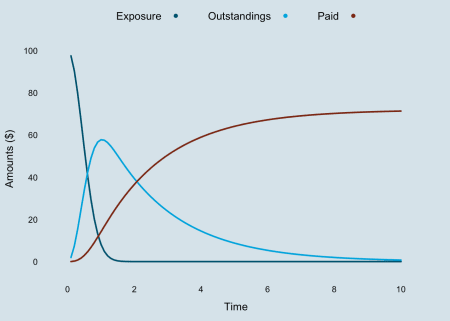
library(latticeExtra)

xyplot(Exposure + Outstandings + Paid ~ Time,

data = out, as.table=TRUE, t="l",

ylab="Amounts ($)", auto.key=list(space="top", columns=3),

par.settings = theEconomist.theme(with.bg = TRUE, box = "transparent"))



The exposure declines faster and the outstanding curve has a more pronounced peak as a result of replacing \(k\_{er}\) with \(\beta\_{er} \cdot t\).

**Updated model**

\[  
\begin{aligned}  
y(t) & \sim \mathcal{N}(\tilde{f}(t, \Pi, \beta\_{er}, k\_p, RLR\_{[i]}, RRF\_{[i]}), \sigma\_{y[\delta]}^2) \\  
\begin{pmatrix} RLR\_{[i]} \\ RRF\_{[i]}\end{pmatrix} & \sim  
\mathcal{N} \left(  
\begin{pmatrix}  
\mu\_{RLR} \\  
\mu\_{RRF}  
\end{pmatrix},  
\begin{pmatrix}  
\sigma\_{RLR}^2 & \rho \ \sigma\_{RLR} \sigma\_{RRF}\\  
\rho \ \sigma\_{RLR} \sigma\_{RRF} & \sigma\_{RRF}^2  
\end{pmatrix}  
\right)  
\end{aligned}  
\]

**Implementation with brms**

Let’s load the data back into R’s memory:

library(data.table)

lossData0 <- fread("https://raw.githubusercontent.com/mages/diesunddas/master/Data/WorkersComp337.csv")

How to implement this model in R with the nlmeODE package, together with more flexible models in OpenBUGS. However, I will continue with brms and Stan.

Using the ODEs with brms requires a little extra coding, as I have to provide the integration function as an additional input, just like I did above, where I defined them as user defined functions for Stan.

To model the group-level terms (\(RLR\), \(RRF\)) of the same grouping factor (accident year) as correlated I add a unique identifier to the | operator, here I use p (looks similar to \(\rho\)), i.e. ~ 1 + (1 | p | accident\_year).

Below is my code for the new updated model. I am using again Gamma distributions as my priors for the parameters .

myFuns <- "

real[] ode\_claimsprocess(real t, real [] y, real [] theta,

real [] x\_r, int[] x\_i){

real dydt[3];

dydt[1] = - theta[1] \* t \* y[1]; // Exposure

dydt[2] = theta[1] \* t \* theta[2] \* y[1] - theta[3] \* y[2]; // OS

dydt[3] = theta[3] \* theta[4] \* y[2]; // Paid

return dydt;

}

real claimsprocess(real t, real premium, real Ber, real kp,

real RLR, real RRF, real delta){

real y0[3];

real y[1, 3];

real theta[4];

theta[1] = Ber;

theta[2] = RLR;

theta[3] = kp;

theta[4] = RRF;

y0[1] = premium;

y0[2] = 0;

y0[3] = 0;

y = integrate\_ode\_rk45(ode\_claimsprocess,

y0, 0, rep\_array(t, 1), theta,

rep\_array(0.0, 0), rep\_array(1, 1),

0.001, 0.001, 100); // tolerances, steps

return (y[1, 2] \* (1 - delta) + y[1, 3] \* delta);

}

"

library(brms)

rstan\_options(auto\_write = TRUE)

options(mc.cores = parallel::detectCores())

b4 <- brm(

bf(loss\_train ~ claimsprocess(dev, premium, Ber, kp, RLR, RRF, delta),

RLR ~ 1 + (1 | p | accident\_year), # 'p' allow for correlation with RRF

RRF ~ 1 + (1 | p | accident\_year), # 'p' allow for correlation with RLR

Ber ~ 1,

kp ~ 1,

sigma ~ 0 + deltaf, # different sigma for OS and paid

nl = TRUE),

stan\_funs = myFuns, # defintion of 'claimsprocess' function

data = lossData0[cal <= max(accident\_year) & dev > 0],

family = brmsfamily("gaussian", link\_sigma = "log"),

prior = c(prior(gamma(4, 5), nlpar = "RLR", lb=0),

prior(gamma(4, 5), nlpar = "RRF", lb=0),

prior(gamma(12, 3), nlpar = "Ber", lb=0),

prior(gamma(3, 4), nlpar = "kp", lb=0),

set\_prior("lkj(2)", class = "cor")),

control = list(adapt\_delta = 0.999, max\_treedepth=15),

seed = 1234, iter = 500)

While the original model with the analytical solution ran in about 3 minutes, this code took about 9 hours per chain for 500 samples. Any suggestions how to speed this up will be much appreciated. The frequentist model using nlmeODE runs in seconds.

b4

## Warning: There were 235 divergent transitions after warmup. Increasing adapt\_delta above 0.999 may help.

## See http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup

## Family: gaussian

## Links: mu = identity; sigma = log

## Formula: loss\_train ~ claimsprocess(dev, premium, Ber, kp, RLR, RRF, delta)

## RLR ~ 1 + (1 | p | accident\_year)

## RRF ~ 1 + (1 | p | accident\_year)

## Ber ~ 1

## kp ~ 1

## sigma ~ 0 + deltaf

## Data: lossData0[cal <= max(accident\_year) & dev > 0] (Number of observations: 110)

## Samples: 4 chains, each with iter = 500; warmup = 250; thin = 1;

## total post-warmup samples = 1000

## ICs: LOO = NA; WAIC = NA; R2 = NA

##

## Group-Level Effects:

## ~accident\_year (Number of levels: 10)

## Estimate Est.Error l-95% CI u-95% CI

## sd(RLR\_Intercept) 0.18 0.06 0.11 0.32

## sd(RRF\_Intercept) 0.15 0.05 0.09 0.27

## cor(RLR\_Intercept,RRF\_Intercept) 0.54 0.24 -0.04 0.88

## Eff.Sample Rhat

## sd(RLR\_Intercept) 197 1.01

## sd(RRF\_Intercept) 397 1.00

## cor(RLR\_Intercept,RRF\_Intercept) 326 1.01

##

## Population-Level Effects:

## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat

## RLR\_Intercept 0.86 0.06 0.74 0.99 161 1.01

## RRF\_Intercept 0.83 0.05 0.73 0.93 179 1.02

## Ber\_Intercept 5.66 0.32 4.97 6.21 359 1.01

## kp\_Intercept 0.40 0.01 0.38 0.41 446 1.00

## sigma\_deltafos -3.61 0.11 -3.81 -3.40 593 1.00

## sigma\_deltafpaid -4.99 0.11 -5.20 -4.79 587 1.00

##

## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample

## is a crude measure of effective sample size, and Rhat is the potential

## scale reduction factor on split chains (at convergence, Rhat = 1).

The output above looks very similar to the output of nlmeODE. That’s good, also that Rhat is close to 1 for all parameters. However, there is a warning message of 235 divergent transitions after warm-up, which I will ignore for the time being.

The correlation coefficient \(\rho\) between \(RLR\_{[i]}\) and \(RRF\_{[i]}\) is estimated as 0.54, but with a wide 95% credible interval from -0.04 to 0.88. Therefore there is moderate evidence of a correlation.

As Jake puts it, a positive correlation between the reported loss ratio and reserve robustness factor parameters by accident year is indicative of a case reserving cycle effect, i.e. more conservative case reserves (low \(RRF\_{[i]}\)) in a hard market (low \(RLR\_{[i]}\)) to create cushions for the future.

stanplot(b4,

pars= c("b\_RLR\_Intercept", "b\_RRF\_Intercept", "b\_Ber\_Intercept",

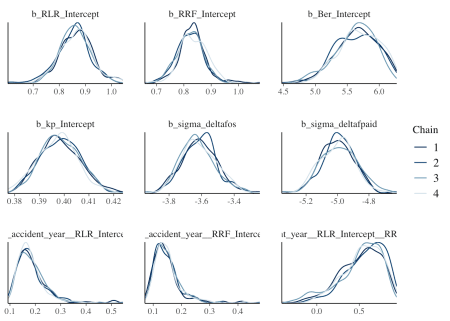
"b\_kp\_Intercept", "b\_sigma\_deltafos", "b\_sigma\_deltafpaid",

"sd\_accident\_year\_\_RLR\_Intercept",

"sd\_accident\_year\_\_RRF\_Intercept",

"cor\_accident\_year\_\_RLR\_Intercept\_\_RRF\_Intercept"),

type="dens\_overlay")



The density plots look OK, all chains seem to have behaved similarly. The last chart in the bottom right shows the distribution of the correlation parameter \(\rho\). Moderate evidence of a correlation might be an understatement. Note also that \(\sigma\_{\delta[OS]}\) and \(\sigma\_{\delta[PD]}\) are on a log-scale.

**Predict future claims developments**

To validate my model against the test data set I can use the predict function in brms. Before I can apply the method I have to expose the Stan functions I wrote above, namely claimsprocess to R.

expose\_functions(b4, vectorize = TRUE) # requires brms >= 2.1.0

predClaimsPred <- predict(b4, newdata = lossData0, method="predict")

plotDevBananas(`2.5%ile`/premium\*100 + `97.5%ile`/premium\*100 +

Estimate/premium\*100 + loss\_train/premium\*100 +

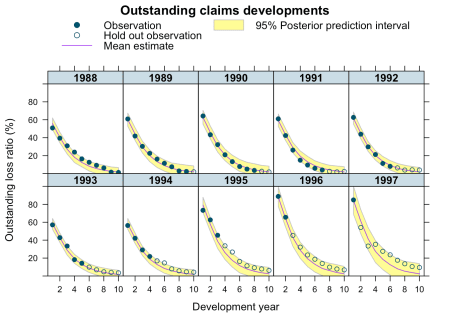
loss\_test/premium\*100 ~

dev | factor(accident\_year), ylim=c(0, 100),

data=cbind(lossData0, predClaimsPred)[delta==0],

main="Outstanding claims developments",

ylab="Outstanding loss ratio (%)")



plotDevBananas(`2.5%ile`/premium\*100 + `97.5%ile`/premium\*100 +

Estimate/premium\*100 + loss\_train/premium\*100 +

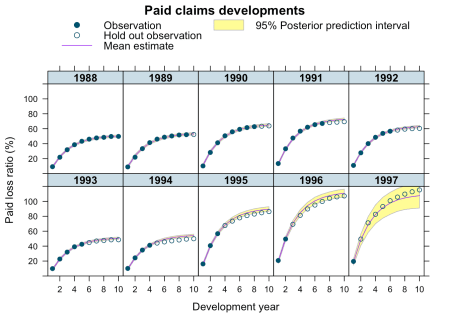
loss\_test/premium\*100 ~

dev | factor(accident\_year), ylim=c(0, 120),

data=cbind(lossData0, predClaimsPred)[delta==1],

main="Paid claims developments",

ylab="Paid loss ratio (%)")



Yet, the prediction for 1997 improved significantly, perhaps because of the correlation between RLR and RRF no longer being assumed to be zero. But this model has tested my patience, as I used a remote 2-core machine waited 18 hours for the results.

**Analytical solution**

The long run time did in the end motivate me to look for an analytical solution of the ODEs, with initial values \(\mbox{EX}(0) = \Pi\) (premium), \(\mbox{OS}(0) = 0\), \(\mbox{PD}(0) = 0\).

\[  
\begin{aligned}  
\mbox{EX}(t) & = \Pi \exp \left(-\frac{\beta\_{er} t^2}{2} \right) \\  
\mbox{OS}(t) & = – \frac{\Pi \cdot RLR}{2 \sqrt{\beta\_{er}}}  
\exp\left(-\frac{\beta\_{er} t^2}{2} – k\_p t \right)  
\left[ \sqrt{2 \pi} k\_p \mbox{erf}\left(\frac{k\_p}{\sqrt{2 \beta\_{er}}}\right)  
\left(-\exp\left(\frac{k\_p^2}{2 \beta\_{er}} + \frac{\beta\_{er} t^2}{2} \right) \right) \right. – \\  
& \left. \qquad \sqrt{2 \pi} k\_p \exp\left(\frac{k\_p^2}{2 \beta\_{er}} + \frac{\beta\_{er} t^2}{2}\right)  
\mbox{erf}\left(\frac{\beta\_{er} t – k\_p}{\sqrt{2 \beta\_{er}}}\right) + 2 \sqrt{\beta\_{er}}  
\exp\left(k\_p t\right) – 2 \sqrt{\beta\_{er}}  
\exp\left(\frac{\beta\_{er} t^2}{2}\right)  
\right] \\  
\mbox{PD}(t) & = \frac{\Pi \cdot RLR \cdot RRF}{2 \sqrt{\beta\_{er}}}  
\exp\left(-k\_p t\right) \left[-\sqrt{2 \pi} k\_p  
\exp\left(\frac{k\_p^2}{2 \beta\_{er}}\right)  
\mbox{erf}\left(\frac{\beta\_{er} t – k\_p}{\sqrt{2 \beta\_{er}}}\right) \right. + \\  
& \left. \qquad \sqrt{2 \pi} k\_p  
\left(-\exp\left(\frac{k^2}{2 \beta\_{er}}\right)\right)  
\mbox{erf}\left(\frac{k}{\sqrt{2 \beta\_{er}}} \right) +  
2 \sqrt{\beta\_{er}} \exp\left(k\_p t\right) –  
2 \sqrt{\beta\_{er}}  
\right]  
\end{aligned}  
\]

Now I can write down \(\tilde{f}(t)\) in a closed-form: \[  
\tilde{f}(t, \delta,\dots) = (1 – \delta) \cdot \mbox{OS}(t, \dots) + \delta \cdot \mbox{PD}(t, \dots)  
\]

The error function is part of the Stan language (not directly in R, although it is just erf <- function(x) 2 \* pnorm(x \* sqrt(2)) - 1). Thus, I create a Stan function for the analytical claims process again and feed this into brm.

library(brms)

library(rstan)

rstan\_options(auto\_write = TRUE)

options(mc.cores = parallel::detectCores())

myFun <- "

real anaclaimsprocess(real t, real premium, real Ber, real kp,

real RLR, real RRF, real delta){

real os;

real paid;

os = -(RLR \* exp(-(Ber \* t^2)/2.0 - kp \* t) \*

(sqrt(2.0 \* pi()) \* kp \* erf(kp/(sqrt(2.0) \* sqrt(Ber))) \*

(-exp(kp^2/(2.0 \* Ber) + (Ber \* t^2)/2.0)) -

sqrt(2.0 \* pi()) \* kp \* exp(kp^2/(2.0 \* Ber) + (Ber \*t^2)/2.0) \*

erf((Ber\*t - kp)/(sqrt(2.0) \* sqrt(Ber))) +

2.0 \* sqrt(Ber) \* exp(kp \* t) - 2.0 \* sqrt(Ber) \*

exp((Ber\*t^2)/2.0)))/(2.0 \* sqrt(Ber));

paid = (RLR \* RRF \* exp(-kp \* t) \*

(-sqrt(2.0 \* pi()) \* kp \* exp(kp^2/(2.0 \* Ber)) \*

erf((Ber\*t - kp)/(sqrt(2.0) \* sqrt(Ber))) +

sqrt(2.0 \* pi()) \* kp \* (-exp(kp^2/(2.0 \* Ber))) \*

erf(kp/(sqrt(2.0) \* sqrt(Ber))) + 2.0 \* sqrt(Ber) \* exp(kp \* t) -

2.0 \* sqrt(Ber)))/(2.0 \* sqrt(Ber));

return (premium \* (os \* (1 - delta) + paid \* delta));

}

"

b5 <- brm(bf(loss\_train ~ anaclaimsprocess(dev, premium, Ber, kp,

RLR, RRF, delta),

RLR ~ 1 + (1 | p | accident\_year),

RRF ~ 1 + (1 | p | accident\_year),

Ber ~ 1,

kp ~ 1,

sigma ~ 0 + deltaf,

nl = TRUE),

stan\_funs = myFun, # defintion of 'anaclaimsprocess' function

data = lossData0[cal <= max(accident\_year)],

family = brmsfamily("gaussian", link\_sigma = "log"),

prior = c(prior(gamma(4, 5), nlpar = "RLR", lb=0),

prior(gamma(4, 5), nlpar = "RRF", lb=0),

prior(gamma(12, 3), nlpar = "Ber", lb=0),

prior(gamma(3, 4), nlpar = "kp", lb=0),

set\_prior("lkj(2)", class = "cor")),

control = list(adapt\_delta = 0.999, max\_treedepth=15),

seed = 1234, iter = 1000)

Wonderful, the code ran in about 5 minutes, with twice the number of samples and without any warnings messages. Let’s take a look at the output:

b5

## Family: gaussian

## Links: mu = identity; sigma = log

## Formula: loss\_train ~ anaclaimsprocess(dev, premium, Ber, kp, RLR, RRF, delta)

## RLR ~ 1 + (1 | p | accident\_year)

## RRF ~ 1 + (1 | p | accident\_year)

## Ber ~ 1

## kp ~ 1

## sigma ~ 0 + deltaf

## Data: lossData0[cal <= max(accident\_year)] (Number of observations: 110)

## Samples: 4 chains, each with iter = 1000; warmup = 500; thin = 1;

## total post-warmup samples = 2000

## ICs: LOO = NA; WAIC = NA; R2 = NA

##

## Group-Level Effects:

## ~accident\_year (Number of levels: 10)

## Estimate Est.Error l-95% CI u-95% CI

## sd(RLR\_Intercept) 0.18 0.06 0.11 0.32

## sd(RRF\_Intercept) 0.15 0.04 0.09 0.26

## cor(RLR\_Intercept,RRF\_Intercept) 0.52 0.26 -0.08 0.89

## Eff.Sample Rhat

## sd(RLR\_Intercept) 852 1.01

## sd(RRF\_Intercept) 860 1.00

## cor(RLR\_Intercept,RRF\_Intercept) 885 1.00

##

## Population-Level Effects:

## Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat

## RLR\_Intercept 0.86 0.06 0.73 0.97 480 1.00

## RRF\_Intercept 0.83 0.05 0.72 0.93 723 1.00

## Ber\_Intercept 5.69 0.35 5.03 6.42 1968 1.00

## kp\_Intercept 0.40 0.01 0.38 0.41 1701 1.00

## sigma\_deltafos -3.61 0.11 -3.80 -3.40 1337 1.00

## sigma\_deltafpaid -4.99 0.11 -5.19 -4.75 1547 1.00

##

## Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample

## is a crude measure of effective sample size, and Rhat is the potential

## scale reduction factor on split chains (at convergence, Rhat = 1).

Perfect, the estimates are very much the same as from the ODE model and the plots haven’t changed much either.

stanplot(b5,

pars= c("b\_RLR\_Intercept", "b\_RRF\_Intercept", "b\_Ber\_Intercept",

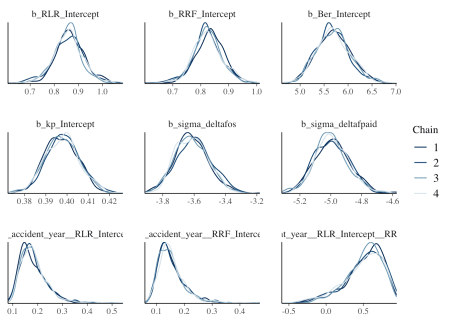
"b\_kp\_Intercept", "b\_sigma\_deltafos", "b\_sigma\_deltafpaid",

"sd\_accident\_year\_\_RLR\_Intercept",

"sd\_accident\_year\_\_RRF\_Intercept",

"cor\_accident\_year\_\_RLR\_Intercept\_\_RRF\_Intercept"),

type="dens\_overlay")



The density plots look smoother and more consistent across chains with twice the number of samples compared to the previous model.

expose\_functions(b5, vectorize = TRUE) # requires brms >= 2.1.0

predClaimsPred2 <- predict(b5, newdata = lossData0, method="predict")

plotDevBananas(`2.5%ile`/premium\*100 + `97.5%ile`/premium\*100 +

Estimate/premium\*100 + loss\_train/premium\*100 +

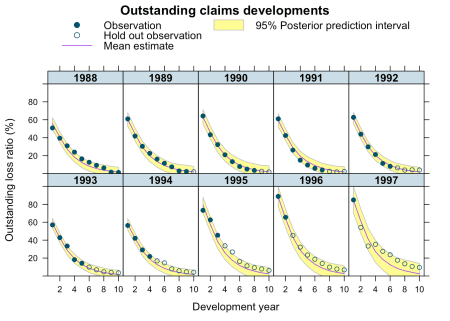
loss\_test/premium\*100 ~

dev | factor(accident\_year), ylim=c(0, 100),

data=cbind(lossData0, predClaimsPred2)[delta==0],

main="Outstanding claims developments",

ylab="Outstanding loss ratio (%)")



plotDevBananas(`2.5%ile`/premium\*100 + `97.5%ile`/premium\*100 +

Estimate/premium\*100 + loss\_train/premium\*100 +

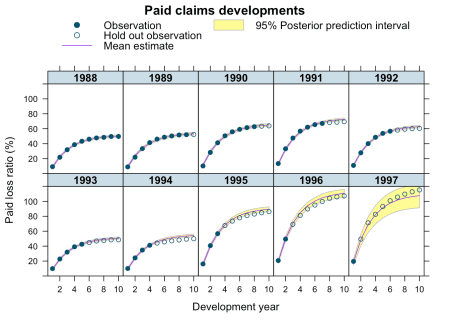
loss\_test/premium\*100 ~

dev | factor(accident\_year), ylim=c(0, 120),

data=cbind(lossData0, predClaimsPred2)[delta==1],

main="Paid claims developments",

ylab="Paid loss ratio (%)")



I am much happier now. The model runs in an acceptable time, allowing me to play around with my assumptions further. I have yet to understand why the integration routine in Stan took so long.

**Session Info**

sessionInfo()

## R version 3.4.3 (2017-11-30)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS High Sierra 10.13.2

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

##

## locale:

## [1] en\_GB.UTF-8/en\_GB.UTF-8/en\_GB.UTF-8/C/en\_GB.UTF-8/en\_GB.UTF-8

##

## attached base packages:

## [1] methods stats graphics grDevices utils datasets base

##

## other attached packages:

## [1] brms\_2.1.0 Rcpp\_0.12.15 data.table\_1.10.4-3

## [4] latticeExtra\_0.6-28 RColorBrewer\_1.1-2 lattice\_0.20-35

## [7] rstan\_2.17.3 StanHeaders\_2.17.2 ggplot2\_2.2.1

##

## loaded via a namespace (and not attached):

## [1] mvtnorm\_1.0-7 gtools\_3.5.0 zoo\_1.8-1

## [4] assertthat\_0.2.0 rprojroot\_1.3-2 digest\_0.6.15

## [7] mime\_0.5 R6\_2.2.2 plyr\_1.8.4

## [10] backports\_1.1.2 stats4\_3.4.3 evaluate\_0.10.1

## [13] coda\_0.19-1 colourpicker\_1.0 blogdown\_0.5

## [16] pillar\_1.1.0 rlang\_0.1.6 lazyeval\_0.2.1

## [19] curl\_3.1 miniUI\_0.1.1 DT\_0.3.3

## [22] Matrix\_1.2-12 rmarkdown\_1.8 labeling\_0.3

## [25] shinythemes\_1.1.1 shinyjs\_1.0 stringr\_1.2.0

## [28] htmlwidgets\_1.0 loo\_1.1.0 igraph\_1.1.2

## [31] munsell\_0.4.3 shiny\_1.0.5 compiler\_3.4.3

## [34] httpuv\_1.3.5 xfun\_0.1 pkgconfig\_2.0.1

## [37] base64enc\_0.1-3 rstantools\_1.4.0 htmltools\_0.3.6

## [40] tibble\_1.4.2 gridExtra\_2.3 bookdown\_0.6

## [43] codetools\_0.2-15 threejs\_0.3.1 matrixStats\_0.53.0

## [46] dplyr\_0.7.4 grid\_3.4.3 nlme\_3.1-131

## [49] xtable\_1.8-2 gtable\_0.2.0 magrittr\_1.5

## [52] scales\_0.5.0 stringi\_1.1.6 reshape2\_1.4.3

## [55] bindrcpp\_0.2 dygraphs\_1.1.1.4 xts\_0.10-1

## [58] tools\_3.4.3 glue\_1.2.0 markdown\_0.8

## [61] shinystan\_2.4.0 crosstalk\_1.0.0 rsconnect\_0.8.5

## [64] abind\_1.4-5 parallel\_3.4.3 yaml\_2.1.16

## [67] inline\_0.3.14 colorspace\_1.3-2 bridgesampling\_0.4-0

## [70] bayesplot\_1.4.0 knitr\_1.18 bindr\_0.1

## [73] Brobdingnag\_1.2-4