Modelling Loss Curves in Insurance with RStan

Mick Cooney

2017-04-17

Model

Loss curves are a standard actuarial technique for helping insurance companies assess the amount of reserve capital they need to keep on hand to cover claims from a line of business. Claims made and reported for a given accounting period are tracked seperately over time. This enables the use of historical patterns of claim development to predict expected total claims for newer policies.

Total claim amounts from a simple accounting period are laid out in a single row of a table, each column showing the total claim amount after that period of time. Subsequent accounting periods have less development, so the data takes a triangular shape - hence the term 'loss triangles'. Using previous patterns, data in the upper part of the triangle is used to predict values in the unknown lower triangle.

The `ChainLadder` package provides functionality to generate and use these loss triangles.

In this case study, we take a related but different approach: we model the growth of the losses in each accounting period as an increasing function of time, and use the model to estimate the parameters which determine the shape and form of this growth. We also use the sampler to estimate the values of the "ultimate loss ratio", i.e. the ratio of the total claims on an accounting period to the total premium received to write those policies. We treat each accounting period as a cohort.

Overview

We will work with two different functional forms for the growth behaviour of the loss curves: a 'Weibull' model and a 'loglogistic' model:

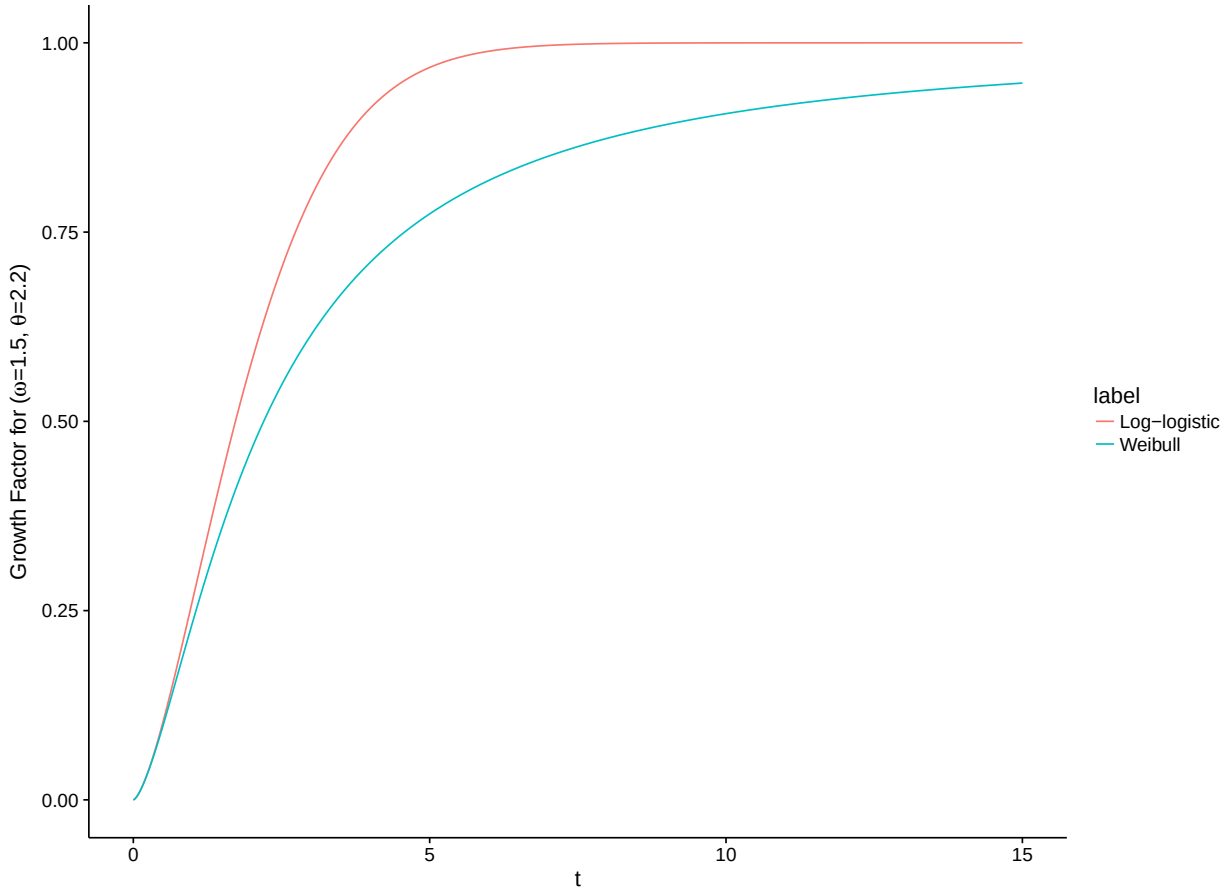
$$g(t\,;\,\theta,\omega) = \frac{t^{\omega}}{t^{\omega} + \theta^{\omega}} \qquad \text{(Weibull)}$$

$$g(t\,;\,\theta,\omega) = 1 - \exp\left(-\left(\frac{t}{\theta}\right)^{\omega}\right) \qquad \text{(Log-logistic)}$$

Weibull vs Log-logistic

We have no prior preference for using either the Weibull or the Log-logistic function to model the growth of the losses.

Visuals are important in this case so we first look at how the two functions differ in value for a given set of parameters.

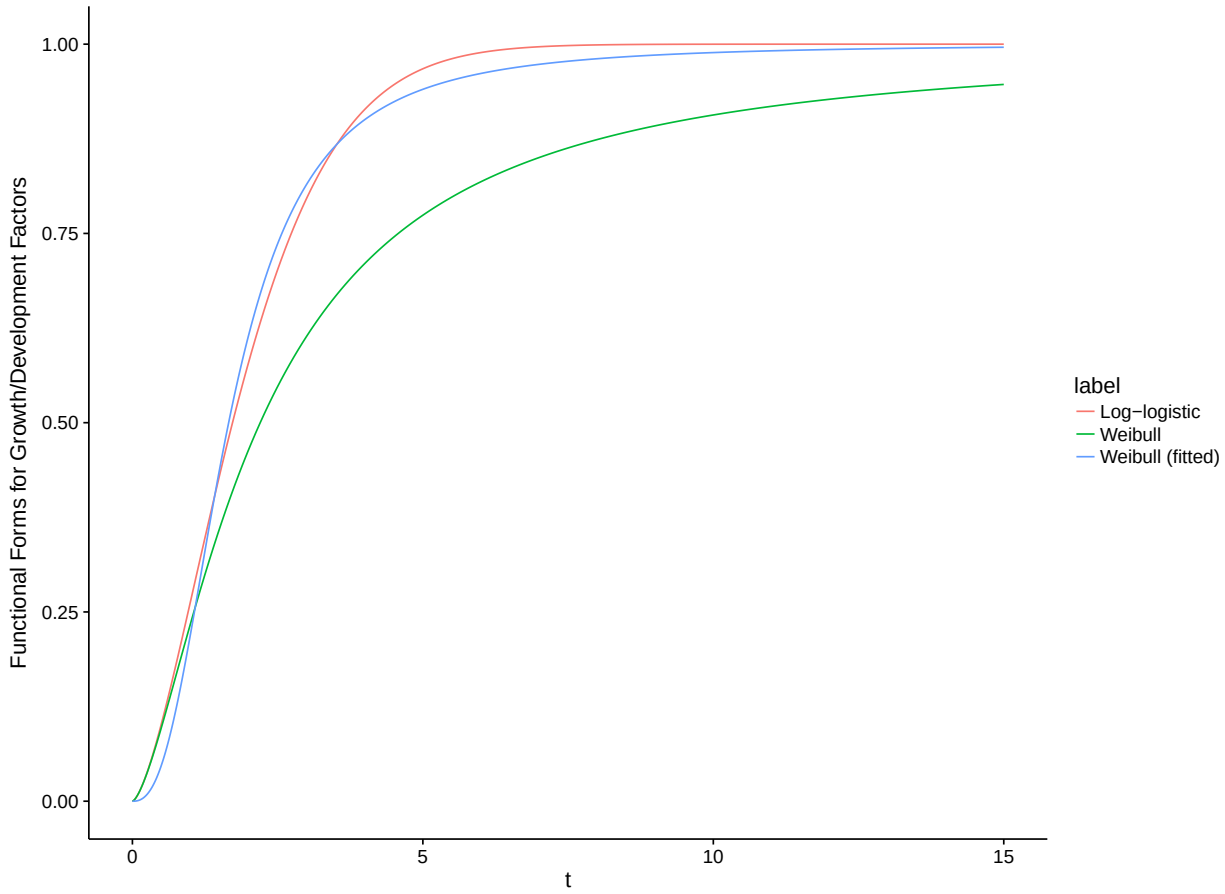


There are references that the Weibull function tends to result in heavier losses in the model, but we have no empirical evidence for such a claim. It is true that for any given set of values for (ω, θ) the log-logistic function plateaus at smaller levels of t, but this seems no guarantee to me: the model could just choose different values for the parameters to counteract this.

To test for this, let us treat the Log-logistic function as the 'true' value and then try to fit a new set of parameters (ω, θ) to see how well a different set of parameters can match another.

```
ll_vals <- loglogistic_tbl$value

new_param_func <- function(x) {
    omega <- x[1]
    theta <- x[2]

    new_vals <- weibull_func(t_seq, omega, theta)

    tot_ss <- sum((new_vals - ll_vals)^2)

    return(tot_ss)
}

optim_params <- optim(c(1, 1), new_param_func)


fittedweibull_tbl <- tibble(
    label = 'Weibull (fitted)'
```

```
   ,t = t_seq
   ,value = weibull_func(t_seq
                         ,optim_params$par[1]
                         ,optim_params$par[2])
)


plot_tbl <- bind_rows(weibull_tbl
                      ,loglogistic_tbl
                      ,fittedweibull_tbl)

ggplot(plot_tbl) +
    geom_line(aes(x = t, y = value, colour = label)) +
    xlab(expression(t)) +
    ylab(expression("Functional Forms for Growth/Development Factors"))
```



We see that the Weibull growth function comes quite close to a log-logistic function so the choice should be a matter of preference in the main.

That said, we shall try both and see if we see much difference between the two.

Load Data

We load the Schedule P loss data from casact.org.

```
### File was downloaded from http://www.casact.org/research/reserve_data/ppauto_pos.csv
data_files <- dir("data/", pattern = "\\.csv", full.names = TRUE)

rawdata_tbl <- data_files %>%
    map(read_claim_datafile) %>%
    bind_rows

glimpse(rawdata_tbl)
```

```
## Observations: 77,900
## Variables: 14
## $ grcode          <int> 266, 266, 266, 266, 266, 266, 266, 266, 266, 266, 2...
## $ grname          <chr> "Public Underwriters Grp", "Public Underwriters Grp...
## $ accidentyear    <int> 1988, 1988, 1988, 1988, 1988, 1988, 1988, 1988, 198...
## $ developmentyear <int> 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 199...
## $ developmentlag  <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7,...
## $ incurloss       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22, 24, 21, 24, 25, 2...
## $ cumpaidloss     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 20, 21, 23, 24, 24...
## $ bulkloss        <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 1, 0, 0, 0, 0, 0, ...
## $ earnedpremdir   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 25, 25, 25, 25, 2...
## $ earnedpremceded <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ earnedpremnet   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 25, 25, 25, 25, 2...
## $ single          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ postedreserve97 <int> 932, 932, 932, 932, 932, 932, 932, 932, 932, 932, 9...
## $ lob             <chr> "comauto", "comauto", "comauto", "comauto", "comaut...
```

```
claimdata_tbl <- rawdata_tbl %>%
    mutate(acc_year   = as.character(accidentyear)
          ,dev_year   = developmentyear
          ,dev_lag    = developmentlag
          ,premium    = earnedpremdir
          ,cum_loss   = cumpaidloss
          ,loss_ratio = cum_loss / premium) %>%
    select(grcode, grname, lob, acc_year, dev_year, dev_lag, premium, cum_loss, loss_ratio)
```

With the data in the format we will use in this analysis, we take a look at it in tabular form:

```
print(claimdata_tbl)
```

```
## # A tibble: 77,900 × 9
##    grcode                  grname     lob acc_year dev_year dev_lag premium
##     <int>                   <chr>   <chr>    <chr>    <int>   <int>   <int>
## 1     266 Public Underwriters Grp comauto     1988     1988       1       0
## 2     266 Public Underwriters Grp comauto     1988     1989       2       0
## 3     266 Public Underwriters Grp comauto     1988     1990       3       0
## 4     266 Public Underwriters Grp comauto     1988     1991       4       0
## 5     266 Public Underwriters Grp comauto     1988     1992       5       0
## 6     266 Public Underwriters Grp comauto     1988     1993       6       0
## 7     266 Public Underwriters Grp comauto     1988     1994       7       0
## 8     266 Public Underwriters Grp comauto     1988     1995       8       0
## 9     266 Public Underwriters Grp comauto     1988     1996       9       0
## 10    266 Public Underwriters Grp comauto     1988     1997      10       0
## # ... with 77,890 more rows, and 2 more variables: cum_loss <int>,
## #   loss_ratio <dbl>
```

Data Exploration

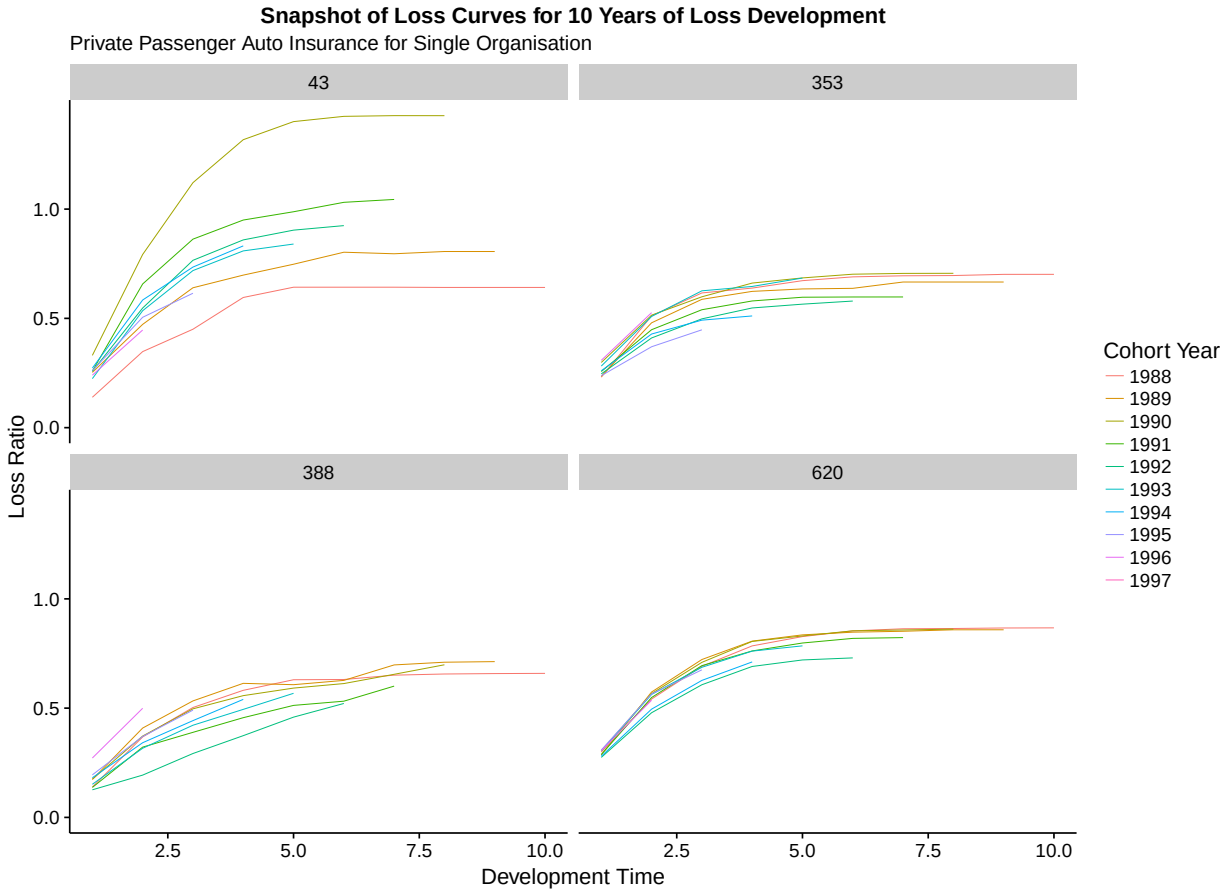
In terms of modeling, we first confine ourselves to a single line of business 'ppauto' and ensure the data we work with is a snapshot in time. We remove all data timestamped after 1997 and use the remaining data as our modelling dataset.

Once we have fits and predictions, we use the later timestamped data as a way to validate the model.

```
use_grcode <- c(43,353,388,620)

carrier_full_tbl <- claimdata_tbl %>%
    filter(lob == 'ppauto')

carrier_snapshot_tbl <- carrier_full_tbl %>%
    filter(grcode %in% use_grcode
          ,dev_year < 1998)
```

We are looking at four insurers with the GRCODEs above. Before we proceed with any analysis, we first plot the data, grouping the loss curves by accounting year and faceting by carrier.

```
ggplot(carrier_snapshot_tbl) +
    geom_line(aes(x = dev_lag, y = loss_ratio, colour = as.character(acc_year))
             ,size = 0.3) +
    expand_limits(y = c(0,1)) +
    facet_wrap(~grcode) +
    xlab('Development Time') +
    ylab('Loss Ratio') +
    ggtitle('Snapshot of Loss Curves for 10 Years of Loss Development'
           ,subtitle = 'Private Passenger Auto Insurance for Single Organisation') +
    guides(colour = guide_legend(title = 'Cohort Year'))
```



We look at the chain ladder of the data, rather than looking at the loss ratios we just look at the dollar amounts of the losses.

```
snapshot_tbl <- carrier_snapshot_tbl %>%
    filter(grcode %in% use_grcode[1])

snapshot_tbl %>%
    select(acc_year, dev_lag, premium, cum_loss) %>%
    spread(dev_lag, cum_loss) %>%
    print
```

```
## # A tibble: 10 × 12
##    acc_year premium   `1`   `2`   `3`   `4`   `5`   `6`   `7`   `8`   `9`  `10`
## *     <chr>   <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
## 1      1988     957   133   333   431   570   615   615   615   614   614   614
## 2      1989    3695   934  1746  2365  2579  2763  2966  2940  2978  2978    NA
## 3      1990    6138  2030  4864  6880  8087  8595  8743  8763  8762    NA    NA
## 4      1991   17533  4537 11527 15123 16656 17321 18076 18308    NA    NA    NA
## 5      1992   29341  7564 16061 22465 25204 26517 27124    NA    NA    NA    NA
## 6      1993   37194  8343 19900 26732 30079 31249    NA    NA    NA    NA    NA
## 7      1994   46095 12565 26922 33867 38338    NA    NA    NA    NA    NA    NA
## 8      1995   51512 13437 26012 31677    NA    NA    NA    NA    NA    NA    NA
## 9      1996   52481 12604 23446    NA    NA    NA    NA    NA    NA    NA    NA
## 10     1997   56978 12292    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

We also look at the loss ratios in a similar fashion

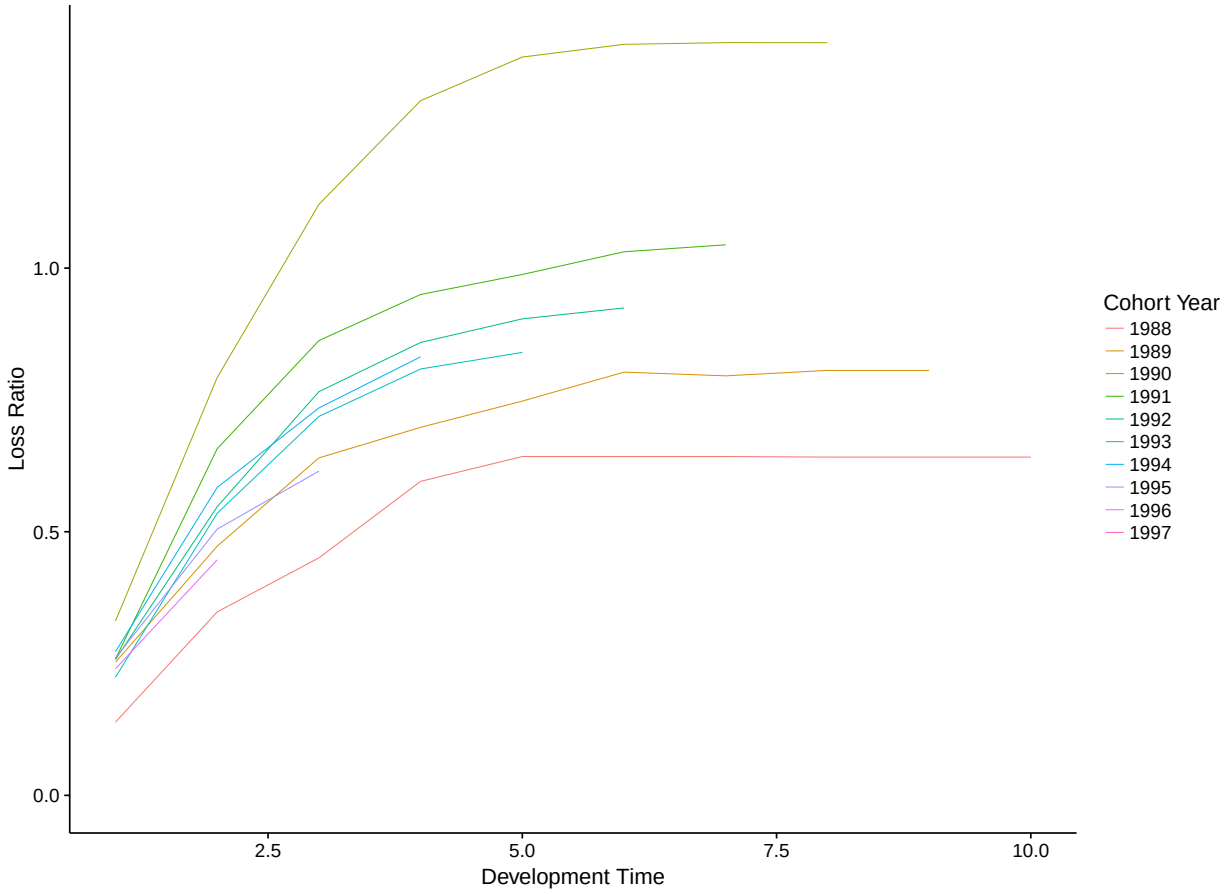
```
snapshot_tbl %>%
    select(acc_year, dev_lag, premium, loss_ratio) %>%
    spread(dev_lag, loss_ratio) %>%
    print.data.frame(digits = 2)
```

```
##    acc_year premium    1    2    3    4    5    6    7    8    9   10
## 1      1988     957 0.14 0.35 0.45 0.60 0.64 0.64 0.64 0.64 0.64 0.64
## 2      1989    3695 0.25 0.47 0.64 0.70 0.75 0.80 0.80 0.81 0.81   NA
## 3      1990    6138 0.33 0.79 1.12 1.32 1.40 1.42 1.43 1.43   NA   NA
## 4      1991   17533 0.26 0.66 0.86 0.95 0.99 1.03 1.04   NA   NA   NA
## 5      1992   29341 0.26 0.55 0.77 0.86 0.90 0.92   NA   NA   NA   NA
## 6      1993   37194 0.22 0.54 0.72 0.81 0.84   NA   NA   NA   NA   NA
## 7      1994   46095 0.27 0.58 0.73 0.83   NA   NA   NA   NA   NA   NA
## 8      1995   51512 0.26 0.50 0.61   NA   NA   NA   NA   NA   NA   NA
## 9      1996   52481 0.24 0.45   NA   NA   NA   NA   NA   NA   NA   NA
## 10     1997   56978 0.22   NA   NA   NA   NA   NA   NA   NA   NA   NA
```

Loss Ratio Ladders

We are working with the loss ratio, so we recreate the chain ladder format but look at loss ratios instead of dollar losses.

```
ggplot(snapshot_tbl) +
    geom_line(aes(x = dev_lag, y = loss_ratio, colour = acc_year)
              ,size = 0.3) +
    expand_limits(y = 0) +
    xlab('Development Time') +
    ylab('Loss Ratio') +
    guides(colour = guide_legend(title = 'Cohort Year'))
```



Initial Model - Single Line-of-Business, Single Insurer (SISLOB)

For our first model, we wish to keep things simple and so restrict the model to considering a single line-of-buiness for a single insurer. Thus, our data is in the form of a single triangle, and the problem confines itself to modelling a single triangle, giving us a simple starting place for improving and extending the model.

The basic concept is to model the growth of losses in a cohort of policies as a function of time. As mentioned, we use two different growth functions, the Weibull and the Log-Logistic. Both functions have two parameters which we label θ and ω.

As each cohort year has different volumes of business, we scale the losses by the total premium received for that cohort, allowing us to more directly compare the cohorts. The total losses is then given by

$$\text{Total Loss}(t) = \text{Premium} \times \text{Final Loss Ratio} \times \text{GF}(t)$$

Each accounting year, Y, in the cohort gets its own value for the Final Loss Ratio, LR_Y, each cumulated loss value in the data can be modelled as

$$\text{Loss}(Y, t) \sim \text{Normal}(\mu(Y, t),\, \sigma_Y)$$

where we have

$$
\begin{aligned}
\mu(Y,t) &= \text{Premium}(Y) \times \text{LR}(Y) \times \text{GF}(t) \\
\text{GF}(t) &= \text{growth function of } t \\
\sigma_Y &= \text{Premium}(Y) \times \sigma \\
\text{LR}_Y &\sim \text{Lognormal}(\mu_{\text{LR}}, \sigma_{\text{LR}}) \\
\mu_{\text{LR}} &\sim \text{Normal}(0, 0.5)
\end{aligned}
$$

All other parameters in the model, (σ_{LR}, ω, θ, σ), have lognormal priors to constrain them to be positive.

The priors on the hyper-parameters are weakly infomative - chosen to cover the feasiable range of parameter values with a lot of uncertainty.

By setting the model up in this way, Stan can fit for both the shape of the growth curve - as this is determined by θ and ω - and the loss ratios for each cohort simultaneously.

Configure Data

We want to only use the data at a given snapshot, so we choose all data current to 1998. Thus, we have 10 years of development for our first 1988 cohort, and one less for each subsequent year. Our final cohort for 1997 has only a single year of development

```
modeldata_tbl <- claimdata_tbl %>%
    filter(lob == 'ppauto'
          ,grcode == use_grcode[1])

usedata_tbl <- modeldata_tbl %>%
    filter(dev_year < 1998)

cohort_maxtime <- usedata_tbl %>%
    group_by(acc_year) %>%
    summarise(maxtime = max(dev_lag)) %>%
    arrange(acc_year) %>%
    .[['maxtime']]

cohort_premium <- usedata_tbl %>%
    group_by(acc_year) %>%
    summarise(premium = unique(premium)) %>%
    .[['premium']]

t_values <- usedata_tbl %>%
    select(dev_lag) %>%
    arrange(dev_lag) %>%
    unique %>%
    .[['dev_lag']]

standata_lst <- list(
    growthmodel_id = 1   # Use weibull rather than loglogistic
   ,n_data         = usedata_tbl %>% nrow
   ,n_time         = usedata_tbl %>% select(dev_lag)  %>% unique %>% nrow
   ,n_cohort       = usedata_tbl %>% select(acc_year) %>% unique %>% nrow
   ,cohort_id      = get_character_index(usedata_tbl$acc_year)
   ,cohort_maxtime = cohort_maxtime
   ,t_value        = t_values
```

```
   ,t_idx          = get_character_index(usedata_tbl$dev_lag)
   ,premium        = cohort_premium
   ,loss           = usedata_tbl$cum_loss
)
```

The full Stan file is shown below:

```
stan_file <- "losscurves_sislob.stan"

cat(read_lines(stan_file), sep = "\n")
```

```
functions {
    real growth_factor_weibull(real t, real omega, real theta) {
        return 1 - exp(-(t/theta)^omega);
    }

    real growth_factor_loglogistic(real t, real omega, real theta) {
        real pow_t_omega = t^omega;
        return pow_t_omega / (pow_t_omega + theta^omega);
    }
}

data {
    int<lower=0,upper=1> growthmodel_id;

    int n_data;
    int n_time;
    int n_cohort;

    int cohort_id[n_data];
    int t_idx[n_data];

    int cohort_maxtime[n_cohort];

    real<lower=0> t_value[n_time];

    real premium[n_cohort];
    real loss[n_data];
}

parameters {
    real<lower=0> omega;
    real<lower=0> theta;

    real<lower=0> LR[n_cohort];

    real mu_LR;
    real<lower=0> sd_LR;

    real<lower=0> loss_sd;
}

transformed parameters {
    real gf[n_time];
    real loss_mean[n_cohort, n_time];
```

```
    for(i in 1:n_time) {
        gf[i] = growthmodel_id == 1 ?
            growth_factor_weibull    (t_value[i], omega, theta) :
            growth_factor_loglogistic(t_value[i], omega, theta);
    }

    for(i in 1:n_cohort) {
        for(j in 1:n_time) {
            loss_mean[i,j] = 0;
        }
    }

    for(i in 1:n_data) {
        loss_mean[cohort_id[i], t_idx[i]] = LR[cohort_id[i]] * premium[cohort_id[i]] * gf[t_idx[i]];
    }
}

model {
    mu_LR ~ normal(0, 0.5);
    sd_LR ~ lognormal(0, 0.5);

    LR ~ lognormal(mu_LR, sd_LR);

    loss_sd ~ lognormal(0, 0.7);

    omega ~ lognormal(0, 0.5);
    theta ~ lognormal(0, 0.5);

    for(i in 1:n_data) {
        loss[i] ~ normal(loss_mean[cohort_id[i], t_idx[i]], premium[cohort_id[i]] * loss_sd);
    }
}


generated quantities {
    real mu_LR_exp;
    real<lower=0> loss_sample[n_cohort, n_time];
    real<lower=0> loss_prediction[n_cohort, n_time];
    real<lower=0> step_ratio[n_cohort, n_time];

    real<lower=0> ppc_minLR;
    real<lower=0> ppc_maxLR;

    real<lower=0> ppc_EFC;

    for(i in 1:n_cohort) {
        for(j in 1:n_time) {
            loss_sample[i, j] = LR[i] * premium[i] * gf[t_idx[j]];
            step_ratio [i, j] = 1.0;
        }
    }

    mu_LR_exp = exp(mu_LR);
```

```
    for(i in 1:n_data) {
        loss_prediction[cohort_id[i], t_idx[i]] = loss[i];
    }

    for(i in 1:n_cohort) {
        for(j in 2:n_time) {
            step_ratio[i, j] = gf[t_idx[j]] / gf[t_idx[j-1]];
        }
    }

    for(i in 1:n_cohort) {
        for(j in (cohort_maxtime[i]+1):n_time) {
            loss_prediction[i,j] = loss_prediction[i,j-1] * step_ratio[i,j];
        }
    }

    // Create PPC distributions for the max/min of LR
    ppc_minLR = min(LR);
    ppc_maxLR = max(LR);

    // Create total reserve PPC
    ppc_EFC = 0;

    for(i in 1:n_cohort) {
        ppc_EFC = ppc_EFC + loss_prediction[i, n_time] - loss_prediction[i, cohort_maxtime[i]];
    }
}
```

There are a few points of note about this Stan model worth highlighting here.

Local functions

To avoid having near-duplicate versions of the Stan model, we define local functions to calculate both the Weibull and Log-logistic functions. In general, Weibull models are said to produce fits with heavier losses, as fact we shall test.

Loss Ratios and Ensuring Positivity

To ensure positivity on the loss ratios, we use the lognormal distribution for the `LR` variables. We also use them for the standard deviations, but we may alter this approach and try half-Cauchy distributions as an alternative.

Prior for `mu_LR`

Because we use lognormals for the underlying losses, we want the prior for the mean of the distribution to take both positive and negative values, and thus use a normal distribution for μ_{LR}, with mean 0 and std dev 0.5.

The `generated quantities` block

We use the `generated quantities` block to facilitate both posterior predictive checks and to also make loss reserving projections across all the cohorts. The use and analysis of the output of this block is discussed in a later section.

Fitting the Stan Model

We now proceed with fitting the stan model and examing the output.

```
model_sislob_stanmodel <- stan_model(stan_file)
```

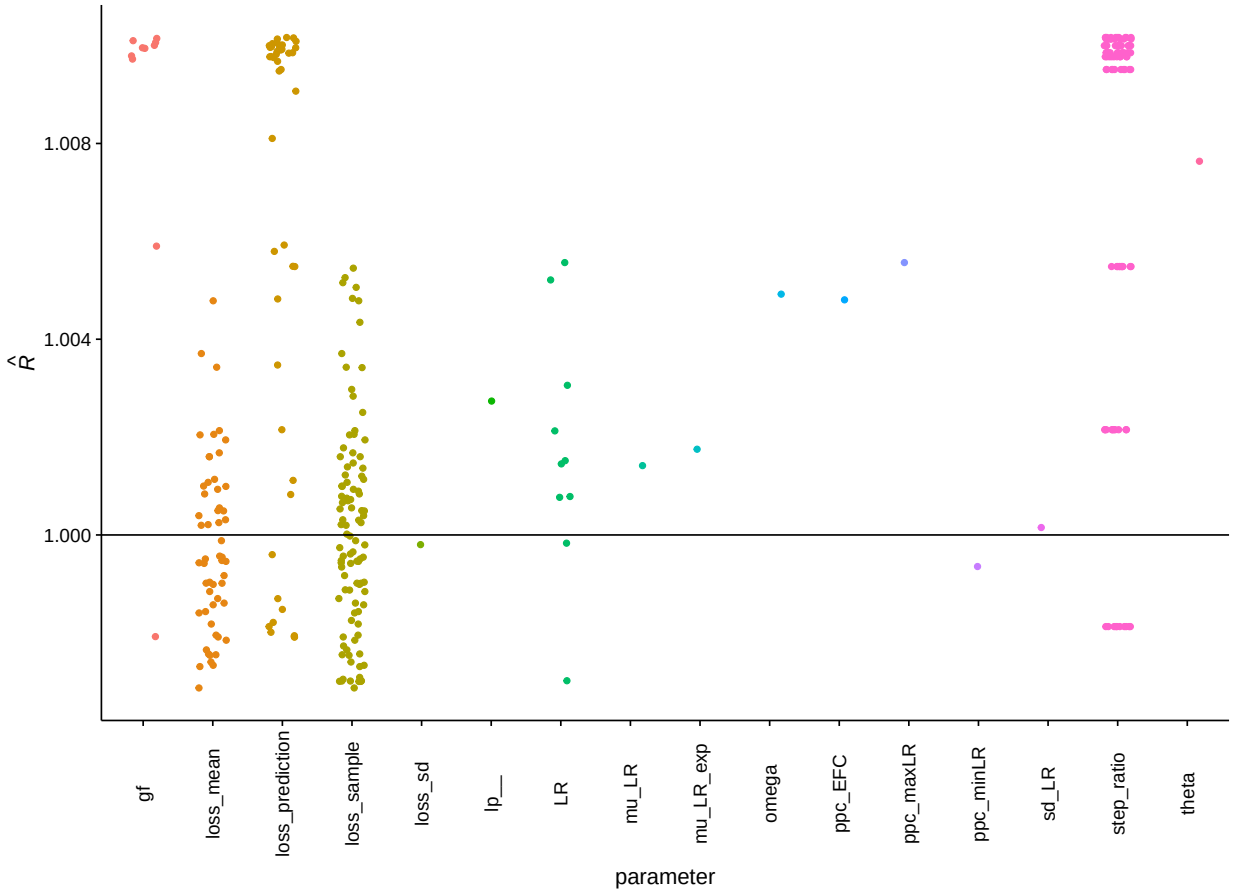
```
model_sislob_stanfit <- sampling(
    object = model_sislob_stanmodel
   ,data   = standata_lst
   ,iter   = 500
   ,chains = 8
   ,seed   = stan_seed
)
```

The Stan sample contains no divergent transitions, a good start.

Sampler Diagnostic Plots

It is always worth checking convergence of the model by checking the $\hat{R}$ and ensuring it is less than about 1.1

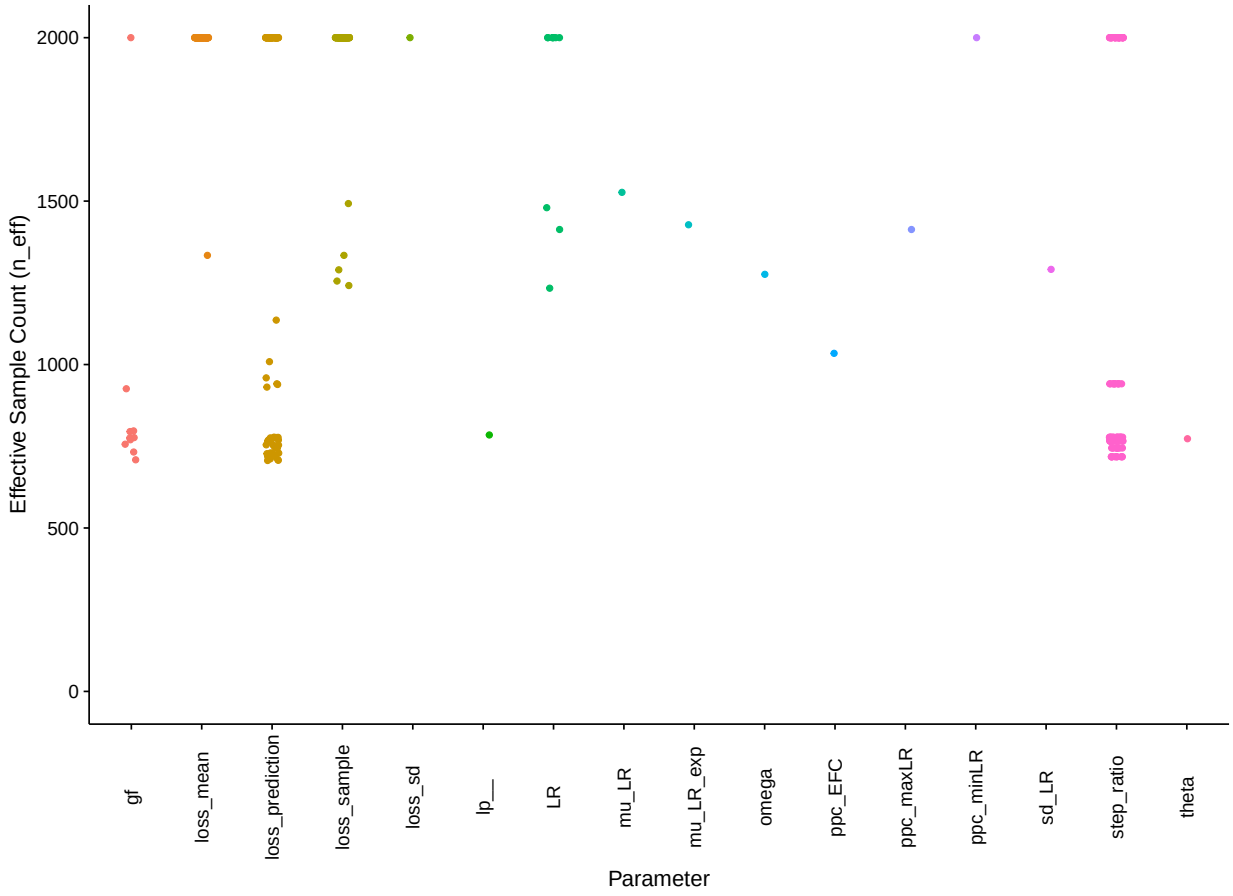
```
# Plot of convergence statistics
model_sislob_draws       <- extract(model_sislob_stanfit, permuted = FALSE, inc_warmup = TRUE)
model_sislob_monitor_tbl <- as.data.frame(monitor(model_sislob_draws, print = FALSE))
model_sislob_monitor_tbl <- model_sislob_monitor_tbl %>%
    mutate(variable  = rownames(model_sislob_monitor_tbl)
          ,parameter = gsub("\\[.*]", "", variable)
           )

ggplot(model_sislob_monitor_tbl) +
    aes(x = parameter, y = Rhat, color = parameter) +
    geom_jitter(height = 0, width = 0.2, show.legend = FALSE) +
    geom_hline(aes(yintercept = 1), size = 0.5) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
    ylab(expression(hat(italic(R))))
```



The $\hat{R}$ values for the parameter appear to be in or around 1. A positive sign, but not sufficient for convergence.

We check the size of `n_eff` for each of the variables:

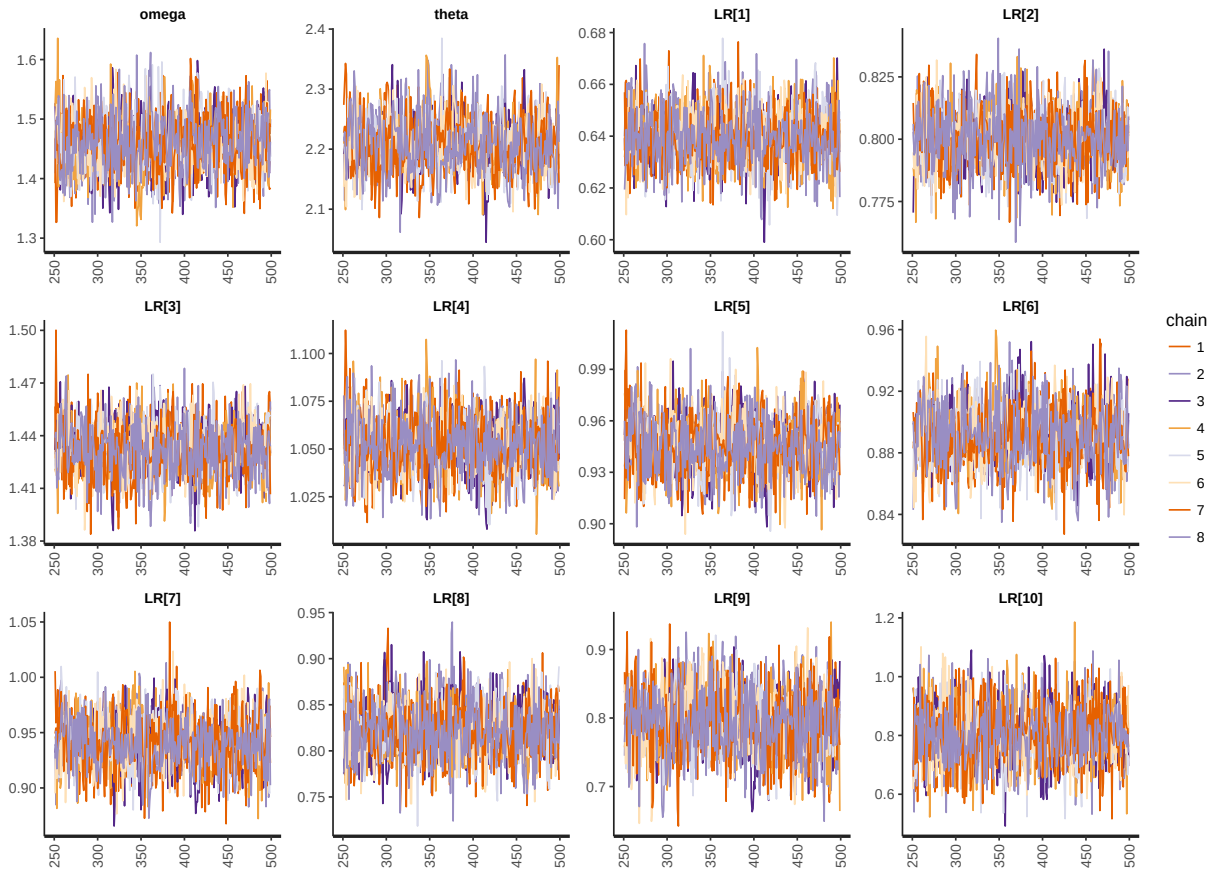
```
ggplot(model_sislob_monitor_tbl) +
    aes(x = parameter, y = n_eff, color = parameter) +
    geom_jitter(height = 0, width = 0.1, show.legend = FALSE) +
    expand_limits(y = 0) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
    xlab("Parameter") +
    ylab(paste0("Effective Sample Count (n_eff)"))
```



The lowest sample size for the parameters is around 800, about 40% of the maximum. This is reasonable, though the higher is better.

Traceplots for the parameters are a useful diagnostic. The large parameter count makes the plots messy, so we break them up into groups. First we look at `omega`, `theta` and `LR`.

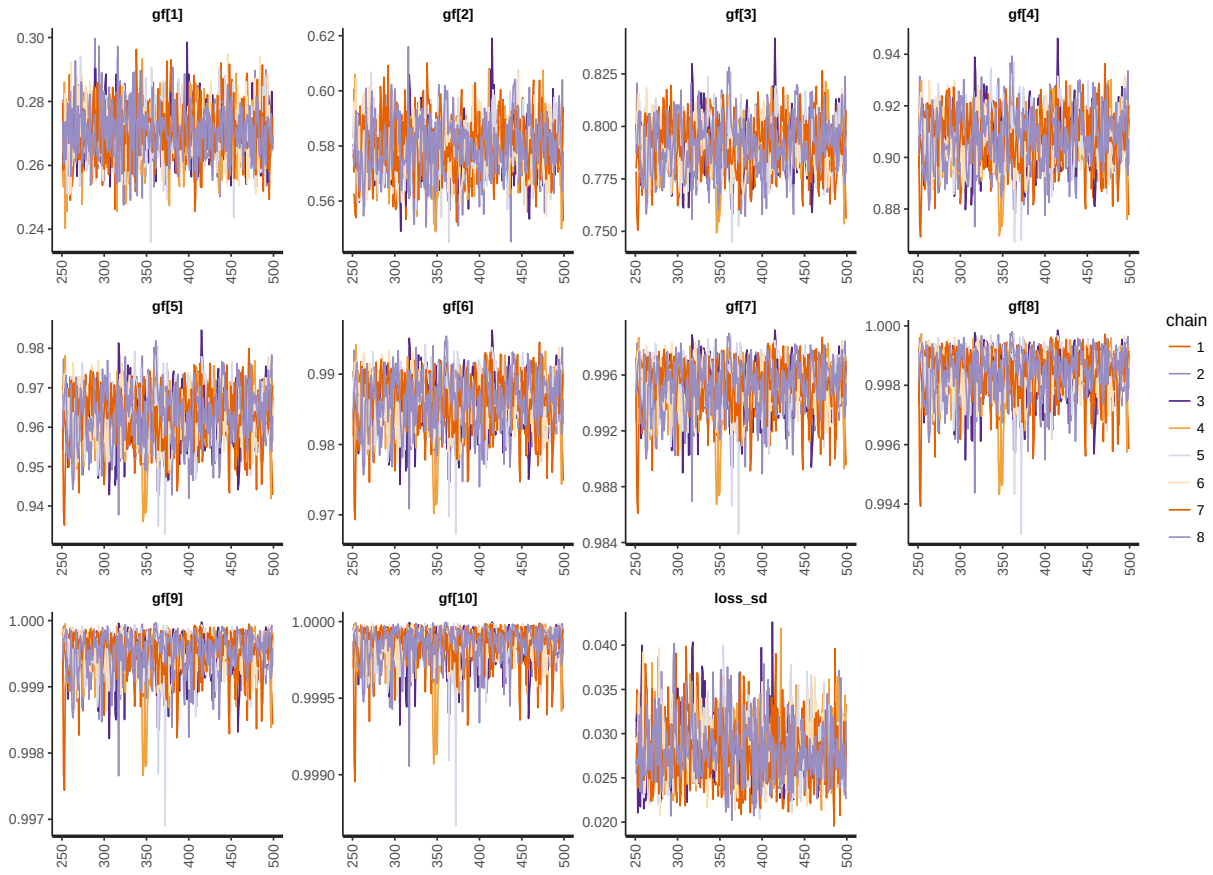
```
traceplot(model_sislob_stanfit, pars = c("omega", "theta", "LR")) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



We have 8 chains, and the plots show signs the chains have mixed well, with no indications of difficult exploration of the posterior.

Now we look at the traces for `gf` and `loss_sd`.

```
traceplot(model_sislob_stanfit, pars = c("gf", "loss_sd")) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



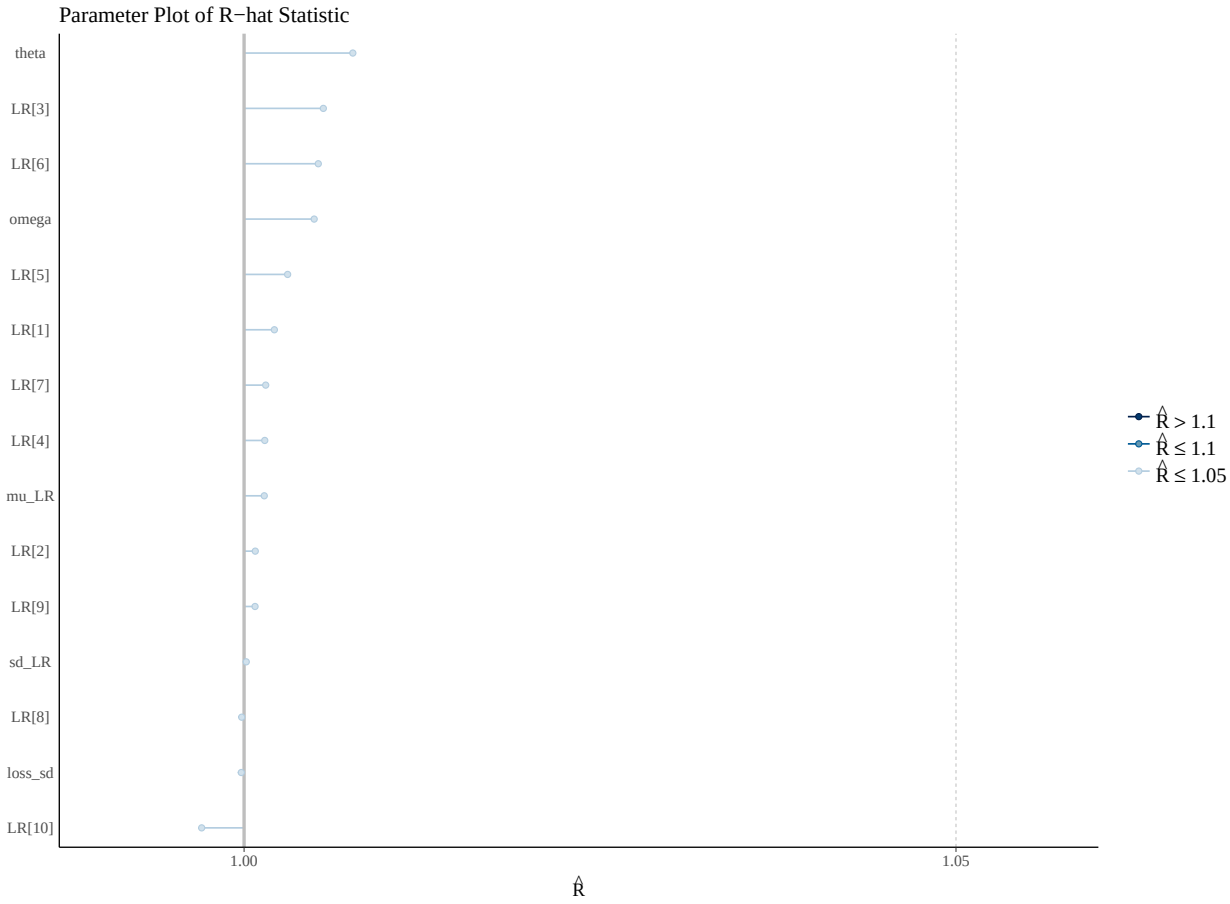
Using `bayesplot` Functionality

The packages `bayesplot` provides some simple diagnostic plots for fits, so we look at those, restricting ourselves to `omega`, `theta`, `LR` and `loss_sd`.

```
stanmodel_pars <- c('omega','theta','LR','mu_LR','sd_LR','loss_sd')
```

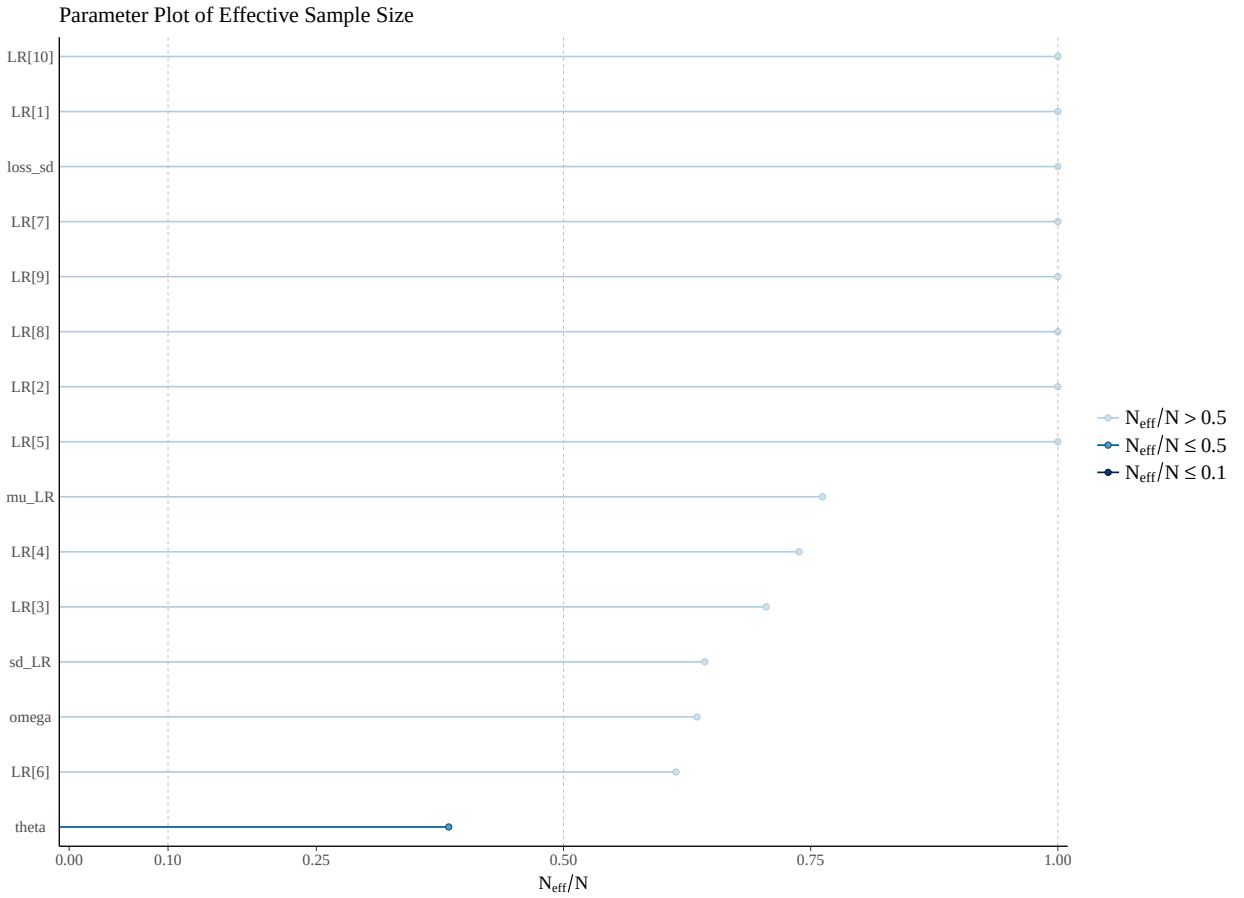
We have some simple lineplots for $\hat{R}$, and this should convey similar information to our previous plots.

```
model_sislob_stanfit %>%
    rhat(pars = stanmodel_pars) %>%
    mcmc_rhat(.) +
        yaxis_text() +
        ggtitle("Parameter Plot of R-hat Statistic")
```



Related to this is n_{eff}:

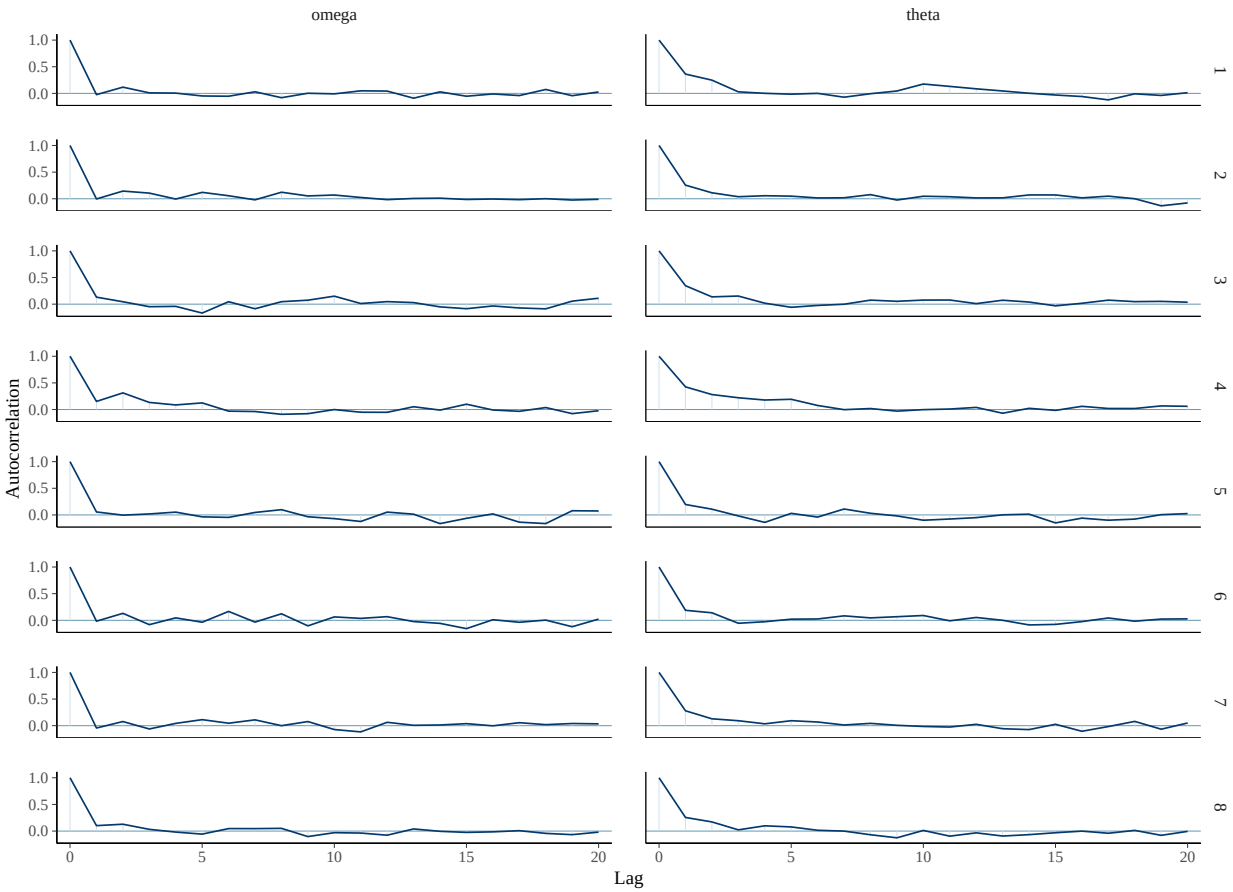
```
model_sislob_stanfit %>%
    neff_ratio(pars = stanmodel_pars) %>%
    mcmc_neff(.) +
        yaxis_text() +
        ggtitle("Parameter Plot of Effective Sample Size")
```



The lowest sample size is just under 50% of the full count, which is good.

`bayesplot` provides functionality to plot the autocorrelation in parameters, and we look at `omega` and `theta` first.

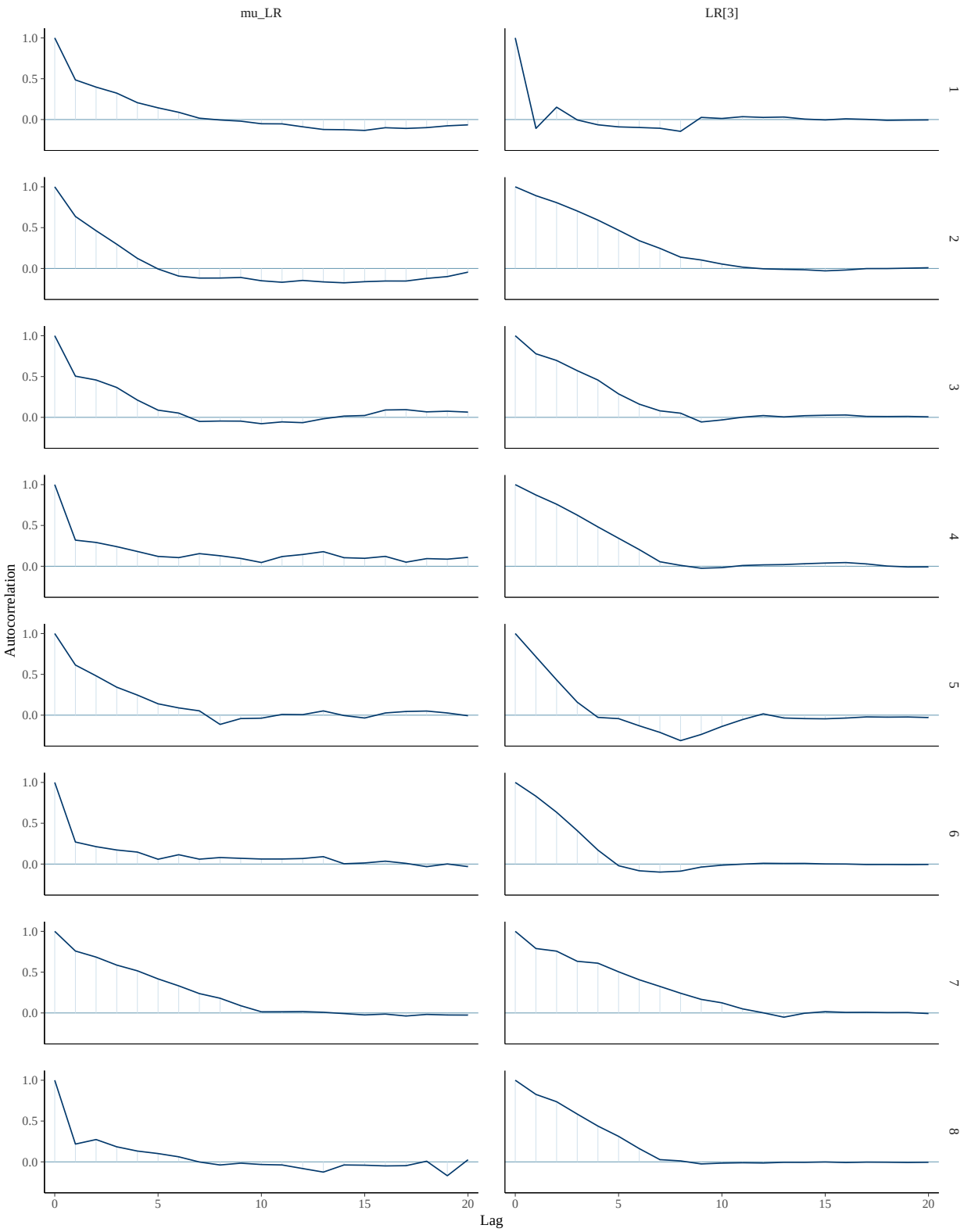
```
model_sislob_stanfit %>%
    extract(inc_warmup = FALSE, permuted = FALSE) %>%
    mcmc_acf(pars = c('omega','theta'))
```



We see the autocorrelation damps down to zero after a few lags.

We will not look at all the loss ratio parameters as there are too many, so we restrict our plots to `mu_LR` and `LR[3]`

```
model_sislob_draws %>% mcmc_acf(pars = c('mu_LR','LR[3]'))
```



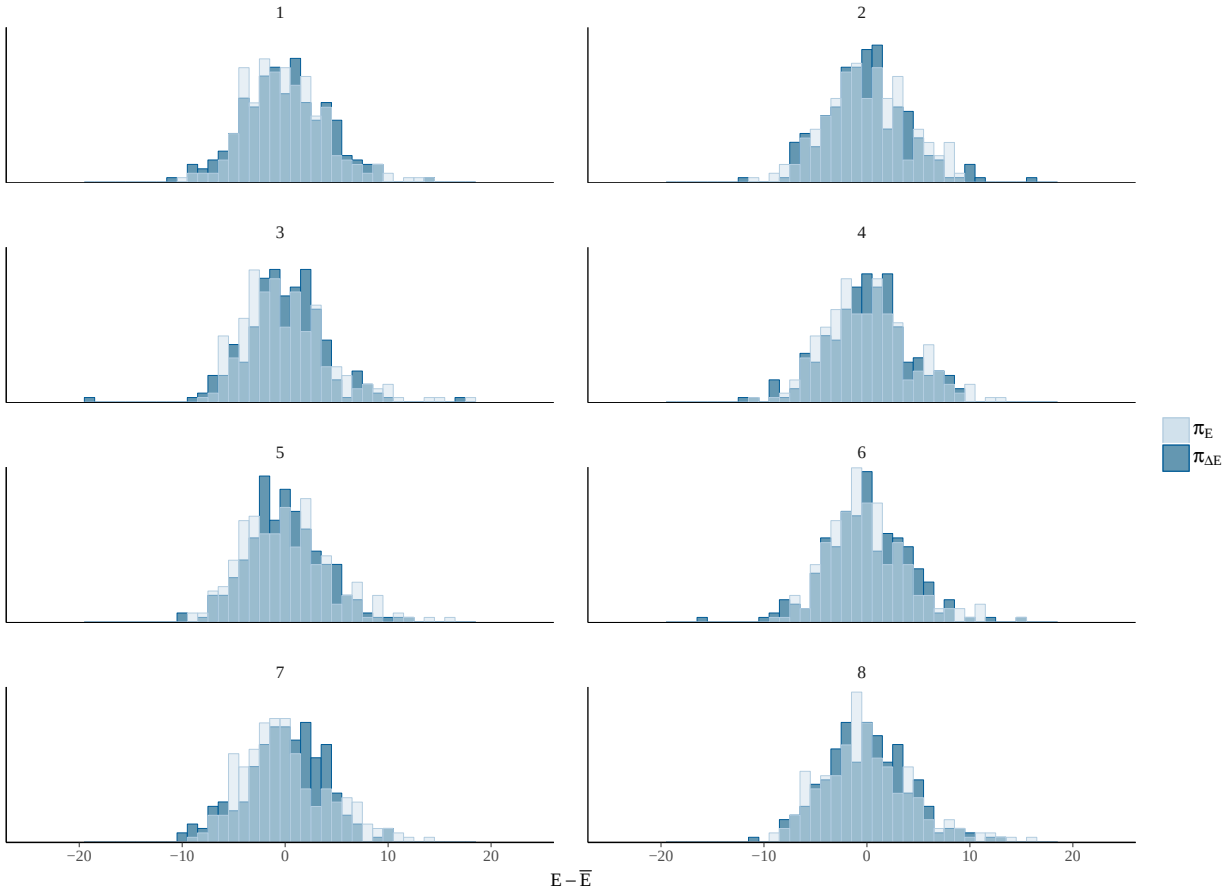
These parameters possess more autocorrelation, taking longer to decay to zero. This is consistent with the $n_{texteff}$ plots, as these parameters have lower effective sample size.

A major benefit of using Hamiltonian Monte Carlo and the NUTS sampler is the presence of powerful

diagnostic tools for convergence. One useful plot is the energy diagnostic.

If the two histograms broadly overlap, the sampler is doing a good job of exploring the posterior probability space.

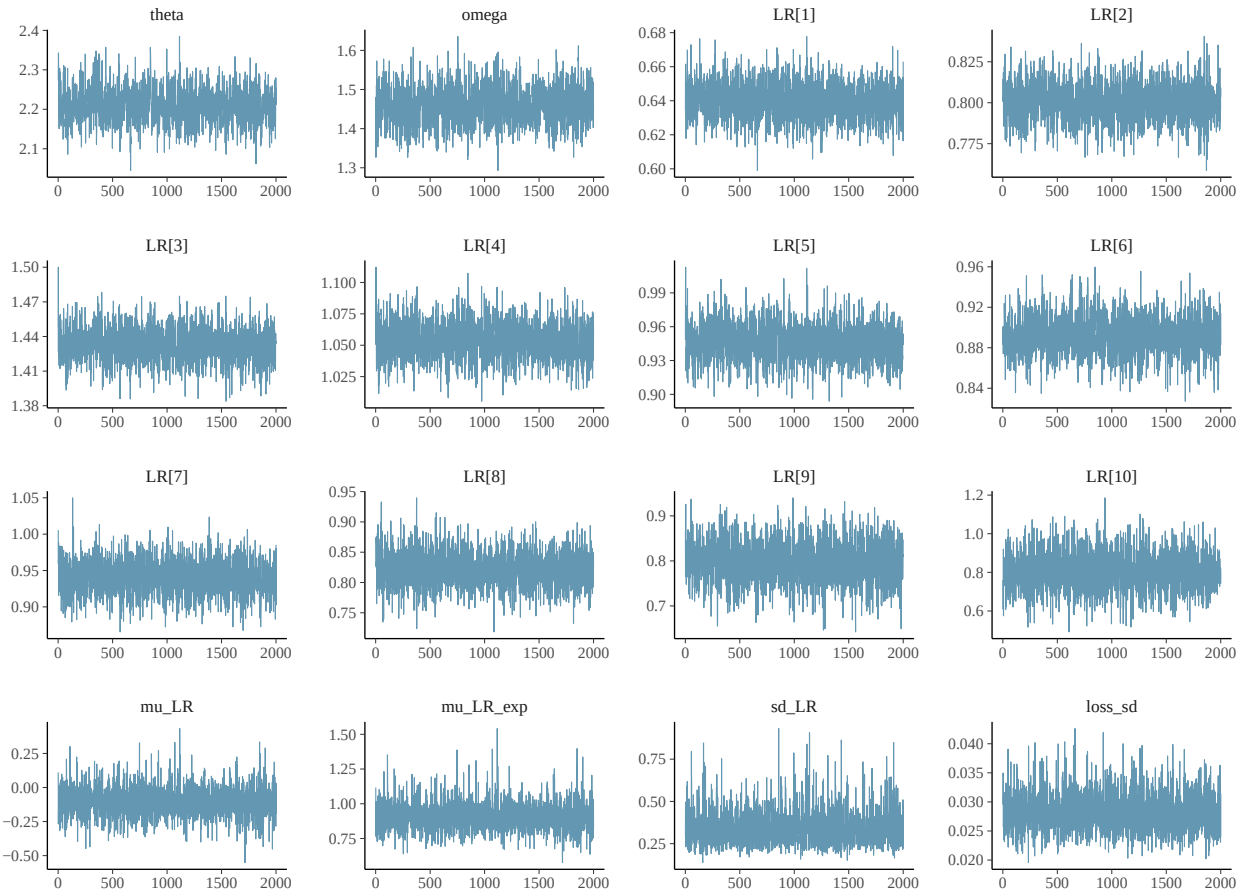
```
model_sislob_stanfit %>%
    nuts_params %>%
    mcmc_nuts_energy(binwidth = 1) +
        facet_wrap(~Chain, ncol = 2)
```



Once again, the diagnostics are not flagging any potential issues with the sample.

Finally we check the parameter traceplots for convergence, looking for indications of a lack of mixing.

```
model_sislob_stanfit %>%
    as.matrix %>%
    mcmc_trace(regex_pars = c('theta','omega','LR\\[','mu_LR','sd_LR','loss_sd'))
```



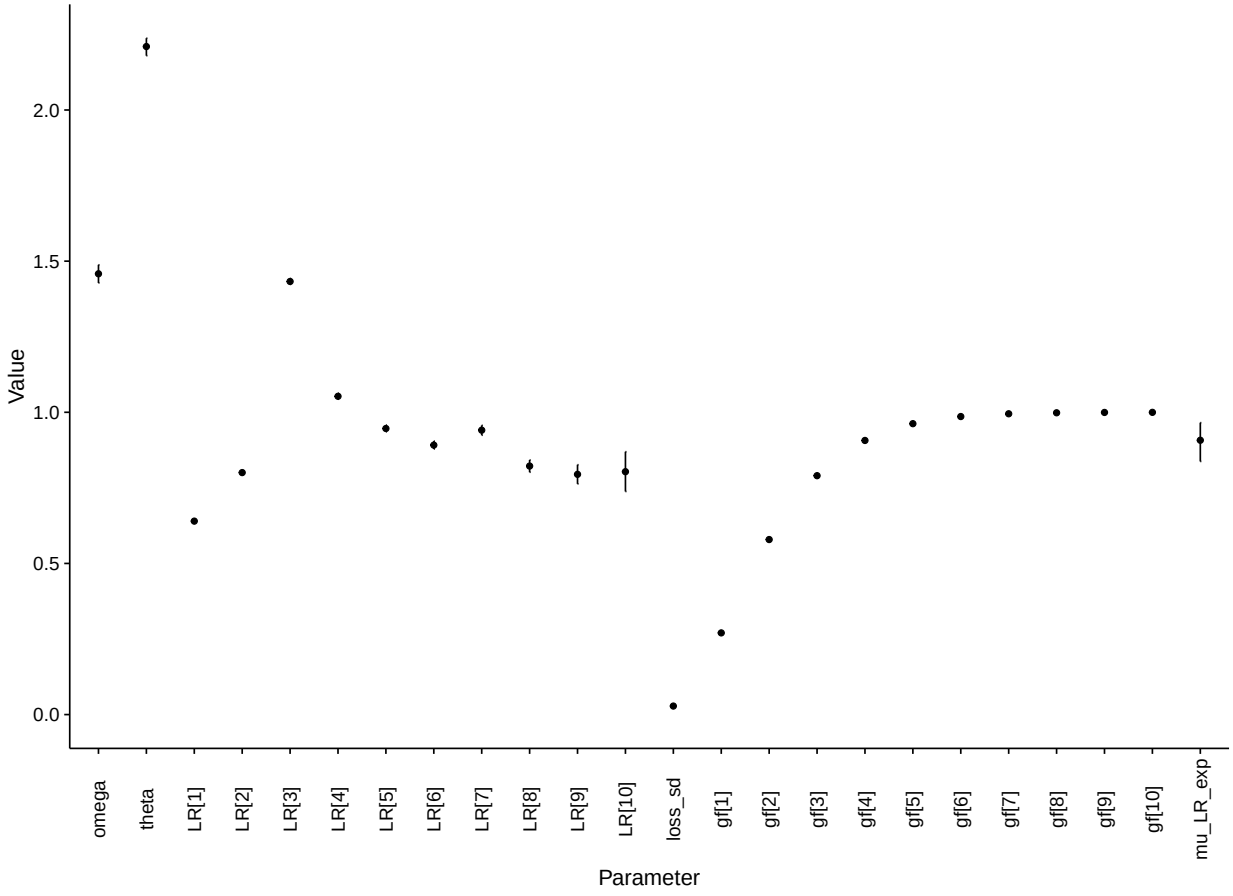
These plots are similar to the traceplots from before, and show no causes for concern.

Assessing the Fit

Having convinced ourselves that the samples have converged, we proceed to checking the quality of the fit. We first look at the 50% credibility intervals of the parameters

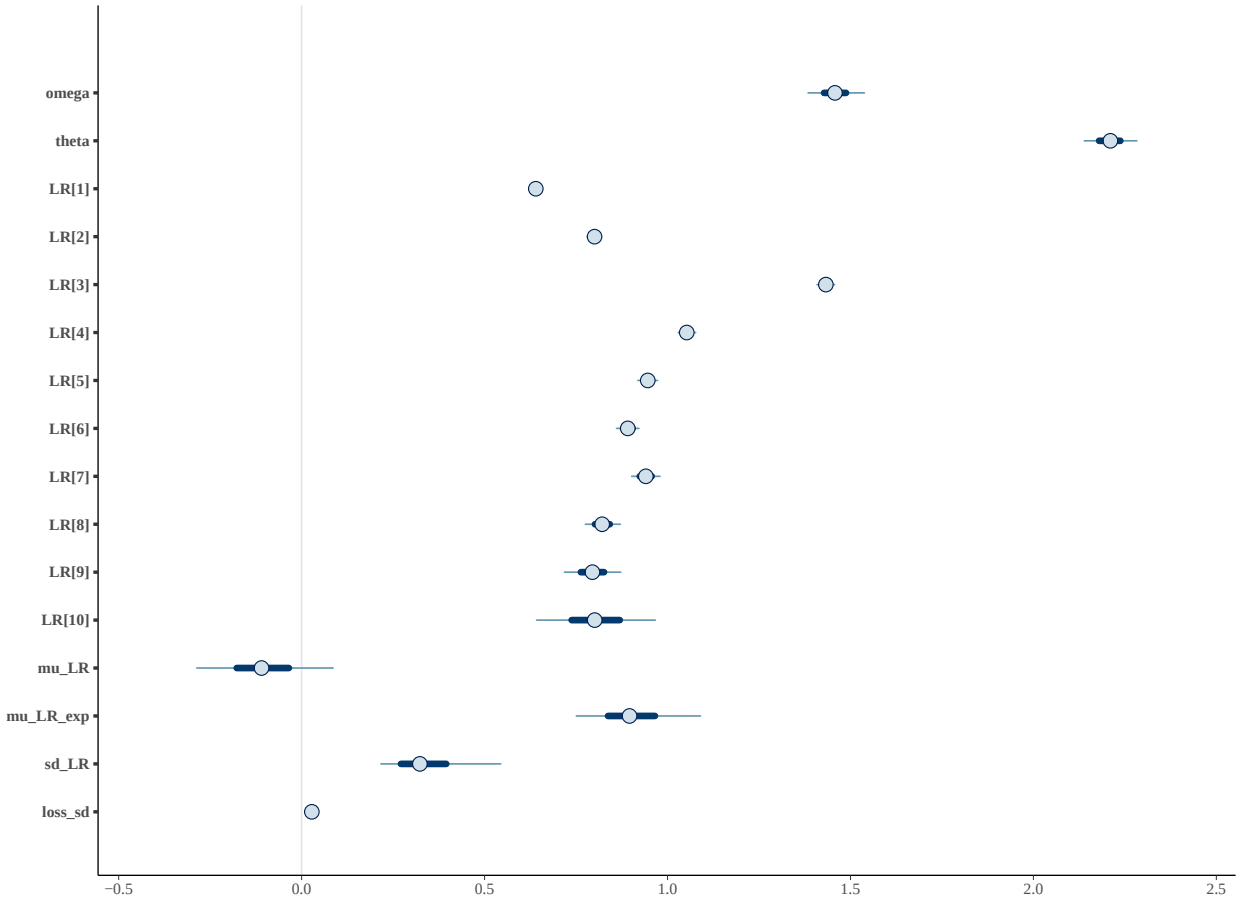
```
param_root <- c("omega", "theta", "LR", "mu_LR_exp", "gf", "loss_sd")

use_vars <- model_sislob_monitor_tbl %>%
    filter(parameter %in% param_root) %>%
    .[["variable"]]

plotdata_tbl <- model_sislob_monitor_tbl %>%
    filter(variable %in% use_vars) %>%
    select(mean, `25%`, `50%`, `75%`) %>%
    mutate(variable = factor(use_vars, levels = use_vars))

ggplot(plotdata_tbl) +
    geom_point(aes(x = variable, y = mean)) +
    geom_errorbar(aes(x = variable, ymin = `25%`, ymax = `75%`), width = 0) +
    expand_limits(y = 0) +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
    xlab("Parameter") +
```

```
    ylab("Value")
```



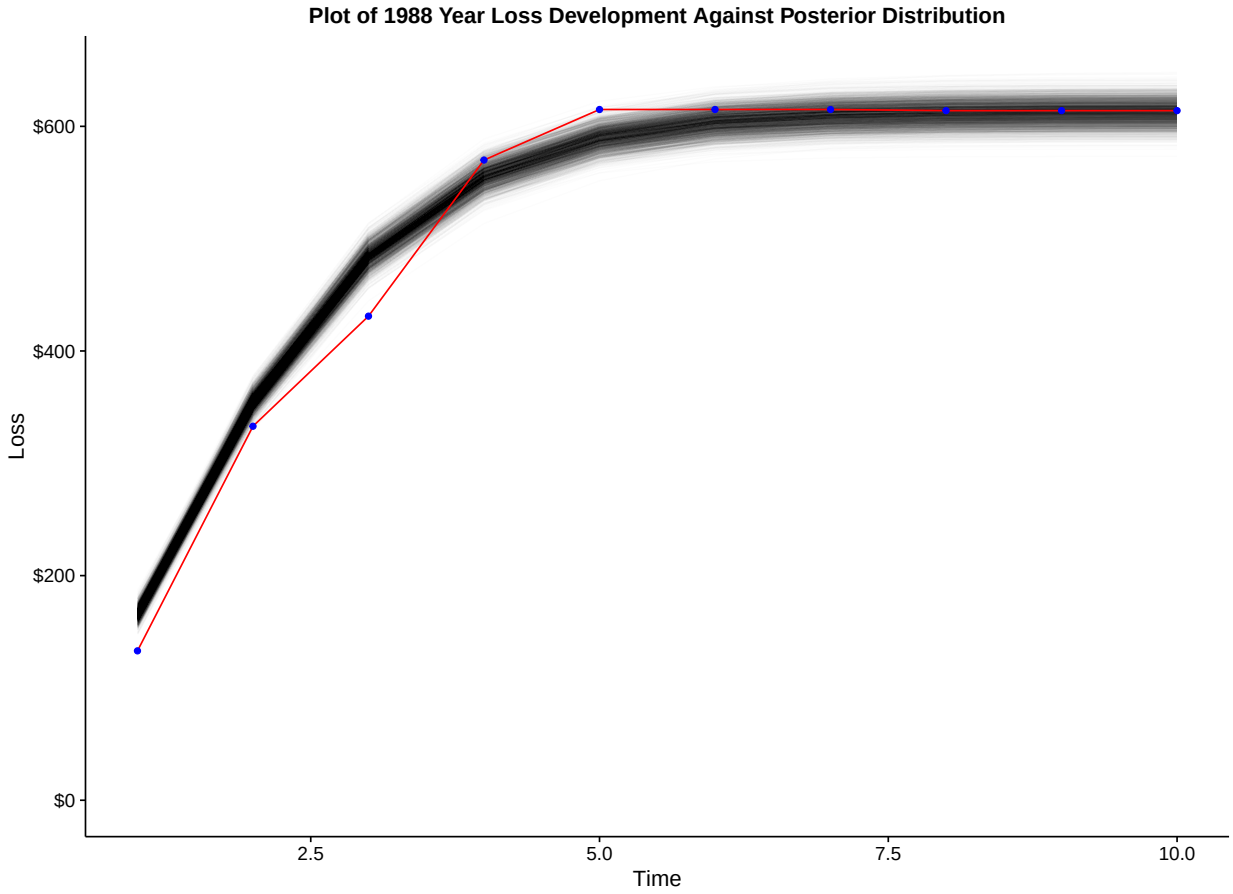
The `bayesplot` package also provides some functionality for plotting the parameters values.

```
weibull_param_plot <- model_sislob_stanfit %>%
    extract(inc_warmup = FALSE, permuted = FALSE) %>%
    mcmc_intervals(regex_pars = c('omega','theta','LR\\[', 'mu_LR','sd_LR','loss_sd')) +
        expand_limits(x = c(-0.5, 2.5))

plot(weibull_param_plot)
```



Now that we have our fit we can start looking at some plots for it. First we look at some very simple sanity-check plots. We look at the full development of an accounting year and see how well our model fits the pattern observed from the data.

```
fitted_curves_tbl <- extract(model_sislob_stanfit)$loss_sample[,1,] %>%
    as_data_frame() %>%
    mutate(iter = 1:n()) %>%
    gather("timelbl", "value", -iter) %>%
    mutate(time = gsub("V", "", timelbl) %>% as.numeric())

ggplot(snapshot_tbl %>% filter(acc_year == 1988)) +
    geom_line (aes(x = time, y = value, group = iter)
               ,data = fitted_curves_tbl, alpha = 0.01) +
    geom_line (aes(x = dev_lag, y = cum_loss), colour = 'red') +
    geom_point(aes(x = dev_lag, y = cum_loss), colour = 'blue') +
    expand_limits(y = 0) +
    scale_y_continuous(labels = dollar) +
    ggtitle("Plot of 1988 Year Loss Development Against Posterior Distribution") +
    xlab("Time") +
    ylab("Loss")
```



The fit seems reasonable.

Predicting Future Patterns

A more interesting question is how we use this model to predict the evolution of the development patterns. How do we condition the model on the exact observed data for a given accounting year so that we can project the patterns forward in time?

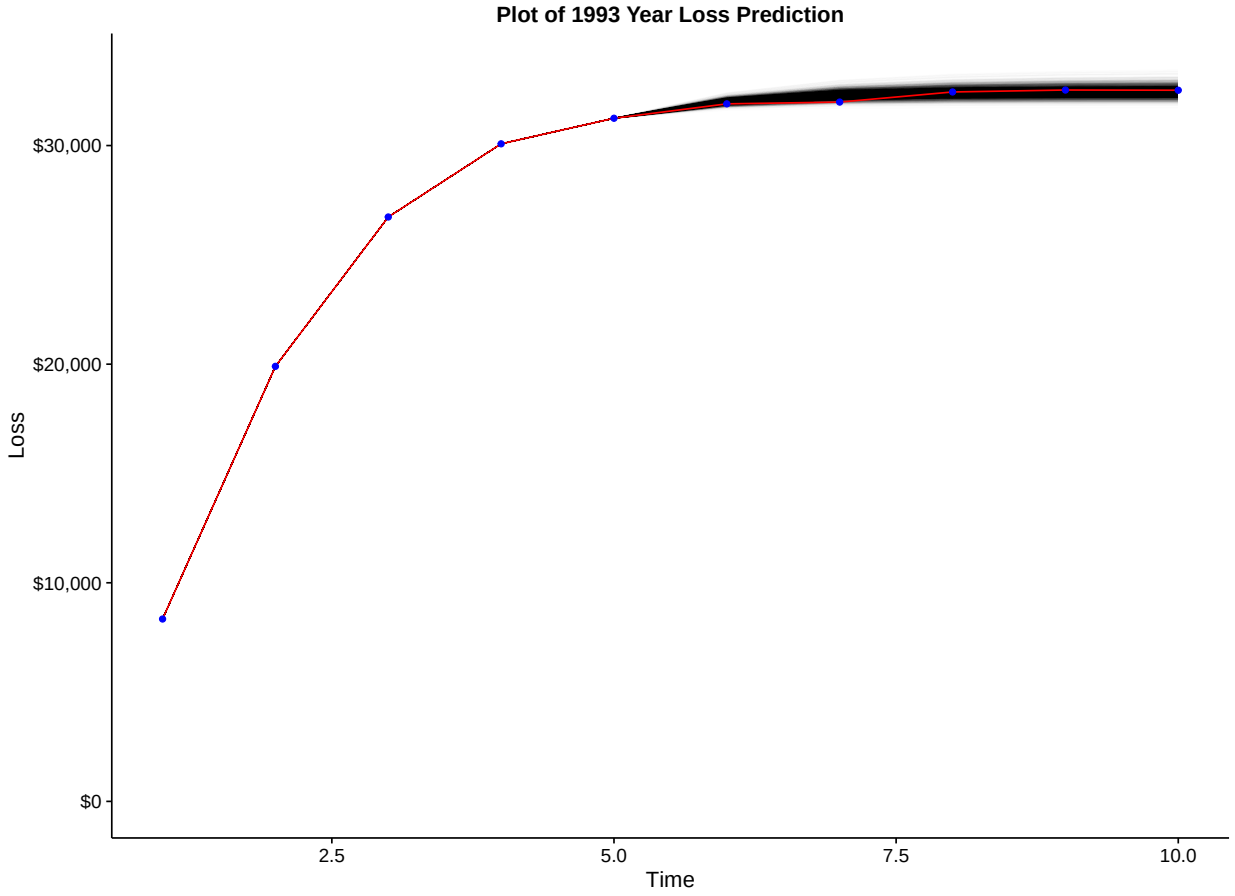
A number of approaches are possible here, we take a proportional growth approach: we look at the percentage change of the growth pattern from one timestep to the next and then apply that forward from the end of the observed data to fill out the remainder of the curve.

We perform these calculations in the `generated quantities` block of the Stan model specification. This makes these values available in the output of the model fit, so all that remains is for us to extract these values and plot them.

We use this approach to predict future losses on the accounting year cohort. We start with 1993, where we have development of five years.

```
predict_cone_tbl <- extract(model_sislob_stanfit)$loss_prediction[,6,] %>%
    as_data_frame() %>%
    mutate(iter = 1:n()) %>%
    gather("timelbl", "value", -iter) %>%
    mutate(time = gsub("V", "", timelbl) %>% as.numeric())

plot_predict <- ggplot(carrier_full_tbl %>% filter(grcode == 43, acc_year == '1993')) +
```

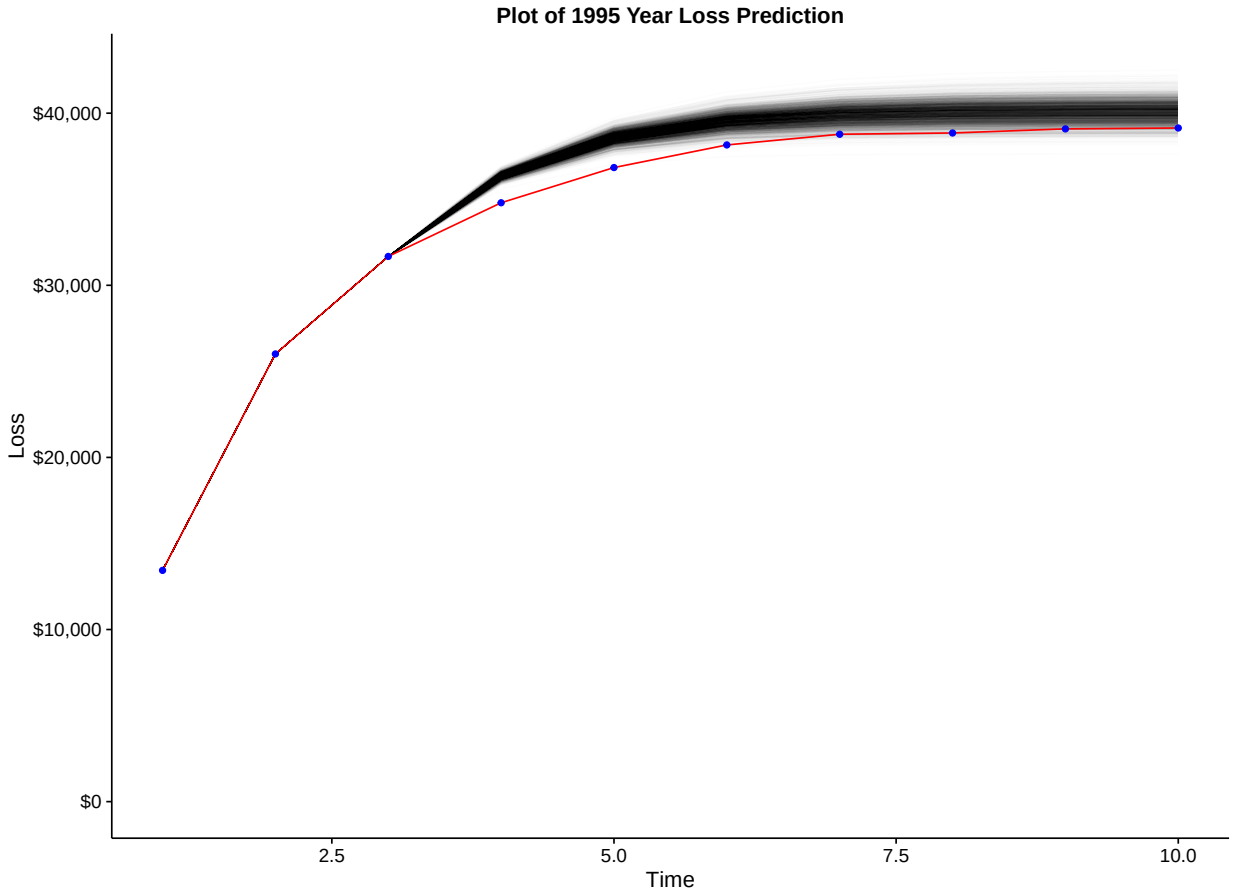
```
    geom_line (aes(x = time, y = value, group = iter)
               ,data = predict_cone_tbl, alpha = 0.01) +
    geom_line (aes(x = dev_lag, y = cum_loss), colour = 'red') +
    geom_point(aes(x = dev_lag, y = cum_loss), colour = 'blue') +
    expand_limits(y = 0) +
    scale_y_continuous(labels = dollar) +
    ggtitle("Plot of 1993 Year Loss Prediction") +
    xlab("Time") +
    ylab("Loss")

plot(plot_predict)
```



We now look at a later year with less claim development, 1995. We have three data points along this development pattern so we will have more uncertainty in our inference on the mean of the losses for this accounting year.

```
predict_cone_tbl <- extract(model_sislob_stanfit)$loss_prediction[,8,] %>%
    as_data_frame() %>%
    mutate(iter = 1:n()) %>%
    gather("timelbl", "value", -iter) %>%
    mutate(time = gsub("V", "", timelbl) %>% as.numeric())

plot_predict <- ggplot(carrier_full_tbl %>% filter(grcode == 43, acc_year == '1995')) +
    geom_line (aes(x = time, y = value, group = iter)
               ,data = predict_cone_tbl, alpha = 0.01) +
```

```
    geom_line (aes(x = dev_lag, y = cum_loss), colour = 'red') +
    geom_point(aes(x = dev_lag, y = cum_loss), colour = 'blue') +
    expand_limits(y = 0) +
    scale_y_continuous(labels = dollar) +
    ggtitle("Plot of 1995 Year Loss Prediction") +
    xlab("Time") +
    ylab("Loss")

plot(plot_predict)
```



Note that in the above plots, we compare the observed data against estimates of the *mean* of the distribution of the losses at each point in time. We do not include the variance around the mean so it is not surprising to observe data outside this cone of uncertainty.

In future iterations of this model we will attempt to include this additional level of variance in our output.

Posterior Predictive Checks

A very important part of all this is getting a sense of the aspects of the data that the model is not capturing well. A recommended method for doing this is creating *posterior predictive checks* (PPCs), that is, assessing the validity of the model in a certain area by comparing the generated values in the sample against the observed values in the dataset.

There are no standard methods for creating PPCs, instead we need to think of different aspects of our data

and see how well our model is doing at modelling those idiosyncracies in the data.

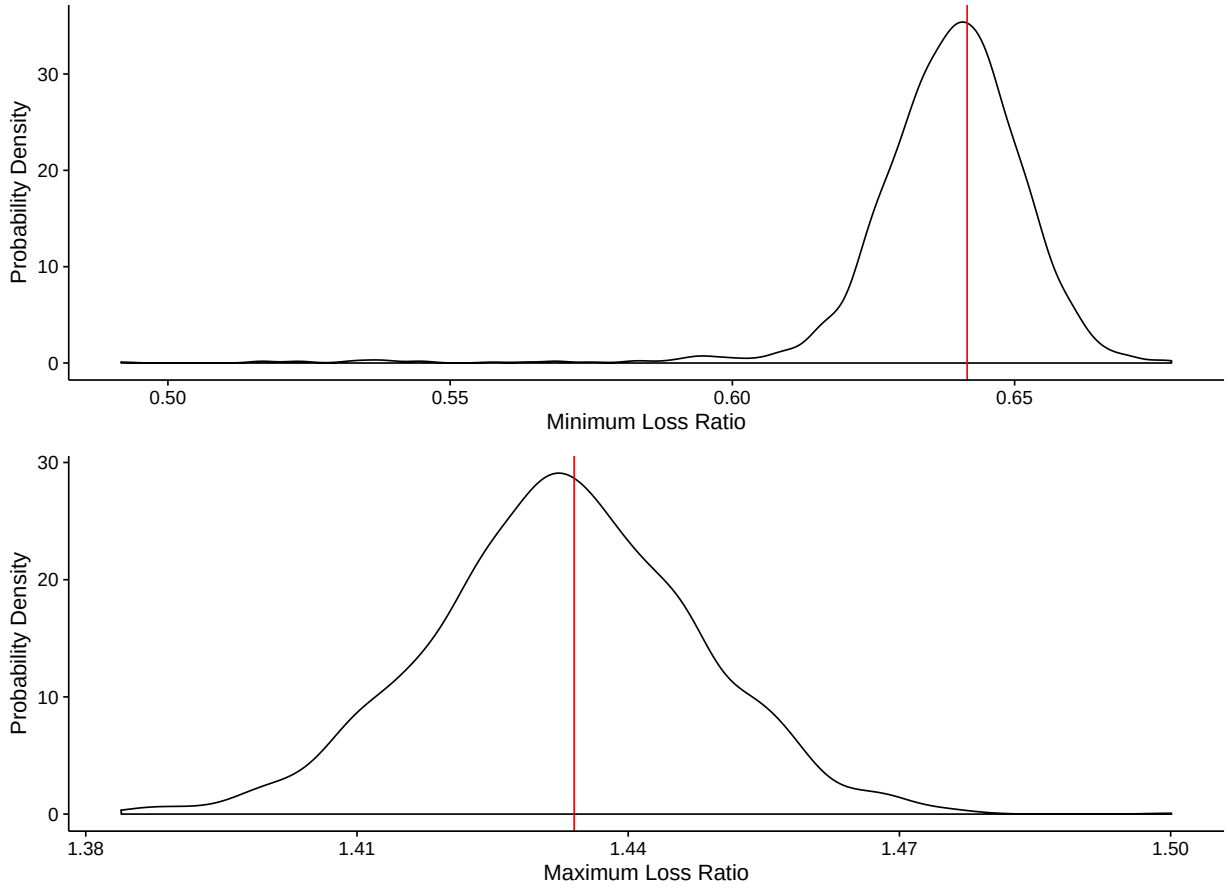
We will look at a number of PPCs for the single insurer dataset here.

Range of Loss Ratios

We first investigate how well the model is doing at capturing the range of Loss Ratios observed in the data: are the largest and smallest predicted Loss Ratios in the model reflecting what we see in the data?

To do this we do a few things: we first add some calculations to the `generated quantities` block in the Stan file. These calculated values are then compared to what we observed in the data to see how well our model does.

```
ppc_min_lr <- extract(model_sislob_stanfit)$ppc_minLR
ppc_max_lr <- extract(model_sislob_stanfit)$ppc_maxLR

lr_tbl <- carrier_full_tbl %>%
    filter(grcode == use_grcode[1]
           ,dev_lag == 10) %>%
    summarise(min_lr = min(loss_ratio)
              ,max_lr = max(loss_ratio))

min_plot <- ggplot() +
    geom_density(aes(x = ppc_min_lr)) +
    geom_vline(aes(xintercept = lr_tbl$min_lr), colour = 'red') +
    xlab("Minimum Loss Ratio") +
    ylab("Probability Density")

max_plot <- ggplot() +
    geom_density(aes(x = ppc_max_lr)) +
    geom_vline(aes(xintercept = lr_tbl$max_lr), colour = 'red') +
    xlab("Maximum Loss Ratio") +
    ylab("Probability Density")


plot_grid(min_plot, max_plot, nrow = 2)
```



Looking at the above plots, we see that the model is doing reasonably well at capturing the spreads of Loss Ratios.

This is not hugely surprising though, as we have only ten accounting years in the dataset and have seen a decent spread of those already in the dataset.

Aggregate Reserve Amounts

While it is useful to break down the loss curves into these different cohorts and model each curve separately, from the point of view of an insurance company we care much less about each year's estimates as we do about the overall amount of money we need to hold back to cover claims.

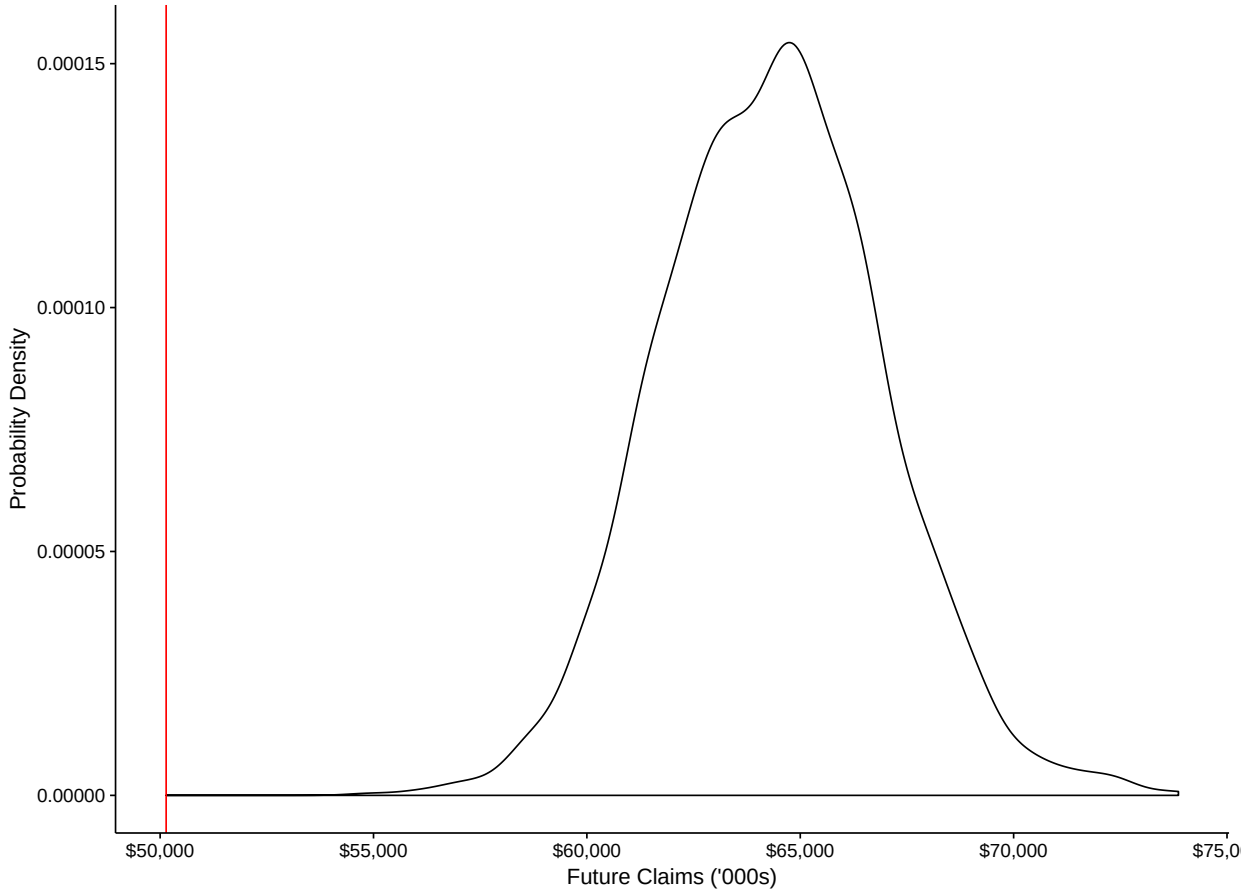
This presents us with a way to run a check: how well does our model do at estimating the reserves required for all the accounting years put together. We hope that while each accounting year will have mistakes, over-estimates and under-estimates will tend to cancel each other somewhat. Thus, we calculate the total future claims for the book as a whole at 1998 and then compare that to the actual final amounts observed in the data.

As this process may still be a little vague, we will be explicit:

- For each accounting year y, we look at the current claim level at the point of modelling, 1998. This gives us ten values as we have ten accounting years.
- We add these ten numbers to calculate $TCKC_{1998}$, the total of current known claims as at 1998.
- For each iteration in the sample, we project forward the estimate for the final amount of claims for each accounting year. Summing across the accounting year we end up with a sample of expected final claims, EFC_{1998}.

- With a sample of this value, we then compare it to the actual, observed values of the variable, AFC_{1998} and see how the sample values are distributed around the data-calculated value.

```
tckc <- carrier_snapshot_tbl %>%
    filter(grcode == use_grcode[1]) %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum

afc <- carrier_full_tbl %>%
    filter(grcode == use_grcode[1]) %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum

future_claims <- afc - tckc


ggplot() +
    geom_density(aes(x = extract(model_sislob_stanfit)$ppc_EFC)) +
    geom_vline(aes(xintercept = future_claims), colour = 'red') +
    scale_x_continuous(labels = dollar) +
    xlab("Future Claims ('000s)") +
    ylab("Probability Density")
```



The Log-logistic Growth Model

We now repeat all of the above work, using the log-logistic functional form for the growth factors instead.

The code is broadly similar, and we have wrapped the code into a single function which takes the input data and creates the fit.

```
model_sislob_ll_list <- create_stanfit(model_sislob_stanmodel, usedata_tbl
                                       ,model_id = 0, stan_seed = stan_seed)

model_sislob_ll_stanfit <- model_sislob_ll_list$stanfit
```

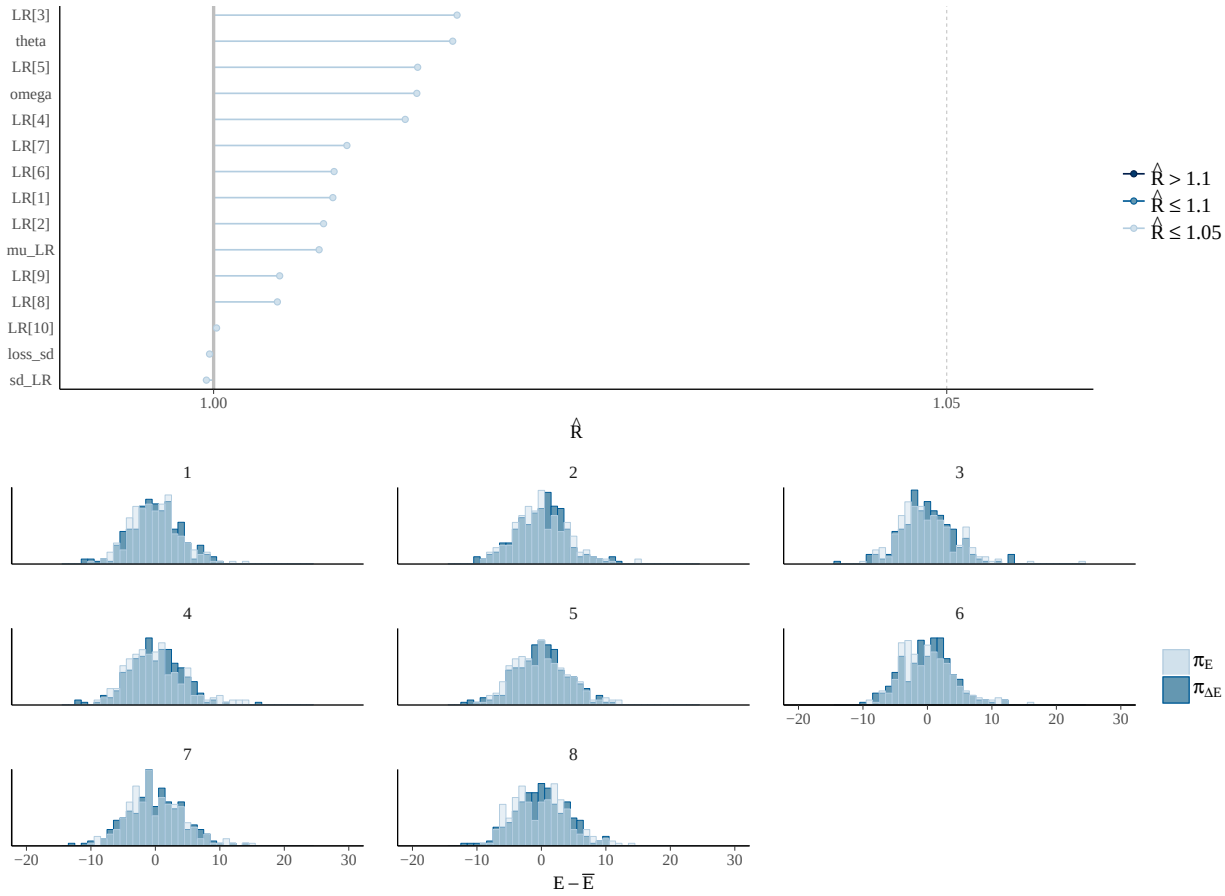
We now repeat the previous exercise of checking that the sample has converged through the use of the various diagnostic tools employed before.

Diagnostic Plots

Rather than go through a comprehensive list of diagnostic plots, we look at a smaller selection of them for the interests of brevity.

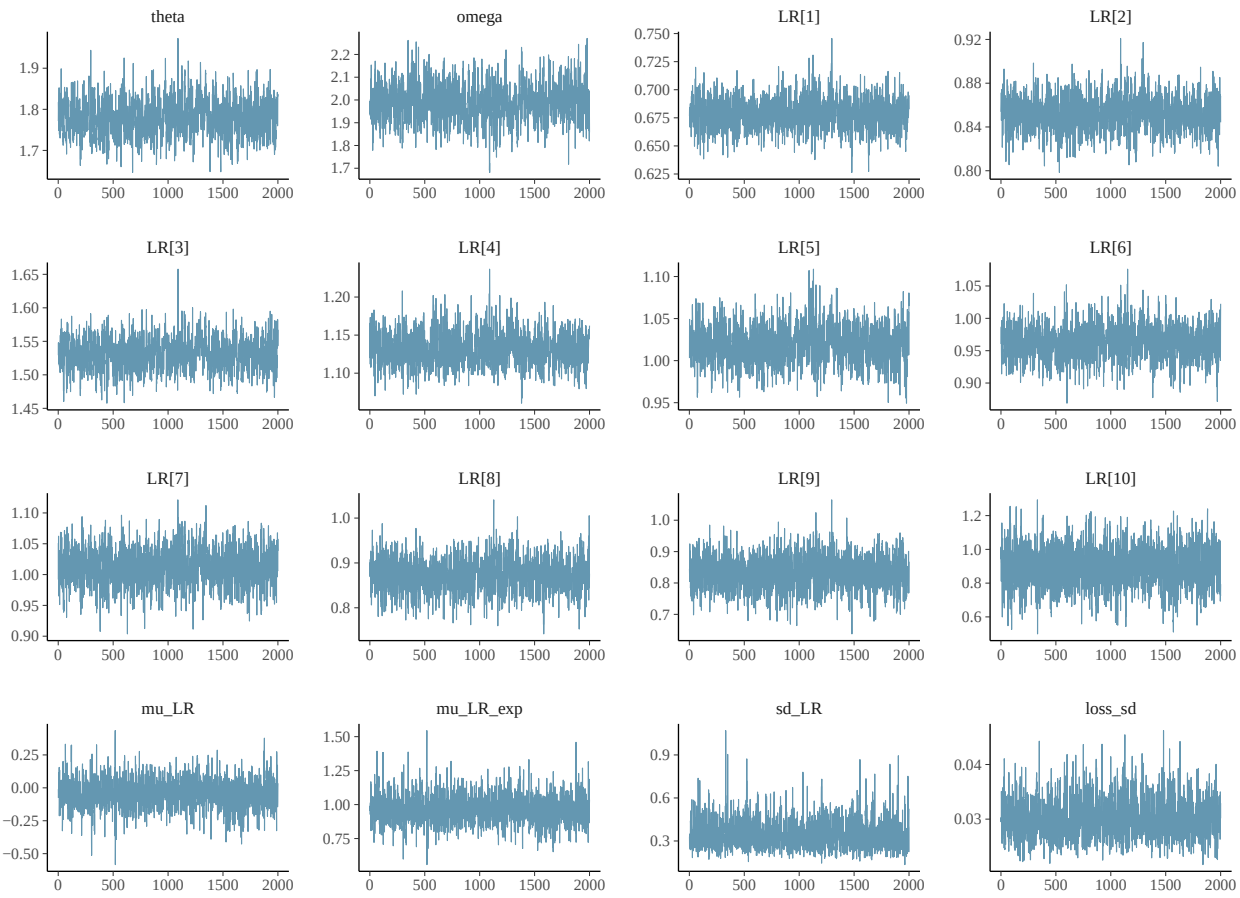
```
rhat_ll_plot <- model_sislob_ll_stanfit %>%
    rhat(pars = stanmodel_pars) %>%
    mcmc_rhat(.) + yaxis_text()
```

```
energy_ll_plot <- model_sislob_ll_stanfit %>%
    nuts_params %>%
    mcmc_nuts_energy(binwidth = 1)

plot_grid(rhat_ll_plot
         ,energy_ll_plot
         ,nrow = 2)
```



Finally we look at the parameter traceplots to look for signs that the chains have not mixed.

```
model_sislob_ll_stanfit %>%
    as.matrix %>%
    mcmc_trace(regex_pars = c('theta','omega','LR\\[','mu_LR','sd_LR','loss_sd'))
```

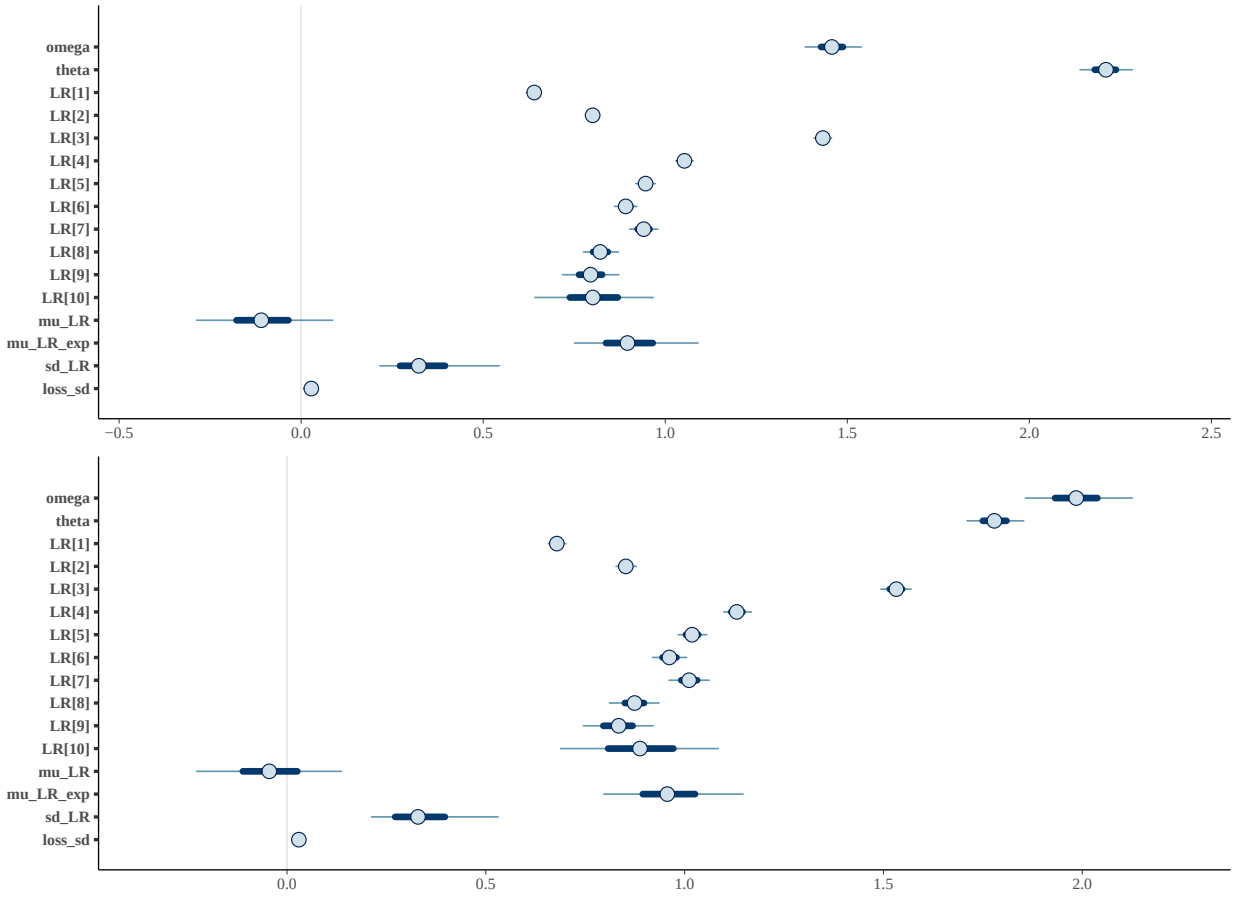


The chains appear to have mixed, none of the diagnostics have alerted us to anything and there are no divergent transitions, so we can proceed with checking the model.

Parameter Values

We look at the parameter outputs, getting broadly similar plots to those we saw before.

```
loglogistic_param_plot <- model_sislob_ll_stanfit %>%
    extract(inc_warmup = FALSE, permuted = FALSE) %>%
    mcmc_intervals(., regex_pars = c('omega','theta','LR\\[', 'mu_LR','sd_LR','loss_sd')) +
        expand_limits(x = c(-0.5, 2.5))

plot_grid(weibull_param_plot
         ,loglogistic_param_plot
         ,nrow = 2)
```



The important parameters are similar across the two functional forms, though ω and θ have different effects in the two forms.

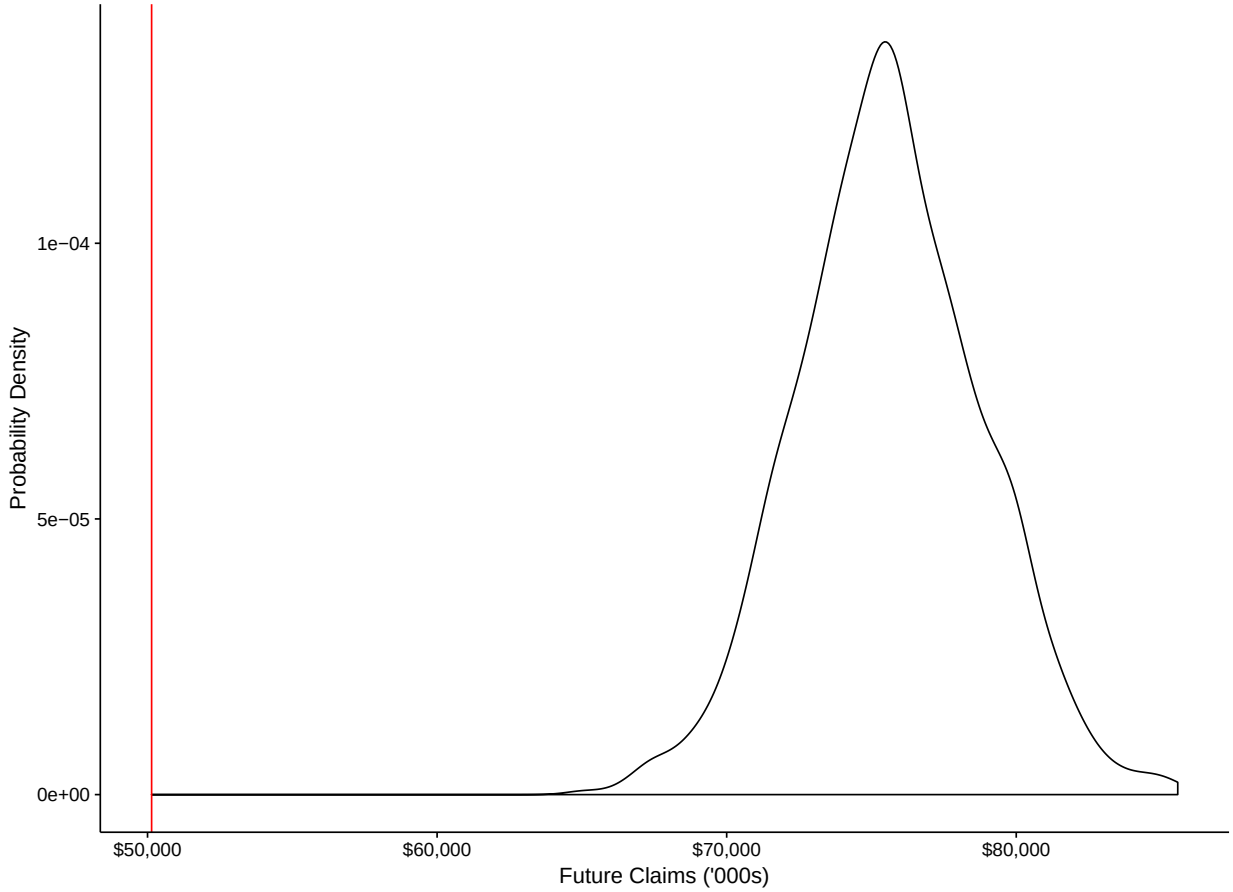
The important distinction is when we look at the posterior predictive checks.

Posterior Predictive Checks

Finally we look at some posterior predictive checks to see if switching to the log-logistic functional form improves on the Weibull.

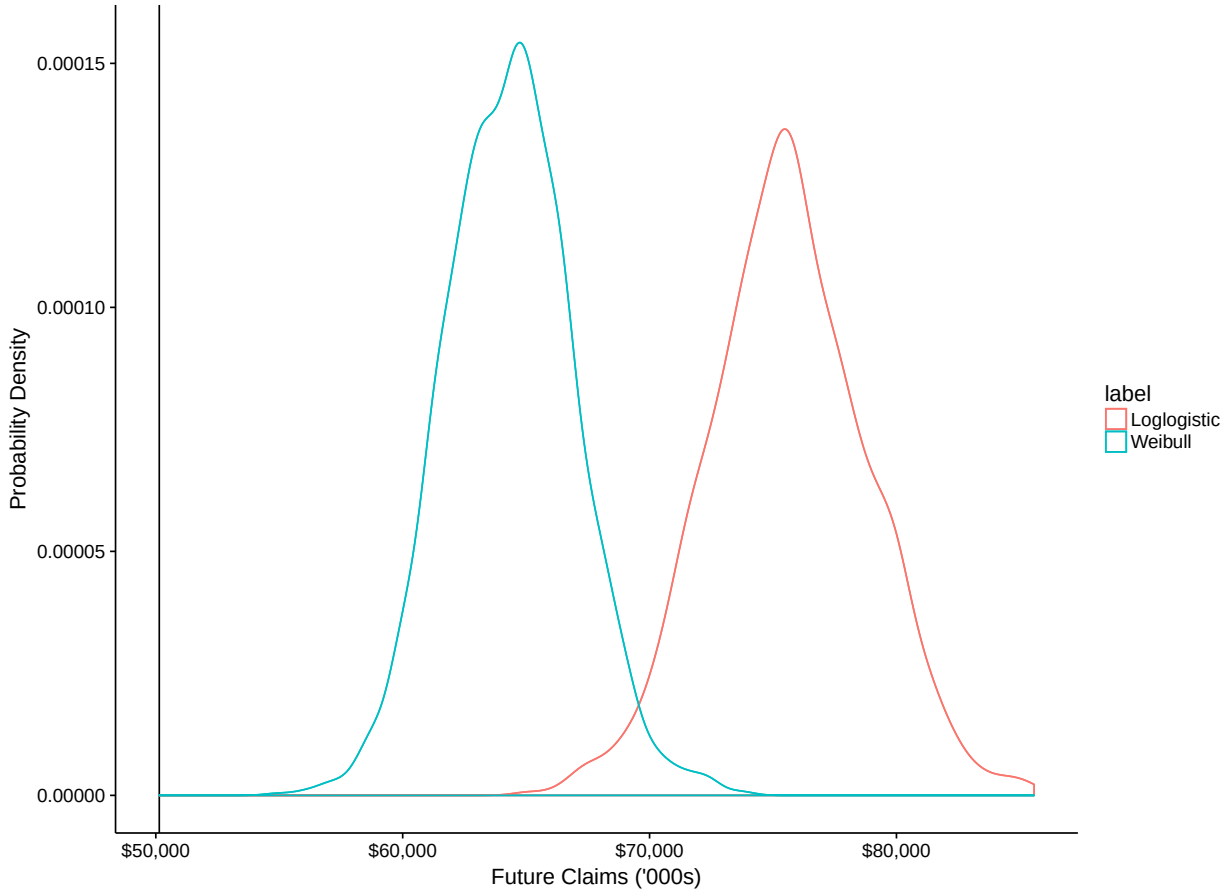
```
tckc <- carrier_snapshot_tbl %>%
    filter(grcode == use_grcode[1]) %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum

afc <- carrier_full_tbl %>%
    filter(grcode == use_grcode[1]) %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum

future_claims <- afc - tckc
```

```
ggplot() +
    geom_density(aes(x = extract(model_sislob_ll_stanfit)$ppc_EFC)) +
    geom_vline(aes(xintercept = future_claims), colour = 'red') +
    scale_x_continuous(labels = dollar) +
    xlab("Future Claims ('000s)") +
    ylab("Probability Density")
```



Finally, we compare the two models simultaneously.

```
plot_tbl <- bind_rows(
    data_frame(label  = 'Weibull'
               ,values = extract(model_sislob_stanfit)$ppc_EFC)
   ,data_frame(label = 'Loglogistic'
               ,values = extract(model_sislob_ll_stanfit)$ppc_EFC)
)

ggplot(plot_tbl) +
    geom_vline(aes(xintercept = future_claims)) +
    geom_density(aes(x = values, colour = label)) +
    geom_density(aes(x = values, colour = label)) +
    scale_x_continuous(labels = dollar) +
    xlab("Future Claims ('000s)") +
    ylab("Probability Density")
```



Analysing A New Insurer

It is difficult to assess exactly why the inferences for this set of triangles overestimate the total reserves required - it is possible the extremely high loss ratio from cohort year 1990 (a loss ratio of over 1.4).

Loss Curves for GRCODE 353

We now repeat all of the above processes using an insurer where the loss ratios have less variance, Celina Mutual Group, GRCODE 353:

```
grcode353_tbl <- claimdata_tbl %>%
    filter(lob == 'ppauto', grcode == 353)

grcode353_snapshot_tbl <- grcode353_tbl %>%
    filter(dev_year < 1998)
```

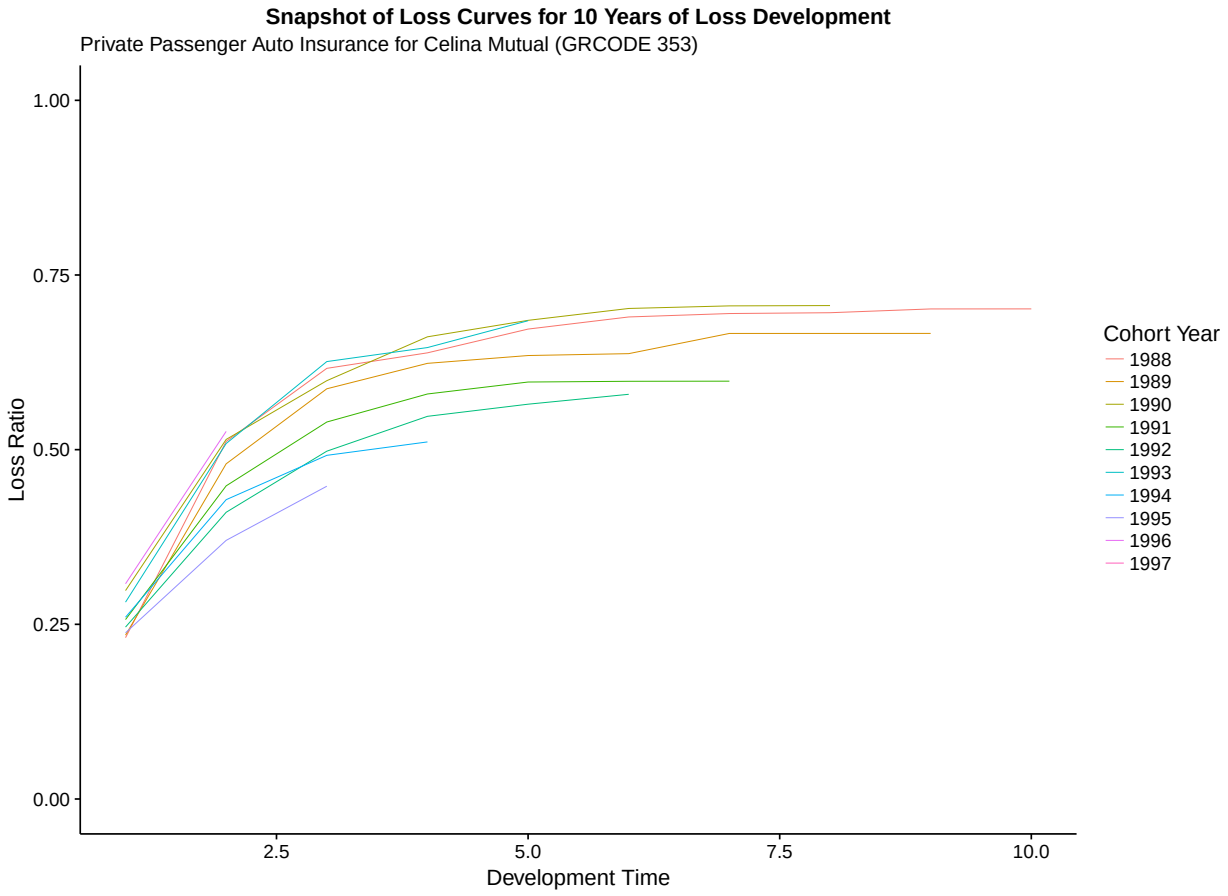
With this data we construct a loss triangle of the loss ratios.

```
grcode353_snapshot_tbl %>%
    select(acc_year, dev_lag, premium, loss_ratio) %>%
    spread(dev_lag, loss_ratio) %>%
    print.data.frame(digits = 2)
```

```
##    acc_year premium    1    2    3    4    5    6    7    8    9  10
## 1      1988   18793 0.23 0.51 0.62 0.64 0.67 0.69 0.69 0.70 0.70 0.7
## 2      1989   18948 0.23 0.48 0.59 0.62 0.63 0.64 0.67 0.67 0.67  NA
## 3      1990   20527 0.30 0.51 0.60 0.66 0.69 0.70 0.71 0.71   NA  NA
## 4      1991   21278 0.26 0.45 0.54 0.58 0.60 0.60 0.60   NA   NA  NA
## 5      1992   20779 0.25 0.41 0.50 0.55 0.57 0.58   NA   NA   NA  NA
## 6      1993   23212 0.28 0.51 0.63 0.65 0.68   NA   NA   NA   NA  NA
## 7      1994   22219 0.26 0.43 0.49 0.51   NA   NA   NA   NA   NA  NA
## 8      1995   18314 0.24 0.37 0.45   NA   NA   NA   NA   NA   NA  NA
## 9      1996   17043 0.31 0.53   NA   NA   NA   NA   NA   NA   NA  NA
## 10     1997   19217 0.30   NA   NA   NA   NA   NA   NA   NA   NA  NA
```

We also plot the curves.

```
ggplot(grcode353_snapshot_tbl) +
    geom_line(aes(x = dev_lag, y = loss_ratio, colour = as.character(acc_year))
              ,size = 0.3) +
    expand_limits(y = c(0, 1)) +
    xlab('Development Time') +
    ylab('Loss Ratio') +
    ggtitle('Snapshot of Loss Curves for 10 Years of Loss Development'
           ,subtitle = 'Private Passenger Auto Insurance for Celina Mutual (GRCODE 353)') +
    guides(colour = guide_legend(title = 'Cohort Year'))
```



Fit Weibull Stan Model on New Data

Now that we have the data, we fit the model using the GRCODE 353.

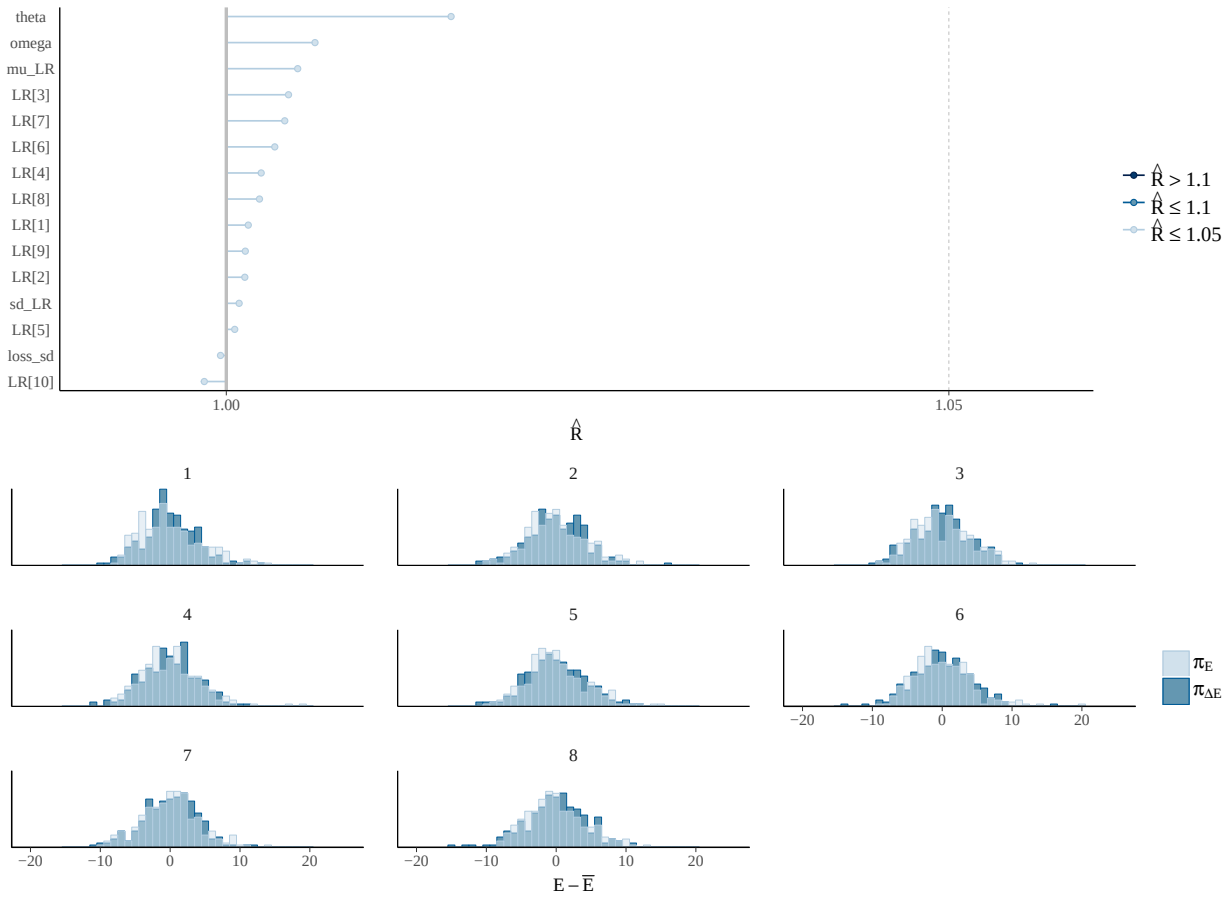
```
model_sislob_grcode353_wb_list <- create_stanfit(model_sislob_stanmodel
                                                 ,grcode353_snapshot_tbl
                                                 ,model_id = 1
                                                 ,stan_seed = stan_seed)

model_sislob_grcode353_wb_stanfit <- model_sislob_grcode353_wb_list$stanfit
```

As before, we first check the output to ensure that our sample does not contain any obvious biases.

Diagnostic Plots

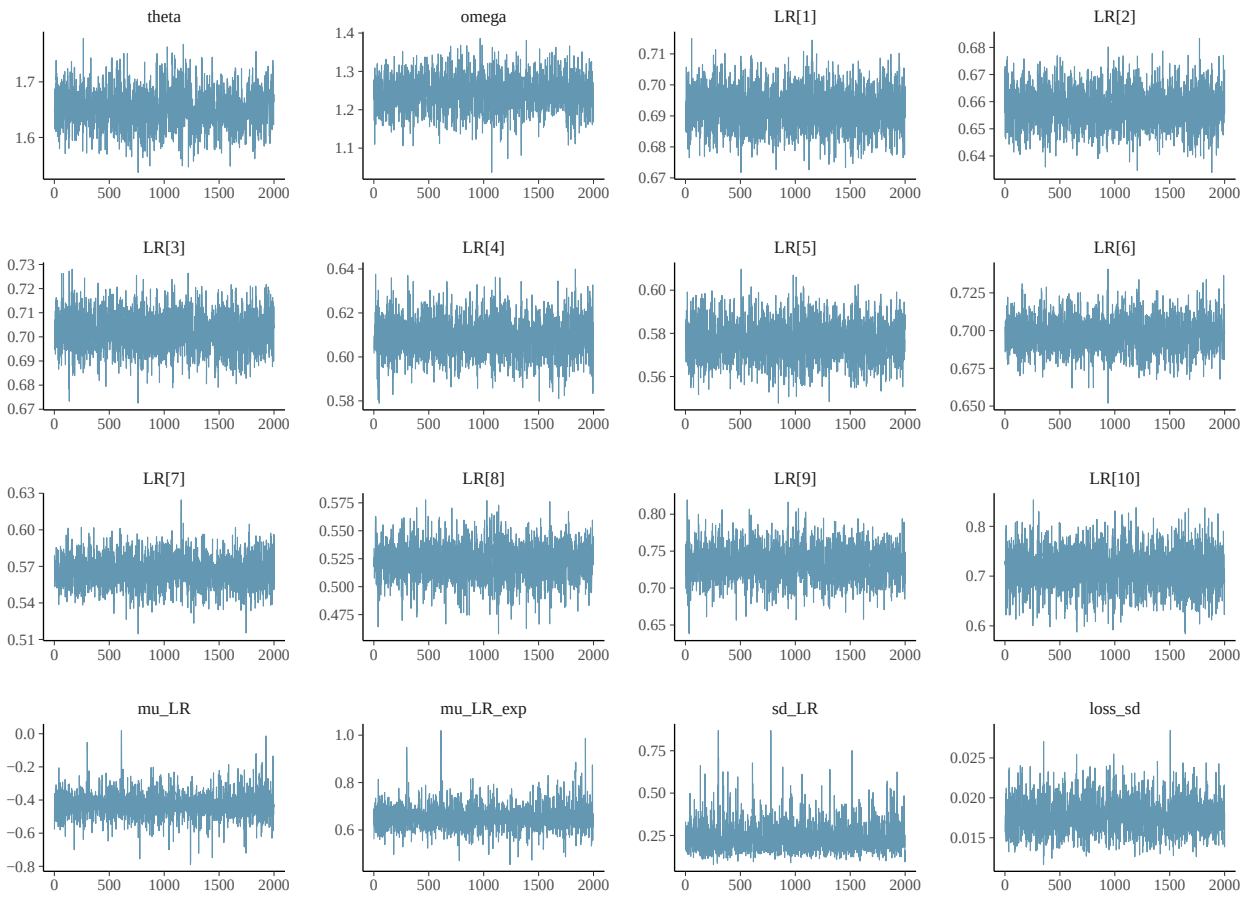
Once again we provide a sample of the diagnostic plots:

```
newins_rhat_plot <- model_sislob_grcode353_wb_stanfit %>%
    rhat(pars = stanmodel_pars) %>%
    mcmc_rhat(.) + yaxis_text()

newins_energy_plot <- model_sislob_grcode353_wb_stanfit %>%
    nuts_params %>%
    mcmc_nuts_energy(binwidth = 1)

plot_grid(newins_rhat_plot
          ,newins_energy_plot
          ,nrow = 2)
```



All of the above look reasonable, so on to the traceplots.

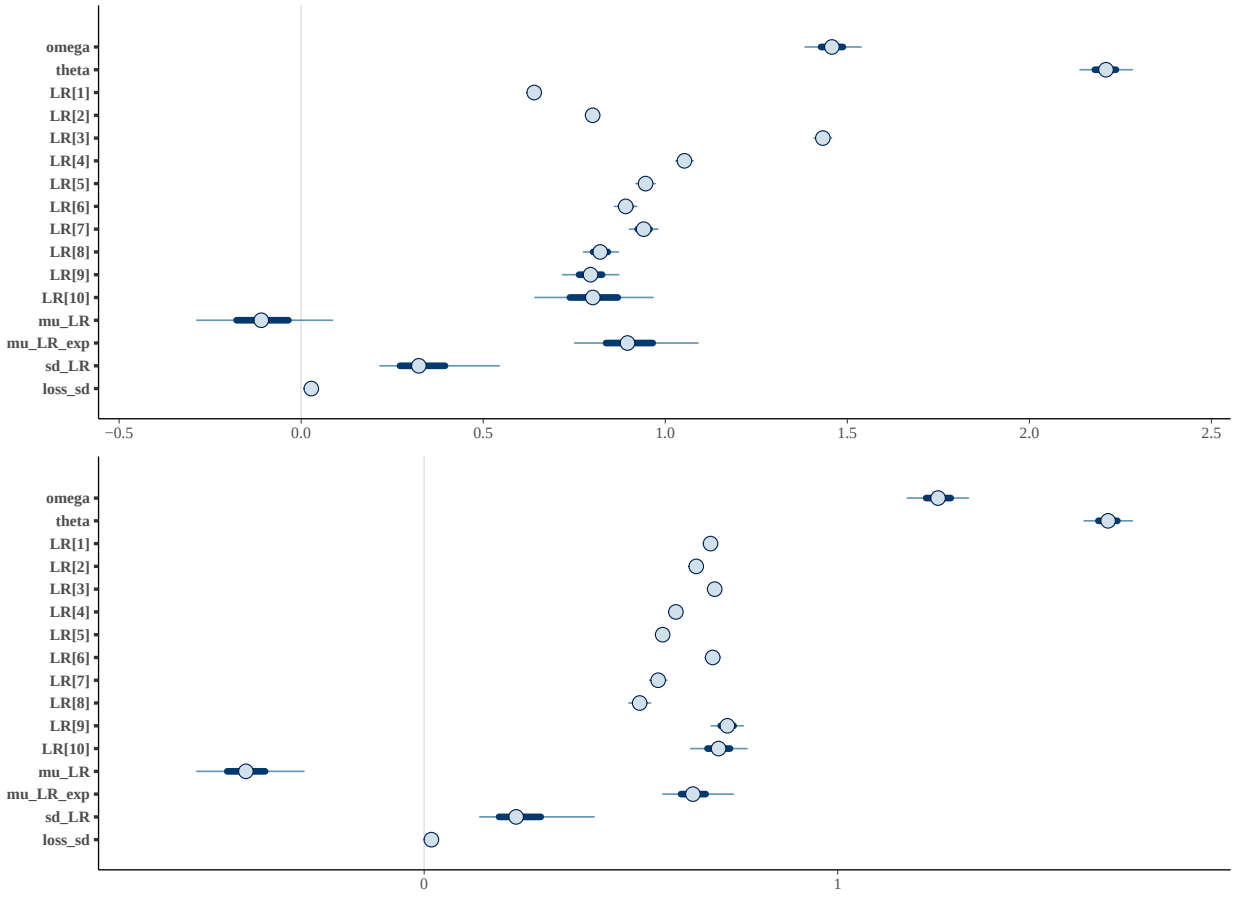
```
model_sislob_grcode353_wb_stanfit %>%
    as.matrix %>%
    mcmc_trace(regex_pars = c('theta','omega','LR\\[','mu_LR','sd_LR','loss_sd'))
```



Parameter Values

Having fit the model and checked the samples, we now compare the inferences on the new dataset against the inferences from the other dataset.

```
gr353_wb_param_plot <- model_sislob_grcode353_wb_stanfit %>%
    extract(inc_warmup = FALSE, permuted = FALSE) %>%
    mcmc_intervals(., regex_pars = c('omega','theta','LR\\[', 'mu_LR','sd_LR','loss_sd')) +
        expand_limits(x = c(-0.5, 2.5))

plot_grid(weibull_param_plot
         ,gr353_wb_param_plot
         ,nrow = 2)
```

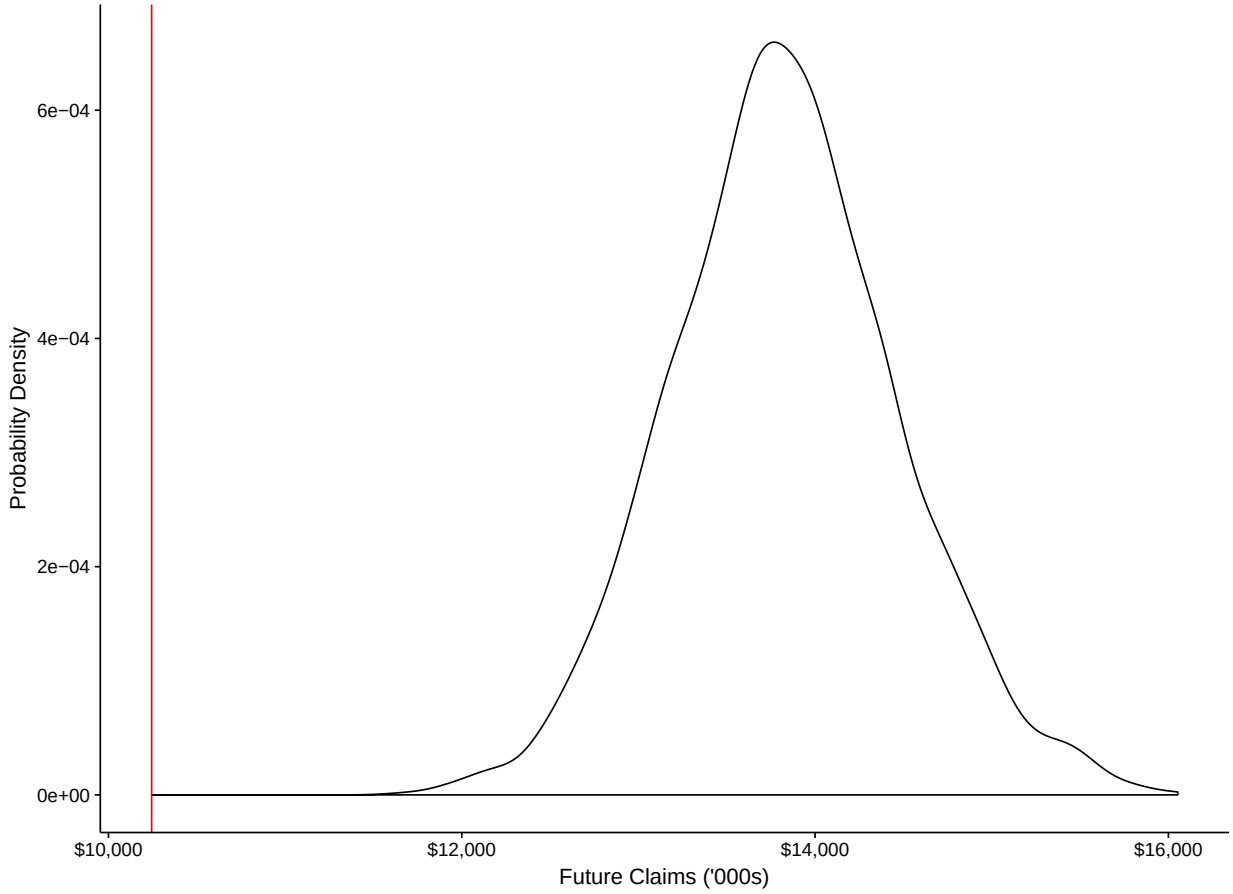


Posterior Predictive Checks

As before, we run some posterior predictive checks to see if we see similar patterns as before.

```
tckc <- grcode353_snapshot_tbl %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum

afc <- grcode353_tbl %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum

future_claims <- afc - tckc


ggplot() +
    geom_density(aes(x = extract(model_sislob_grcode353_wb_stanfit)$ppc_EFC)) +
    geom_vline(aes(xintercept = future_claims), colour = 'red') +
    scale_x_continuous(labels = dollar) +
    xlab("Future Claims ('000s)") +
```

```
    ylab("Probability Density")
```



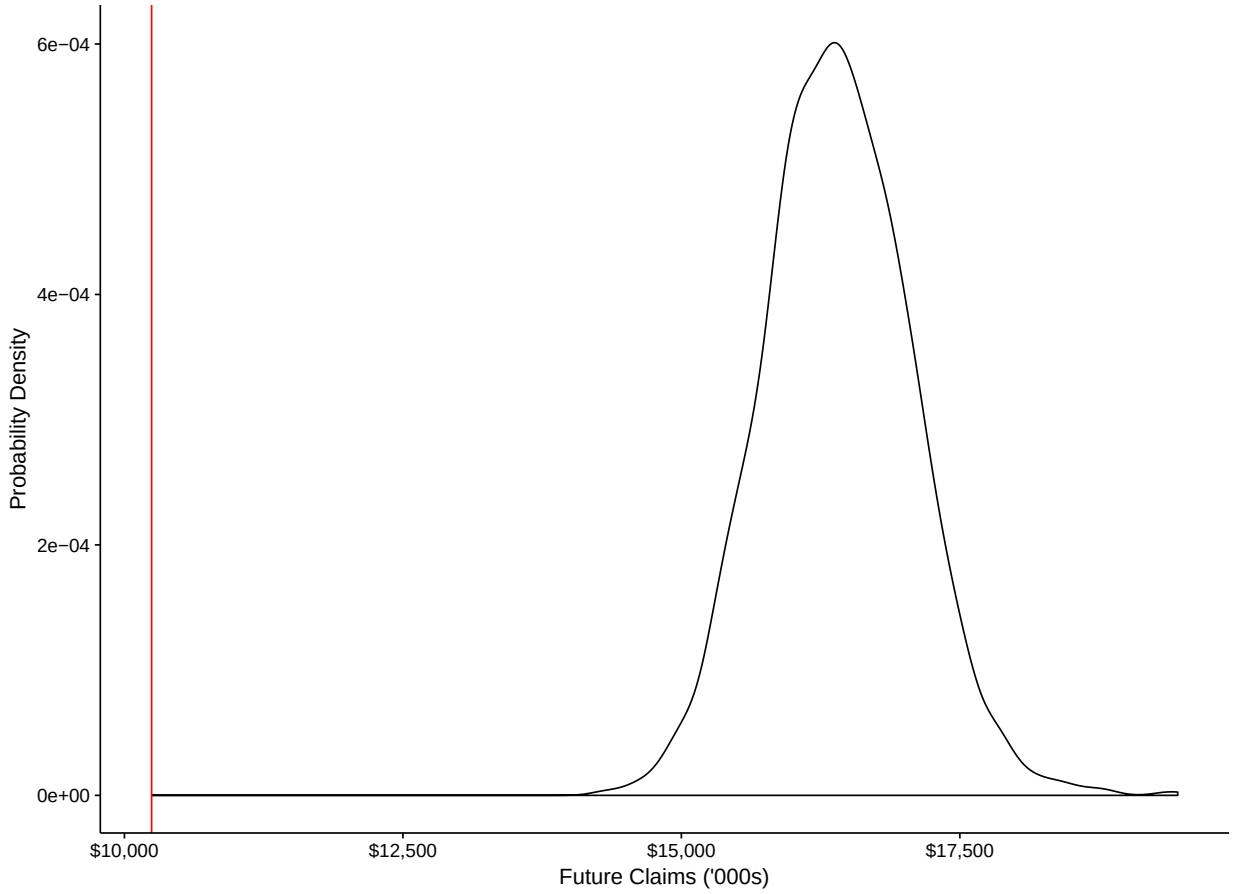
Fit Loglogistic Stan Model on New Data

We fit the model using the GRCODE 353 data and the loglogistic growth functional form.

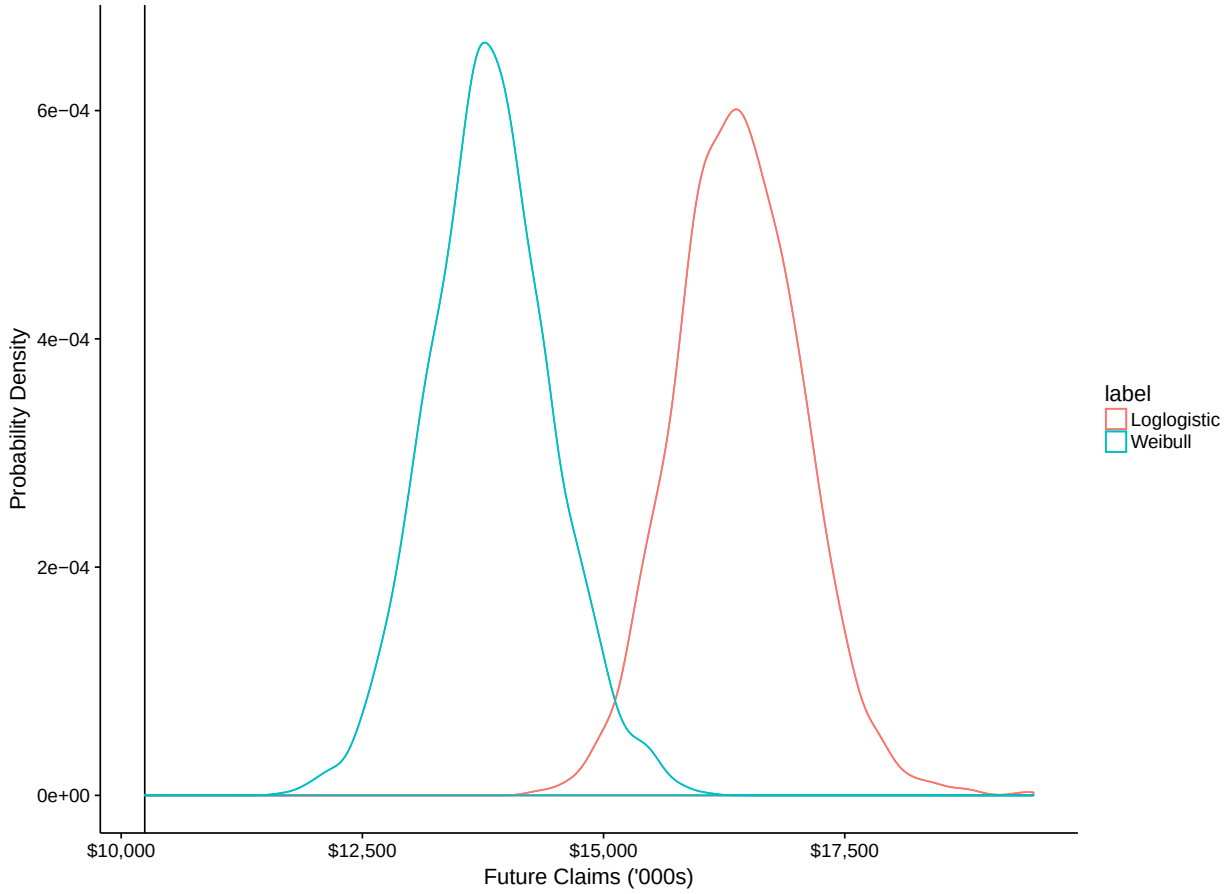
```
model_sislob_grcode353_ll_list <- create_stanfit(model_sislob_stanmodel
                                                 ,grcode353_snapshot_tbl
                                                 ,model_id = 0
                                                 ,stan_seed = stan_seed)

model_sislob_grcode353_ll_stanfit <- model_sislob_grcode353_ll_list$stanfit
```

```
tckc <- grcode353_snapshot_tbl %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum

afc <- grcode353_tbl %>%
    group_by(acc_year) %>%
    filter(dev_lag == max(dev_lag)) %>%
    .[["cum_loss"]] %>%
    sum
```

```
future_claims <- afc - tckc


ggplot() +
    geom_density(aes(x = extract(model_sislob_grcode353_ll_stanfit)$ppc_EFC)) +
    geom_vline(aes(xintercept = future_claims), colour = 'red') +
    scale_x_continuous(labels = dollar) +
    xlab("Future Claims ('000s)") +
    ylab("Probability Density")
```




```
plot_tbl <- bind_rows(
    data_frame(label  = 'Weibull'
               ,values = extract(model_sislob_grcode353_wb_stanfit)$ppc_EFC)
   ,data_frame(label = 'Loglogistic'
               ,values = extract(model_sislob_grcode353_ll_stanfit)$ppc_EFC)
)

ggplot(plot_tbl) +
    geom_vline(aes(xintercept = future_claims)) +
    geom_density(aes(x = values, colour = label)) +
    geom_density(aes(x = values, colour = label)) +
    scale_x_continuous(labels = dollar) +
    xlab("Future Claims ('000s)") +
    ylab("Probability Density")
```



References

- Morris, J. (2016) Hierarchical Compartmental Models for Loss Reserving *Casualty Actuarial Society E-Forum, Summer 2016* pdf
- Guszcza, J. C. (2008) Hierarchical Growth Curve Models for Loss Reserving *CAS Forum 2008: Fall*, pdf