Robust Loss Development Using MCMC: A Vignette

Christopher W. Laws Frank A. Schmid

November 9, 2009

Abstract

This vignette describes and demonstrates the use of the package `lossDev`, which offers a Bayesian time series model for developing aggregate loss triangles in property casualty insurance. This model distinguishes itself from other time series loss development models by its heavy-tailed, skewed, and heteroskedastic Student-t likelihood, its use of Reversible Jump MCMC for estimating the trajectory of the consumption path, and its ability to accommodate a structural break in this consumption path. Further, the model is capable of incorporating expert information in the calendar year effect. The use of the model is exemplified on two widely studied General Liability and Auto Bodily Injury Liability loss triangles. The package is available for download from `http://lossdev.r-forge.r-project.org/`.

1 Installation

At the time of writing this vignette, the current version of `lossDev` is 0.7.0. `lossDev` is released as an R package and can be downloaded from `http://lossdev.r-forge.r-project.org/`. Ultimately, `lossDev` will be available on CRAN. (For instructions on installing R packages please see the help files for R.) `lossDev` requires `rjags` for installation. `rjags` requires that a valid version of JAGS be installed on the system. JAGS is an open source platform for analysis of Bayesian hierarchical models using Markov Chain Monte Carlo (MCMC) simulation. JAGS can be freely downloaded from `http://calvin.iarc.fr/~martyn/software/jags/`.

2 Model Overview

The `lossDev` model splits the data-generating process of the loss triangle into three time dimensions, which are exposure time, the calendar year, and the development time, thus resulting in a time series model with three dimensions. Specifically, the incremental payments are driven by three time processes, which manifest themselves in exposure growth, the calendar year effect, and decay in consumption; these processes are illustrated in Figure 1.

In the model, the growth rate that represents the calendar year effect is denoted κ. The rate of exposure growth, η, is net of the calendar year effect. The growth rate δ is the rate of decay in incremental payments, adjusted for the calendar year effect. Incremental payments that have been adjusted for the calendar year effect (and, hence, inflation) represent consumption of units of services; for instance, for an auto bodily injury triangle, this consumption pertains to medical services. A decline in consumption at the level of the aggregate loss triangle may be due to claimants exiting or remaining claimants decreasing their consumption.

For a more detailed explanation, including model equations, please see
Schmid, Frank A. "Robust Loss Development Using MCMC," 2009.

`lossDev` currently provides two models. Both of which are designed to develop losses for annual insurance data.

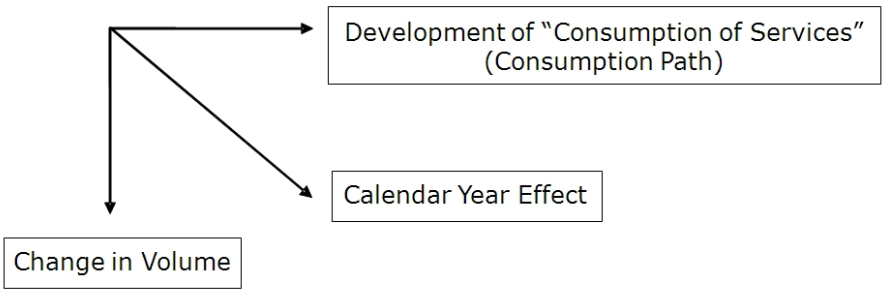

Figure 1: Triangle Dynamics.

Section 3 uses the first model, which assumes that all exposure years are subject to a common consumption path, to develop an example loss triangle.

Section 4 uses the second model to develop another loss triangle. The second model allows for a structural break in the consumption path and assumes earlier exposure years are subject to one consumption path and later exposure years are subject to another.

3 Using the Standard Model for Estimation

3.1 Data

For a demonstration of the standard ("no break") model an example loss triangle is taken from Automatic Facilitative business in General Liability (excluding Asbestos & Environmental). The payments are on an incurred basis.

This triangle is from
Mack, Thomas, "Which Stochastic Model is Underlying the Chain Ladder Method," *Casualty Actuarial Society Forum*, Fall 1995, pp. 229-240, `http://www.casact.org/pubs/forum/95fforum/95ff229.pdf`.

3.2 Model Specification

Standard models are specified with the function `makeStandardAnnualInput`. This function takes as input all data used in the estimation process. `makeStandardAnnualInput` also allows the user to vary the model specification through several arguments. Most of these arguments have defaults which should be suitable for most purposes.

To ensure portability, the data used in this vignette is packaged in `lossDev` and as such is loaded using the `data` function. However, the user wishing to develop other loss triangles should load the data using standard R functions (such as `read.table` or `read.csv`). See the R manual for assistance.

3.2.1 Loading and Manipulating the Data

The Triangle As input, `makeStandardAnnualInput` can take either a cumulative loss triangle or an incremental loss triangle (or in the case where one might not be directly calculable from the other, both triangles may be supplied). `makeStandardAnnualInput` expects any supplied loss triangle to be a matrix. The row names for the matrix must be the Accident (or Policy) Year

and must appear in ascending order. (Note that if the triangle is a Policy Year Triangle, the first column is assumed to be a *half* report.) The matrix must be square and *all* values below the diagonal must be missing. Missing values on and above the diagonal are permitted.

Note the negative value for this example in Accident Year 1982. Since incremental payments are modeled on the log scale, this value will be treated as missing and could result in a slightly overstated ultimate loss. A comparison of predicted vs observed cumulative payments in Figure 9 indicates that, at least in this instance, this has a minimal effect.

```
> library(lossDev)

loading JAGS module
   basemod
   bugs
loading JAGS module
   lossDev

> data(IncrementalGeneralLiablityTriangle)
> IncrementalGeneralLiablityTriangle <- as.matrix(IncrementalGeneralLiablityTriangle)
> print(IncrementalGeneralLiablityTriangle)

     DevYear1 DevYear2 DevYear3 DevYear4 DevYear5 DevYear6 DevYear7 DevYear8
1981     5012     3257     2638      898     1734     2642     1828      599
1982      106     4179     1111     5270     3116     1817     -103      673
1983     3410     5582     4881     2268     2594     3479      649      603
1984     5655     5900     4211     5500     2159     2658      984       NA
1985     1092     8473     6271     6333     3786      225       NA       NA
1986     1513     4932     5257     1233     2917       NA       NA       NA
1987      557     3463     6926     1368       NA       NA       NA       NA
1988     1351     5596     6165       NA       NA       NA       NA       NA
1989     3133     2262       NA       NA       NA       NA       NA       NA
1990     2063       NA       NA       NA       NA       NA       NA       NA
     DevYear9 DevYear10
1981       54       172
1982      535        NA
1983       NA        NA
1984       NA        NA
1985       NA        NA
1986       NA        NA
1987       NA        NA
1988       NA        NA
1989       NA        NA
1990       NA        NA
```

The Stochastic Inflation Expert Prior It is possible that incremental payments are subject to price increases due to external forces such as inflation. `makeStandardAnnualInput` can take such forces into account by suppling a rate of inflation from a price index, such as the CPI, to be used as an expert prior. The supplied rate of inflation must cover the years of the supplied incremental triangle and can extend (both into the past and future) beyond these years. If a future year's rate of inflation is needed and is unobserved, it will be simulated from an Ornstein–Uhlenbeck process calibrated to the observed series.

For this example, the CPI is taken as an expert prior for the stochastic rate of inflation.

Note that rates of inflation beyond the final observed diagonal in `IncrementalGeneralLiablity-Triangle` are excluded from this example. While `lossDev` is capable of utilizing such information

beyond the last observed diagonal, to make the example more realistic, the series has been truncated.

```
> data(CPI)
> CPI <- as.matrix(CPI)[, 1]
> CPI.rate <- CPI[-1]/CPI[-length(CPI)] - 1
> CPI.rate.length <- length(CPI.rate)
> print(CPI.rate[(-10):0 + CPI.rate.length])

      1997       1998       1999       2000       2001       2002       2003
0.02294455 0.01557632 0.02208589 0.03361345 0.02845528 0.01581028 0.02279044
      2004       2005       2006       2007
0.02663043 0.03388036 0.03225806 0.02848214

> CPI.years <- as.integer(names(CPI.rate))
> years.available <- CPI.years <= max(as.integer(dimnames(IncrementalGeneralLiablityTriangle)[[1]
> CPI.rate <- CPI.rate[years.available]
> CPI.rate.length <- length(CPI.rate)
> print(CPI.rate[(-10):0 + CPI.rate.length])

      1980       1981       1982       1983       1984       1985       1986
0.13498623 0.10315534 0.06160616 0.03212435 0.04317269 0.03561116 0.01858736
      1987       1988       1989       1990
0.03649635 0.04137324 0.04818259 0.05403226
```

3.2.2 Selection of Model Options

The function `makeStandardAnnualInput` has many options to allow for customization of model specification; however, in an aim to avoid overwhelming the reader, not all options are illustrated in this tutorial.

For this example, the loss history is supplied as incremental payments to the argument `incremental.payments`. The exposure year type of this triangle is also set to Accident Year by setting the value of `exp.year.type` to "ay." The default is "ambiguous" which should be sufficient in most cases as this information is only utilized by a handful of functions and the information can be supplied (or overridden) at the execution time of those functions.

The function allows for the specification of two rates of inflation (plus a zero rate of inflation). One of these rates is allowed to be stochastic, meaning that future rates of this inflation series are simulated from a process calibrated to the observed series, thus incorporating uncertainty. For the current demonstration, it will be assumed that the CPI is the only applicable inflation rate. As only historical values for the CPI are known, the CPI is assumed to be a stochastic rate of inflation. This is done by setting the value of `stoch.inflation.rate` to `CPI.rate` (which was created earlier). The user has the option of specifying what percentage (with this value being allowed to vary for each cell) of dollars inflate at `stoch.inflation.rate`. For the current illustration, it is assumed that all dollars (in all periods) follow the CPI. This is done by setting `stoch.inflation.weight` to 1 and `non.stoch.inflation.weight` to 0.

By default, the measurement equation for the logarithm of the incremental payments is a Student-t. The user has the option of using a skewed-t by setting the value of `use.skew.t` to TRUE. For this demonstration, a skewed-t will be used.

Because `lossDev` is designed to develop loss triangles to ultimate, some assumptions must be made with regard to the extension of the consumption path beyond the number of development years comprised in the observed triangle. The default assumes the final estimated decay rate is applicable for all future development years and is assumed for this example. This default can be overridden by the argument `projected.rate.of.decay`. Additionally, either the final number of (possibly)

non-zero payments must be supplied via the argument `total.dev.years` or the number of non-zero payments in addition to the number of development years in the observed triangle must be supplied via the argument `extra.dev.years`. Similarly, the number additional of exposure years can also be specified.

```
> standard.model.input <- makeStandardAnnualInput(incremental.payments = IncrementalGeneralLiabli
+     stoch.inflation.weight = 1, non.stoch.inflation.weight = 0,
+     stoch.inflation.rate = CPI.rate, exp.year.type = "ay", extra.dev.years = 5,
+     use.skew.t = TRUE)
```

3.3 Estimating the Model

Once the model has been specified, it must be estimated.

MCMC Overview As the model is Bayesian, the model is estimated by means of Markov chain Monte Carlo Simulation (MCMC). To perform MCMC, a Markov chain is constructed in such a way that the limiting distribution of the chain is the posterior distribution of interest. The chain is then initialized with starting values and run until it has reached a point of convergence in which samples adequately represent random (albeit sequentially dependent) draws from this posterior distribution. The set of iterations performed (and discarded) until samples are assumed to be draws from the posterior is called a "burn-in." Once the chain has converged, the chain is iterated further to collect samples. The samples are then used to calculated the statistic of interest.

While the user is not responsible for the construction of the Markov chain, he is responsible for assessing the convergence of the chains. (Section 3.4.1 gives some pointers on this.) The most common way of accomplishing this task is to run several chains simultaneously, with each chain started with a different set of initial values. Once all the chains are producing similar results, one can assume the chains have converged.

To estimate the model, the function `runLossDevModel` is called with the first argument being the input object created by `makeStandardAnnualInput`. To specify the number of iterations to discard, the user sets the value of `burnIn`. To specify the number of iterations to perform after the burn-in, set the value of `sampleSize`. To set the number of chains to run simultaneously, supply a value for `nChains`. The default value for `nChains` is 3 and should be sufficient for most cases.

It is also common practice (due to the autocorrelation in MCMC samples) to "thin" the samples. This simply means that only every n-th draw is stored. The argument `thin` is available for this purpose.

Memory Issues MCMC can require large amounts of memory. To allow `lossDev` to work with limited hardware, the R package `filehash` is used to cache the codas of monitored values to the hard-drive in an efficient way. While such caching can allow estimation of large triangles on computers with less memory, it can also slow down some computations. As such, the user has the option of turning this feature on and off. This is done via the function `lossDevOptions` by setting the argument `keepCodaOnDisk` to FALSE.

R also makes available the function `memory.limit` which one may find useful.

```
> standard.model.output <- runLossDevModel(standard.model.input,
+     burnIn = 30000, sampleSize = 40000, thin = 10)

Compiling data graph
   Resolving undeclared variables
   Allocating nodes
   Initializing
   Reading data back into data table
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 3952

[1] "Update took 25.6075500011444"
```

3.4 Examining Output

`makeStandardAnnualInput` returns a very complex output object. `lossDev` provides many user-level functions to access the information contained in this object. Many of these functions are described below.

3.4.1 Assessing Convergence

As previously mentioned, the user is responsible for assessment of the convergence of the Markov chains used to estimate the model. To this aim, `lossDev` provides several functions to produce trace and density plots.

Arguably, the most important charts to examine for convergence are the trace plots associated with the three time dimensions of the model. Convergence of exposure growth, the consumption path, and the calendar year effect are assessed in Figures 2, 3, and 4 respectively. These charts are produced with the functions `exposureGrowthTracePlot`, `consumptionPathTracePlot`, and `calendarYearEffectErrorTracePlot`.

```
> exposureGrowthTracePlot(standard.model.output)
```

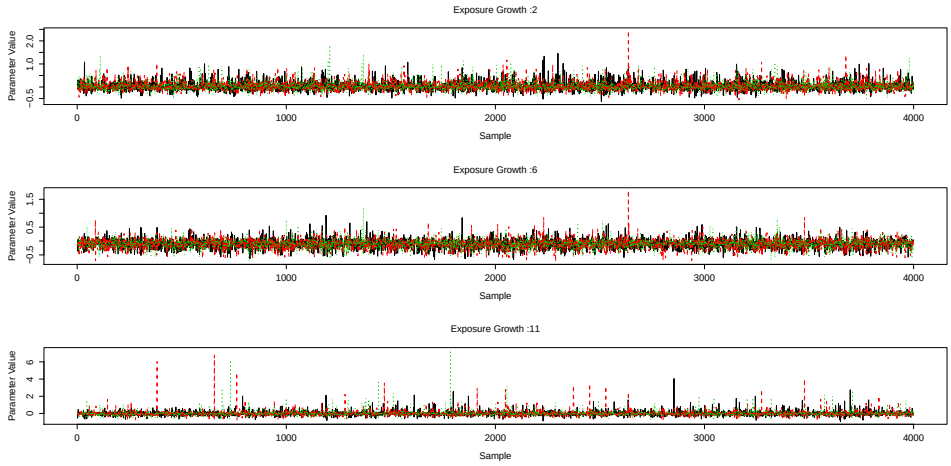

Figure 2: Trace plots for select exposure growth parameters.

3.4.2 Assessing Model Fit

`lossDev` provides many diagnostic charts to asses how well the model fits the observed triangle.

Residuals For analysis of residuals, `lossDev` provides the function `triResi`. `triResi` plots the residuals (on the log scale) by the three time dimensions. The time dimension is selected by means of the argument `timeAxis`. By default, residual charts are standardized to account for any assumed/estimated heteroskedasticity in the payments. These charts can be found in Figures 5, 6, and 7.

```
> consumptionPathTracePlot(standard.model.output)
```

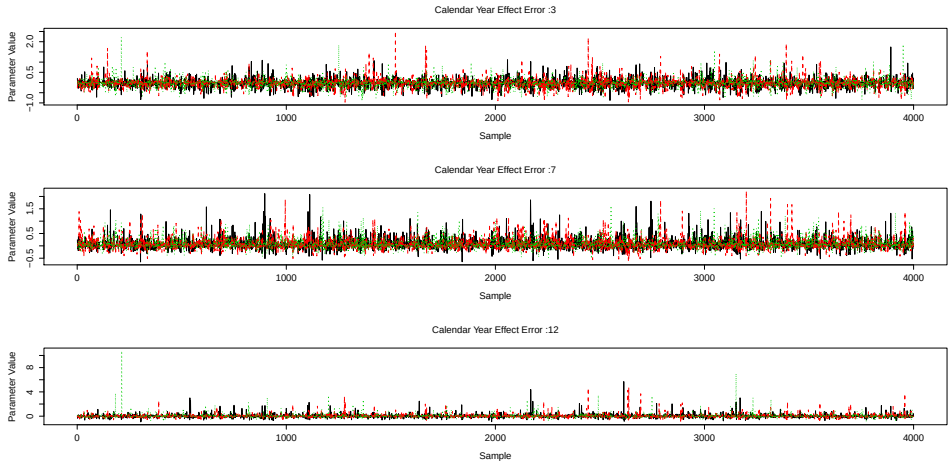

Figure 3: Trace plots for select development years on the consumption path.

```
> calendarYearEffectErrorTracePlot(standard.model.output)
```

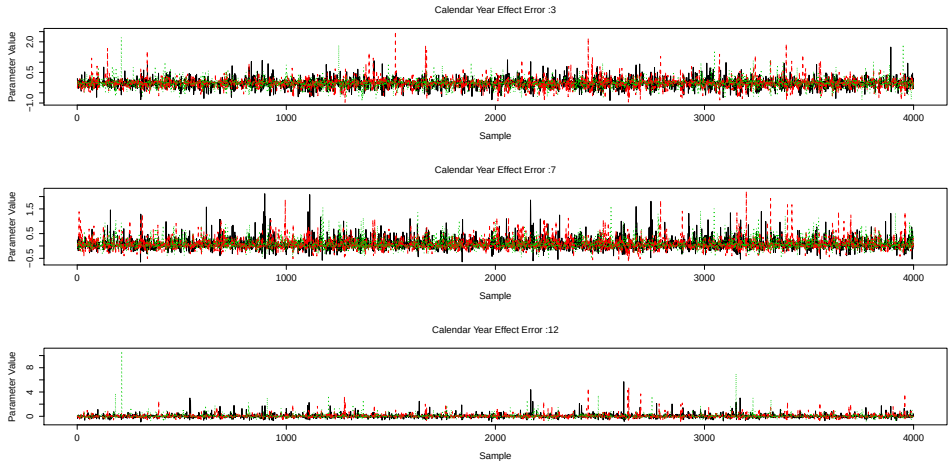

Figure 4: Trace plots for select calendar year effect errors.

Note that because log incremental payments are allowed to be skewed, the residuals need not be symmetric.

```
> triResi(standard.model.output, timeAxis = "dy")
```

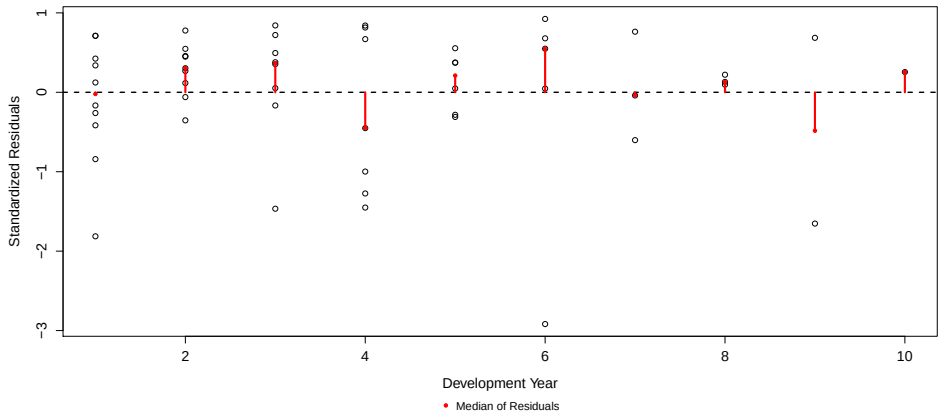

Figure 5: Residuals by development year.

```
> triResi(standard.model.output, timeAxis = "ey")
```

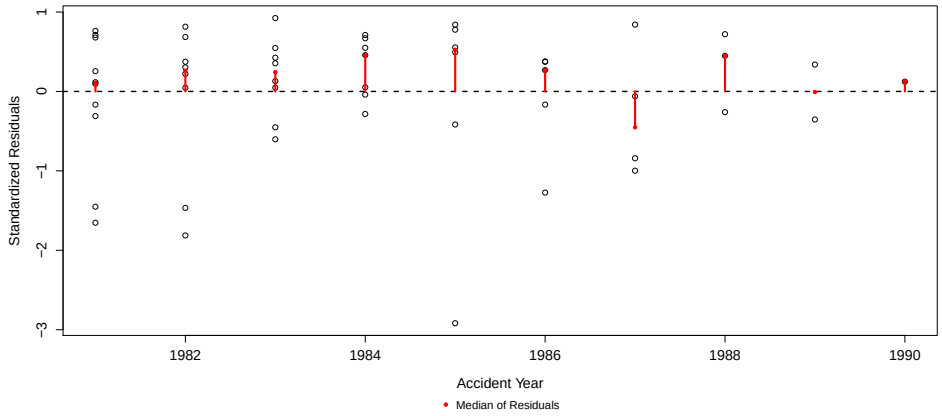

Figure 6: Residuals by exposure year.

QQ-Plot `lossDev` provides a QQ-Plot in the function `QQPlot`. `QQPlot` plots the median of simulated incremental payments (sorted at each iteration) against the observed incremental payments. Plotted points from a well calibrated model are close to the 45-degree line. These results are shown in Figure 8.

Comparison of Cumulative Payments As a means of assessing how well the predicted cumulative payments line up with the observed values, `lossDev` provides the function `finalCumulativeDiff`. This function will plot the relative difference between the predicted and observed cumulative payments (when such payments exists) for the last observed cumulative payment in

```
> triResi(standard.model.output, timeAxis = "cy")
```

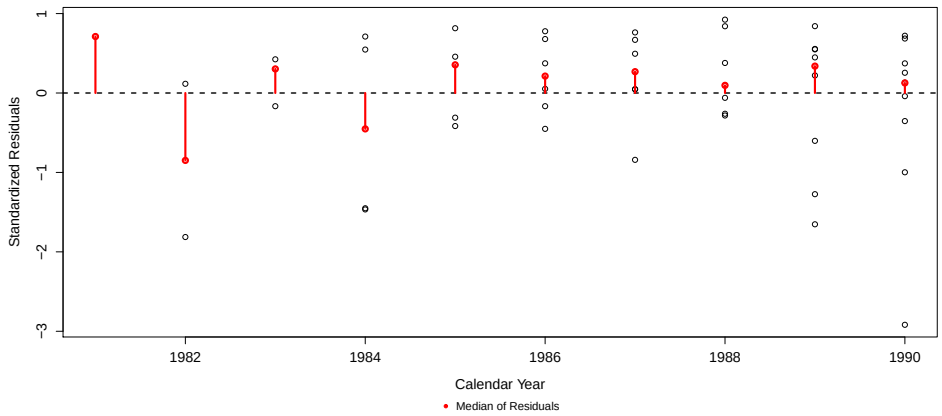

Figure 7: Residuals by calendar year.

```
> QQPlot(standard.model.output)
```

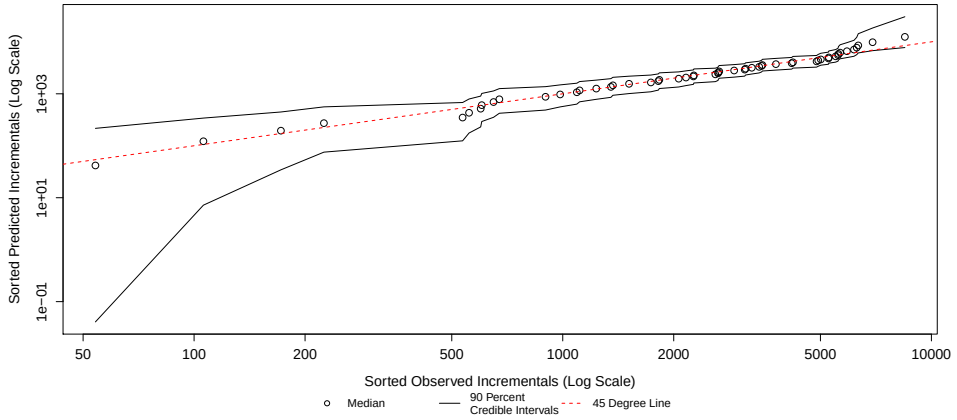

Figure 8: QQ-Plot.

each exposure year. Credible intervals are also plotted. These results are shown in Figure 9 and can be useful for assessing the impact of negative incremental payments as previously stated.

```
> finalCumulativeDiff(standard.model.output)
```

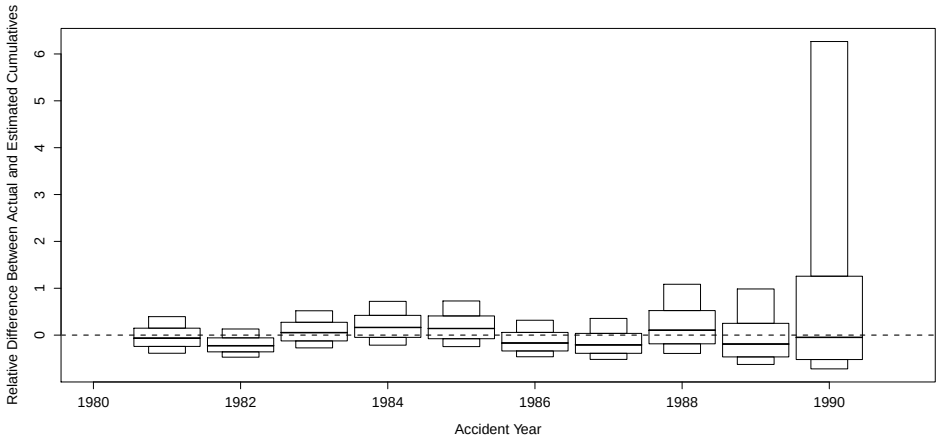

Figure 9: Difference in Final Observed Cumulative Payments.

3.4.3 Extracting Inference and Results

Once the user has deemed the Markov chains converged and once the user has determined that the model has (at least reasonably) captured the underly process observed in the triangle, he will wish to extract results from the output. Many of the functions mentioned in this section also return the values of some plotted information. These values are returned invisibly and as such are not printed in R unless such an operation is requested. Additionally, many of these functions also provide an option to suppress plotting.

Predicted Payments Perhaps the most practically useful function is `predictedPayments`. This function can plot and return the estimated incremental predicated payments. As the function can also plot the observed values against the predicted values (`plotObservedValues`), it also has a diagnostic function. The log incremental payments are plotted against the predicted values in Figure 10.

`predictedPayments` can also plot and return the estimated cumulative payments and has the option of taking observed payments at "face value" (meaning that predicted payments are replaced with observed payments whenever possible) in the returned calculations; this can be useful for construction of reserve estimates. In Figure 11, only the predicted cumulative payments are plotted. The function is also used to construct an estimate (with credible intervals) of the ultimate loss.

```
> standard.ult <- predictedPayments(standard.model.output, type = "cumulative",
+     plotObservedValues = FALSE, mergePredictedWithObserved = TRUE,
+     logScale = TRUE, quantiles = c(0.025, 0.5, 0.0975), plot = FALSE)
> standard.ult <- standard.ult[, , dim(standard.ult)[3]]
> print(standard.ult)

          1981     1982     1983     1984     1985     1986     1987     1988
2.5%  18862.28 16761.20 23620.25 27484.90 27040.92 17310.41 14602.39 16499.59
50%   19083.63 17175.02 24324.57 28724.58 29255.95 20681.08 19859.84 24368.60
```

```
> predictedPayments(standard.model.output, type = "incremental",
+     logScale = TRUE)
```

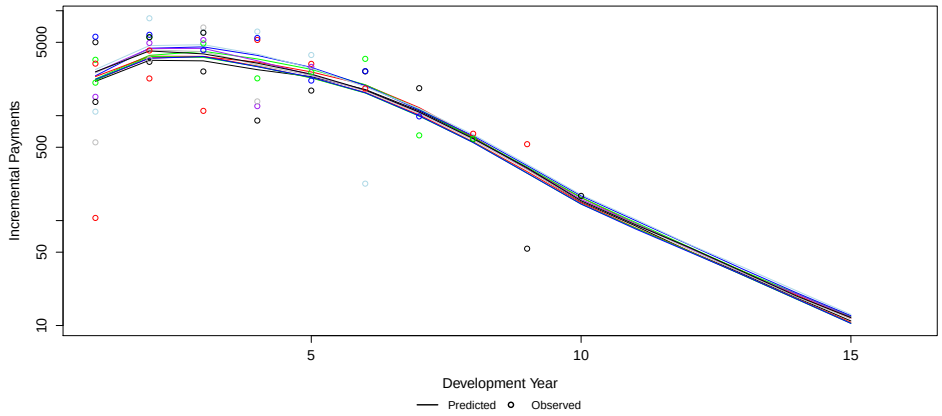

Figure 10: Predicted Incremental Payments.

```
9.75% 18903.72 16854.73 23794.63 27795.92 27591.06 18199.91 15998.86 18654.21
          1989      1990      1991
2.5%   9597.32  7045.024  4684.006
50%   20273.17 21268.737 22910.038
9.75% 12446.19 10319.657  8488.105
```

```
> predictedPayments(standard.model.output, type = "cumulative",
+     plotObservedValues = FALSE, logScale = TRUE)
```

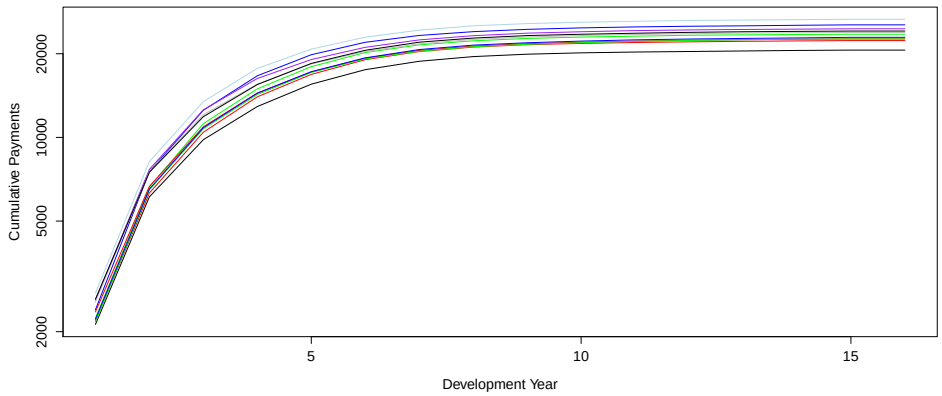

Figure 11: Predicted Cumulative Payments.

Consumption Path `lossDev` makes the consumption path available via `consumptionPath`. The consumption path is the trajectory of exposure-adjusted and calendar year effect adjusted log incremental payments and is modeled as a linear spline. The standard model assumes a common consumption path for all exposure years in the triangle. The use of this function is demonstrated in Figure 12.

```
> consumptionPath(standard.model.output)
```

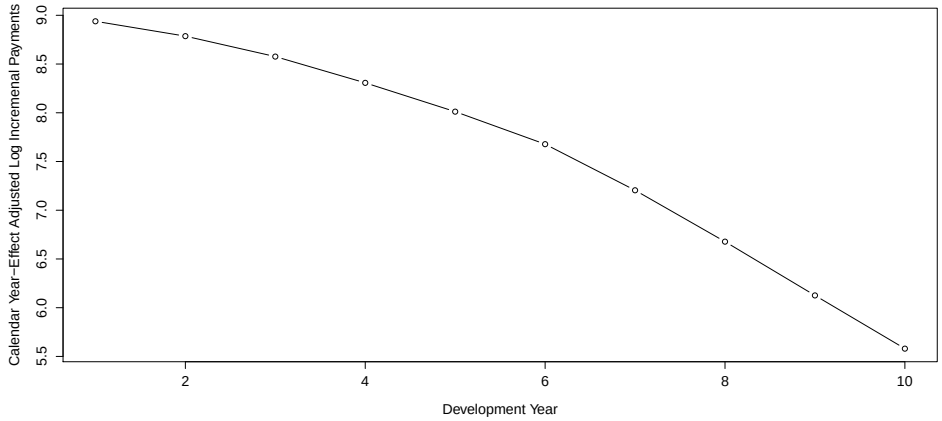

Figure 12: Consumption Path.

Knots in the Consumption Path As previously mentioned, the consumption path is modeled as a linear spline. The number of knots in this spline is endogenous to the model. The function `numberOfKnots` can be used to extract information regarding the posterior number of knots. All else equal, a higher number of knots indicates a more flexible functional form. Figure 13 illustrates the use of this function.

```
> numberOfKnots(standard.model.output)
```

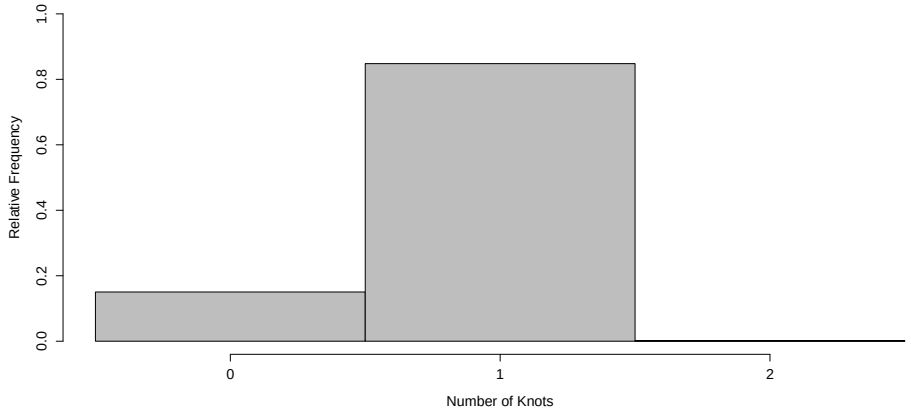

Figure 13: Number of Knots.

Rate of Decay While the consumption path illustrates the level of exposure-adjusted and calendar year effect adjusted log incremental payments, sometimes one may prefer to examine the consumption path in terms of its decay rate. The rate of decay from one development year to the next (which is approximately the slope of the consumption path) is made available via the function `rateOfDecay`. Because the standard model assumes a common consumption path for all exposure years, the standard model also has a single decay rate. An example of this function can be found in Figure 14.

```
> rateOfDecay(standard.model.output)
```

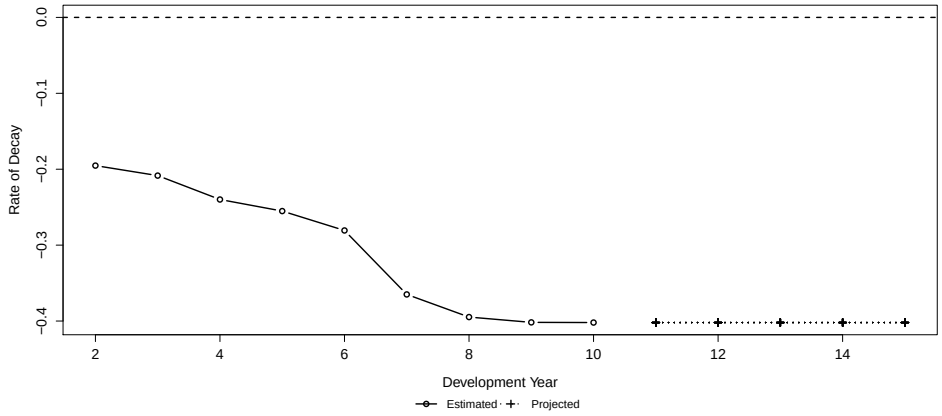

Figure 14: Rate Of Decay.

Exposure Growth The consumption path shown in Figure 12 is exposure-adjusted to the first exposure year. This path is renormalized to each exposure year so as to reflect the level of exposure for that particular year. The year over year changes in the estimated exposure level are made available by the function `exposureGrowth`. An example of this function can be found in Figure 15.

```
> exposureGrowth(standard.model.output)
```

Figure 15: Exposure Growth.

Calendar Year Effect The model assumes that every cell on a diagonal receives a related shock. The shock consists of a component exogenous to the triangle (generally an inflationary index such as the CPI or MCPI) and an endogenous stochastic component. As `lossDev` theoretically allows the user to vary the expert prior for each cell, graphically displaying the entire calendar year effect requires three dimensions. This is done by plotting a grid of colored block and varying the intensity of each color according to the associated calendar year effect. An example of this can be found in Figure 16. Note that the value in cell (1,1) is undefined.

```
> calendarYearEffect(standard.model.output)
```

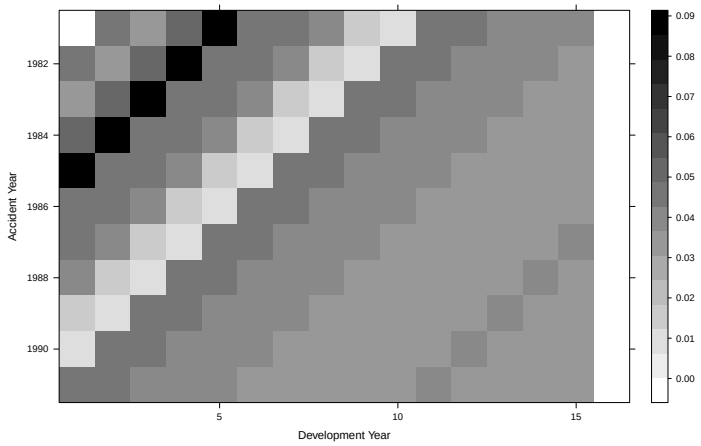

Figure 16: Calendar Year Effect.

Alternatively, one could merely plot the difference between the parameter and the expert prior. As this difference is common to all cells on a given diagonal, the number of dimensions is reduced by one. An illustration of this isolated component is displayed in Figure 17. This example displays some degree of autocorrelation. `lossDev` can account for such correlation by setting the argument `use.ar1.in.calendar.year` in `makeStandardAnnualInput` to TRUE. Exploring this is left as an exercise to the reader.

```
> calendarYearEffectErrors(standard.model.output)
```

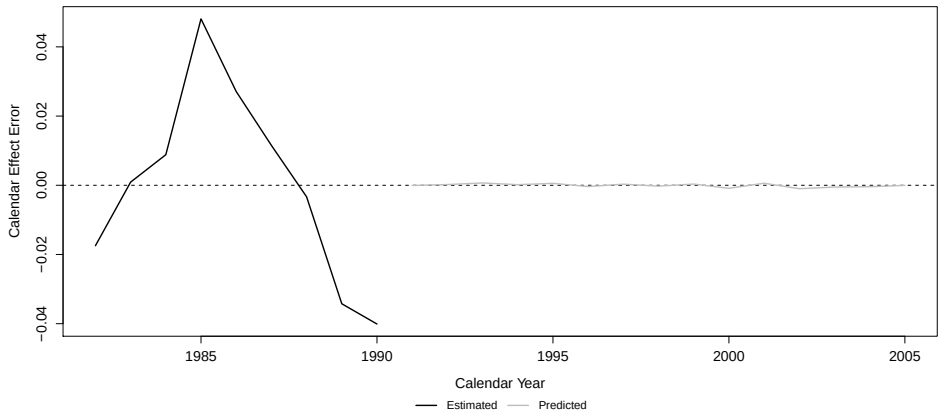

Figure 17: Calendar Year Effect Errors.

Changes In Variance As development time progresses, the number of claims that are associated with the incremental payments declines. This can lead not only to a decrease in the level of incremental payments, but also an increase in the variance associated with each cell. In order to account for this potential increase in variance, the model (optionally) allows for the scale parameter to vary with development time. This scale parameter is smoothed via a second-order random walk on the log scale. As a result, the standard deviation can vary for each development year. An

example is displayed in Figure 18.

```
> standardDeviationVsDevelopmentTime(standard.model.output)
```

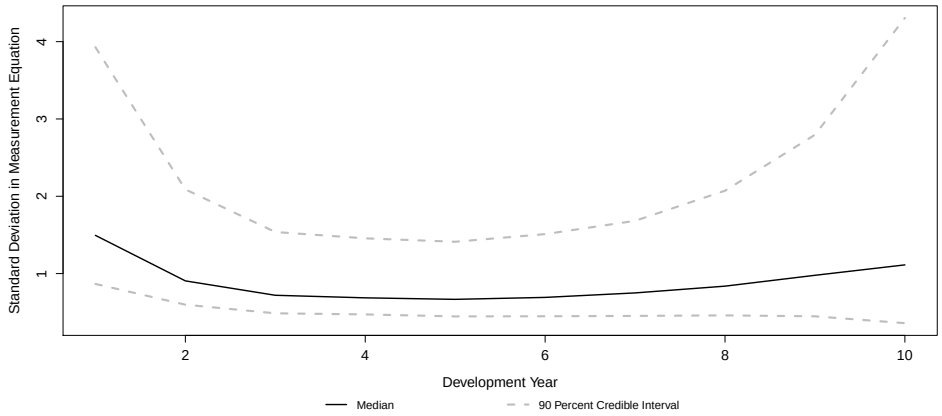

Figure 18: Standard Deviation vs Development Time.

Skewness Parameter As previously mentioned, the measurement equation for the log incremental payments is (optionally) a skewed-t. `skewnessParameter` allows for the illustration of the posterior skewness parameter. (For reference, the prior is also illustrated.) Although the skewness parameter does not directly translate to the estimated skewness, it is related. For instance, a skewness parameter of zero corresponds to zero skew. An example is displayed in Figure 19.

```
> skewnessParameter(standard.model.output)
```

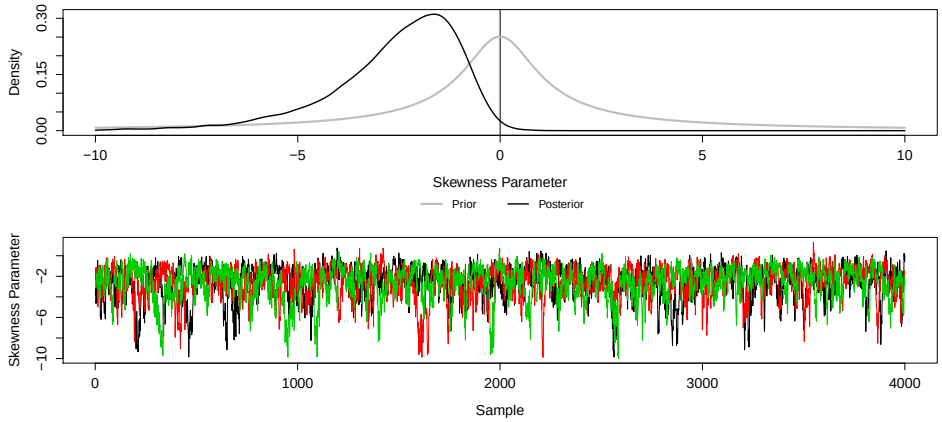

Figure 19: Skewness Parameter.

Degrees of Freedom The degrees of freedom associated with the measurement equation is endogenous to the model estimation. To ensure a finite variance, when estimating a skewed-t, the degrees of freedom is constrained to be greater than 4. Otherwise this value is constrained to be greater than 2. All else equal, lower degrees of freedom indicate the presence of fat tails.

The `lossDev` function `degreesOfFreedom` allows for the illustration of the posterior degrees of freedom. (For reference, the prior is also illustrated.) Figure 19 displays the posterior degrees of freedom for this example.

```
> degreesOfFreedom(standard.model.output)
```

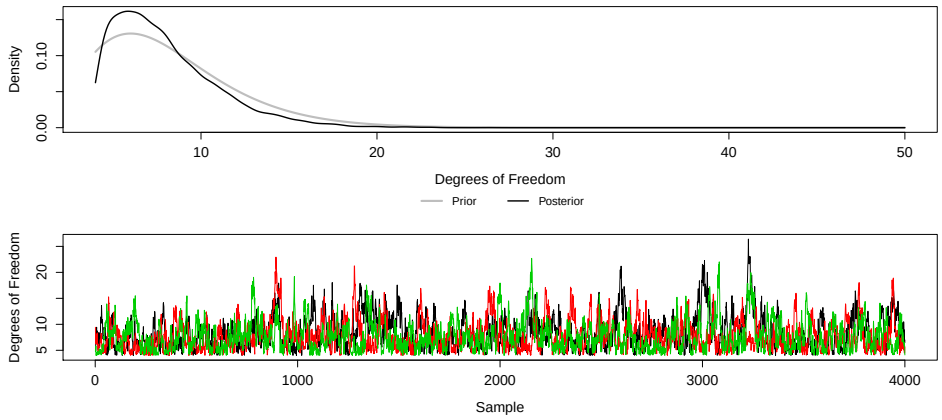

Figure 20: Degrees Of Freedom.

3.4.4 The Ornstein–Uhlenbeck Process

As previously mentioned, future values for the assumed stochastic rate of inflation are simulated from an Ornstein–Uhlenbeck process. `lossDev` allows the user to examine predicted and forecast values as well as some of the underlying parameters. Such options are outlined below.

Fit and Forecast To display the fitted values vs the observed values, as well as the forecast values, the user must use the function `stochasticInflation`. The chart for the example illustrated above is displayed in Figure 21.

```
> stochasticInflation(standard.model.output)
```

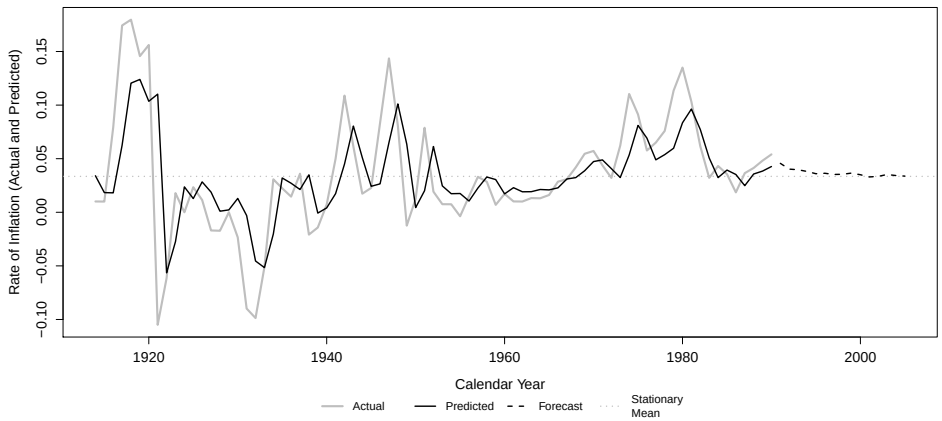

Figure 21: Stochastic Inflation Fit.

Stationary Mean The Ornstein–Uhlenbeck process has a stationary mean to which future simulated rates of inflation will return. In other words, the projected rate of inflation will (geometrically) approach a value as time progresses. This stationary mean can be graphed with the function `StochasticInflationStationaryMean`. The chart for the example illustrated above is displayed in Figure 22.

```
> stochasticInflationStationaryMean(standard.model.output)
```

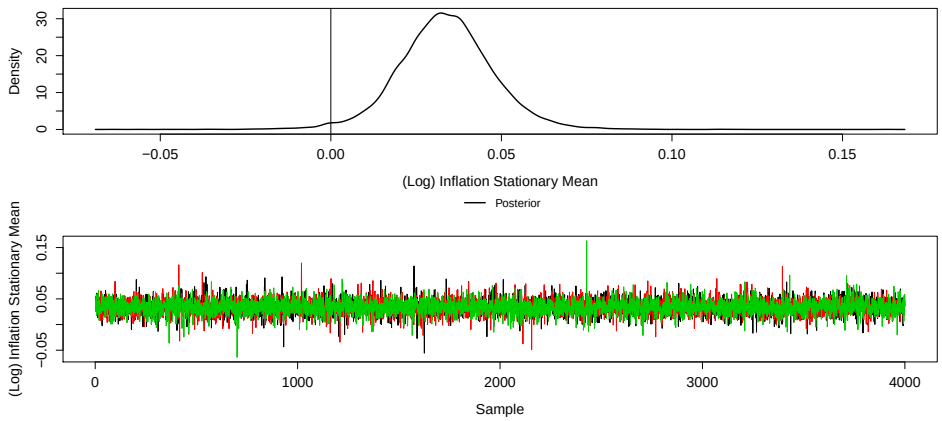

Figure 22: Estimated Stochastic Inflation Stationary Mean.

Persistence The Ornstein – Uhlenbeck process assumes that the influence of a disturbance decays geometrically with time. The parameter governing this rate is traditionally referred to as ρ. To obtain this value, call the function `StochasticInflationRhoParameter`. The chart for the example illustrated above is displayed in Figure 23.

```
> stochasticInflationRhoParameter(standard.model.output)
```

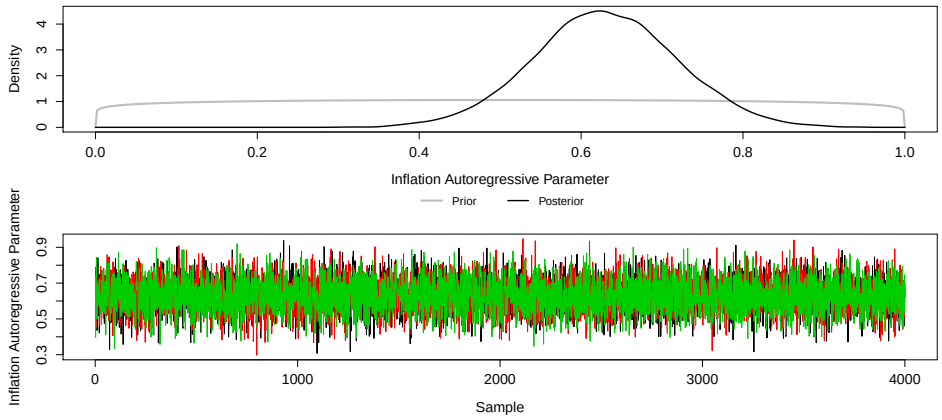

Figure 23: Estimated Stochastic Inflation Rho Parameter.

4 Using the Change Point Model for Estimation

The standard model outlined in Section 3 assumes the same consumption path for all exposure years. Due to changes in the loss environment, this may not be appropriate for all loss triangles. Such a triangle is outlined below.

4.1 Data

The triangle used for this example is a Private Passenger Auto Bodily Injury Liability triangle and consists of accident year data on a paid basis.

In December 1986, the ability of a judge to dismiss a case was limited by a judicial decision. As such, there is (at least the possibility of) a change in the consumption path at this point in time which makes this triangle a good example for the change point model.

This triangle is take from
Hayne, Roger M., "Measurement of Reserve Variability," *Casualty Actuarial Society Forum*, Fall 2003, pp. 141-172, `http://www.casact.org/pubs/forum/03fforum/03ff141.pdf`.

4.2 Model Specification

4.2.1 Loading and Manipulating the Data

The Triangle Section 3.2.1 supplied incremental payments as model input. For variety, here cumulative payments are supplied.

Note the large number of zero payments. Because the model will treat these as missing values (since they are equal to negative infinity on the log scale), the predicted payments will be overstated. This issue is addressed in Section 5.

```
> data(CumulativeAutoBodilyInjuryTriangle)
> CumulativeAutoBodilyInjuryTriangle <- as.matrix(CumulativeAutoBodilyInjuryTriangle)
> sample.col <- (dim(CumulativeAutoBodilyInjuryTriangle)[2] - 6:0)
> print(decumulate(CumulativeAutoBodilyInjuryTriangle)[1:7, sample.col])
```

```
     DevYear12 DevYear13 DevYear14 DevYear15 DevYear16 DevYear17 DevYear18
1974        20        25         0        12         0         0         0
1975        18        67         0         0         0        32        NA
1976        96         7         8        18         0        NA        NA
1977         2         0        50         0        NA        NA        NA
1978       -55        18        21        NA        NA        NA        NA
1979        19        26        NA        NA        NA        NA        NA
1980         5        NA        NA        NA        NA        NA        NA
```

The Stochastic Inflation Expert Prior The MCPI is chosen as the expert prior for the stochastic rate of inflation. While in Section 3.2.1 the stochastic inflation expert prior was truncated for realism, here a few extra observed years are kept for illustration purposes.

```
> data(MCPI)
> MCPI <- as.matrix(MCPI)[, 1]
> MCPI.rate <- MCPI[-1]/MCPI[-length(MCPI)] - 1
> print(MCPI.rate[(-10):0 + length(MCPI.rate)])
```

```
      1997       1998       1999       2000       2001       2002       2003
0.02804557 0.03196931 0.03510946 0.04070231 0.04601227 0.04692082 0.04026611
```

```
      2004       2005       2006       2007
0.04375631 0.04224444 0.04022277 0.04418203
```

```
> MCPI.years <- as.integer(names(MCPI.rate))
> max.exp.year <- max(as.integer(dimnames(CumulativeAutoBodilyInjuryTriangle)[[1]]))
> years.to.keep <- MCPI.years <= max.exp.year + 3
> MCPI.rate <- MCPI.rate[years.to.keep]
```

4.2.2 Selection of Model Options

While `makeStandardAnnualInput` (Section 3.2.2) is used to specify models without a change point, `makeBreakAnnualInput` is used to specify models with a change point. `makeBreakAnnualInput` has most of its arguments in common with `makeStandardAnnualInput`, and all these common arguments carry their meanings forward. However, `makeBreakAnnualInput` adds a few new arguments, which pertain to the location of the structural break.

Most notably is the argument `first.year.in.new.regime` which, as the name suggests, indicates the first year in which the new consumption path applies. This argument can be supplied with a single value, in which case the model will give a hundred percent probability to this year being the first year in the new regime. However, this argument can also be supplied with a range of values and the model will then estimate the first year in the new regime. Since the possible break occurs in late 1986, the range of years chosen for this example is 1986 to 1987.

The prior for the first year in the new regime is a discretized beta distribution. The user has the option of choosing the parameters for this prior by setting the argument `prior.for.first.year.in.new.regime`. Here, since the change was in late 1986, we choose a prior to give more probability to the latter year.

The argument `bound.for.skewness.parameter` is also set to 5. This avoids the MCMC chain from getting "stuck" for this particular example. One should use the function `skewnessParemter` (Figure 37) to evaluate the need to set this value. If the user is experiencing difficulties with the skewed-t, he may wish to use the non-skewed-t by ensuring that the argument `use.skew.t` is set to FALSE (which is the default).

```
> break.model.input <- makeBreakAnnualInput(cumulative.payments = CumulativeAutoBodilyInjuryTrian
+     stoch.inflation.weight = 1, non.stoch.inflation.weight = 0,
+     stoch.inflation.rate = MCPI.rate, first.year.in.new.regime = c(1986,
+         1987), priors.for.first.year.in.new.regime = c(2, 1),
+     exp.year.type = "ay", extra.dev.years = 5, use.skew.t = TRUE,
+     bound.for.skewness.parameter = 5)
```

4.3 Estimating the Model

Just like in Section 3.3, the S4 object returned by `makeBreakAnnualInput` must be supplied to the function `runLossDevModel` in order to produce estimates.

```
> break.model.output <- runLossDevModel(break.model.input, burnIn = 30000,
+     sampleSize = 40000, thin = 10)
```

```
Compiling data graph
   Resolving undeclared variables
   Allocating nodes
   Initializing
   Reading data back into data table
Compiling model graph
```

```
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 14499

[1] "Update took 2.31303277777301"
```

4.4 Examining Output

4.4.1 Assessing Convergence

As discussed previously, the user must again examine the MCMC runs for convergence using the same functions mentioned in Section 3.4.1. Here, to avoid repetition, only a few of the previously illustrated charts will be discussed below.

Because the change point model has two consumption paths, the method `consumptionPathTracePlot` for output related to this model has an additional argument to specify the consumption path. If the argument `preBreak` equals TRUE, then the trace for the consumption path relevant to exposure years prior to the structural break will be plotted. Otherwise, the trace for the consumption path relevant to exposure years after the break will be plotted.

The trace for the pre-break consumption path is plotted in Figure 24. The trace for the post-break path is plotted in Figure 25.

```
> consumptionPathTracePlot(break.model.output, preBreak = TRUE)
```

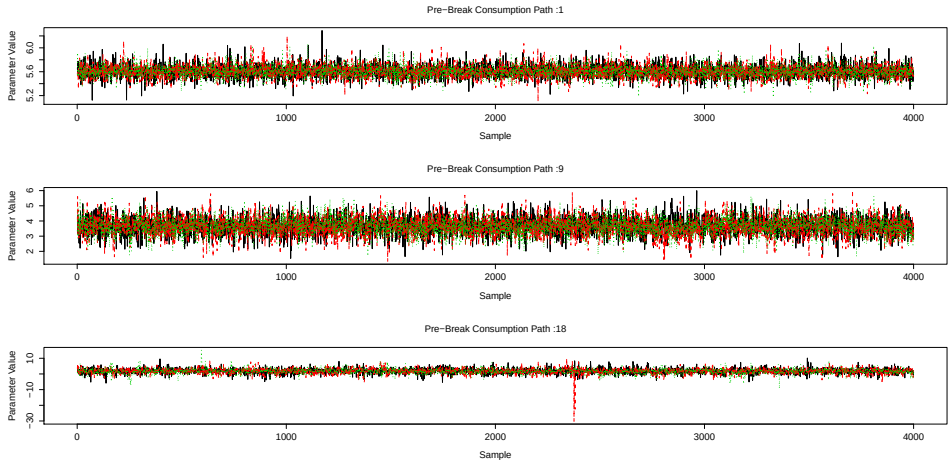

Figure 24: Trace plots for select development years on the pre-break consumption path.

4.4.2 Assessing Model Fit

All of the functions mentioned in Section 3.4.2 are available for the change point model as well.

Residuals One feature of `triResi` not mentioned in Section 3.4.2 is the option to turn off the standardization. The model accounts for an increase in the variance of incremental payments as development time progresses by allowing a scale parameter to vary with development time. By default, `triResi` also accounts for this by standardizing all the residuals to have a standard deviation of one. Turning off this feature (via the argument `standardize`) can lead insight into this process.

The standardized residuals for the change point model are displayed by development time in Figure 26. Figure 27 shows the residuals without this standardization.

```
> consumptionPathTracePlot(break.model.output, preBreak = FALSE)
```

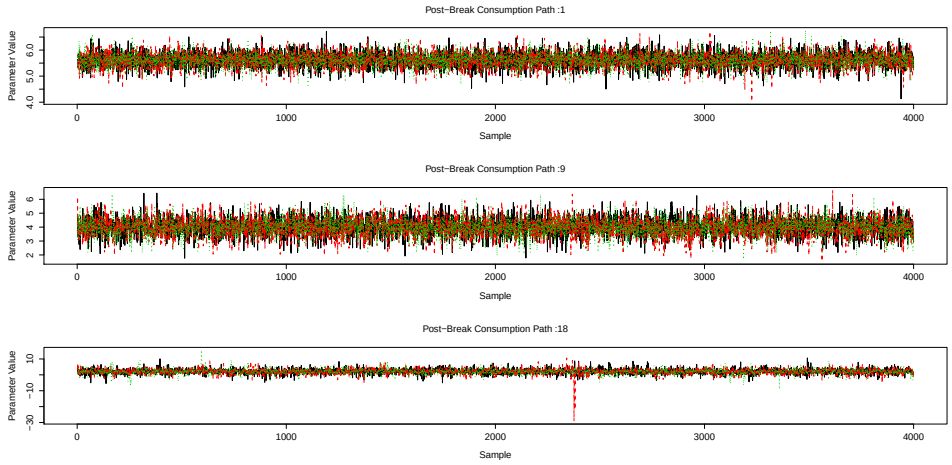

Figure 25: Trace plots for select development years on the post-break consumption path.

```
> triResi(break.model.output, timeAxis = "dy")
```

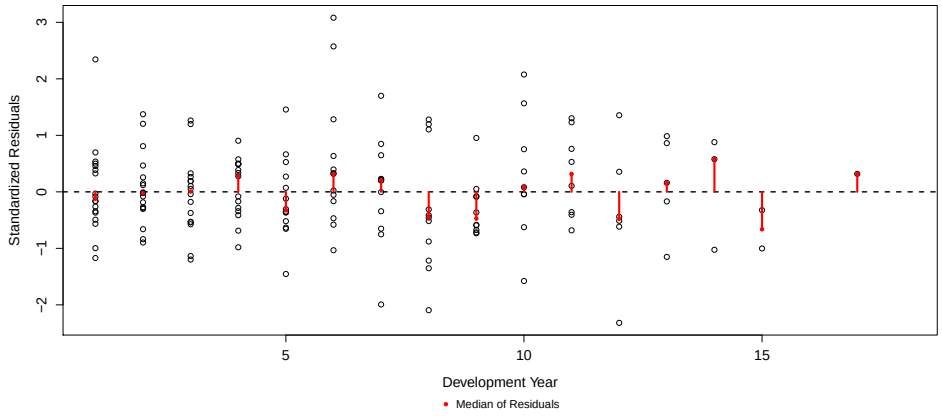

Figure 26: (Standardized) Residuals by development year.

```
> triResi(break.model.output, standardize = FALSE, timeAxis = "dy")
```

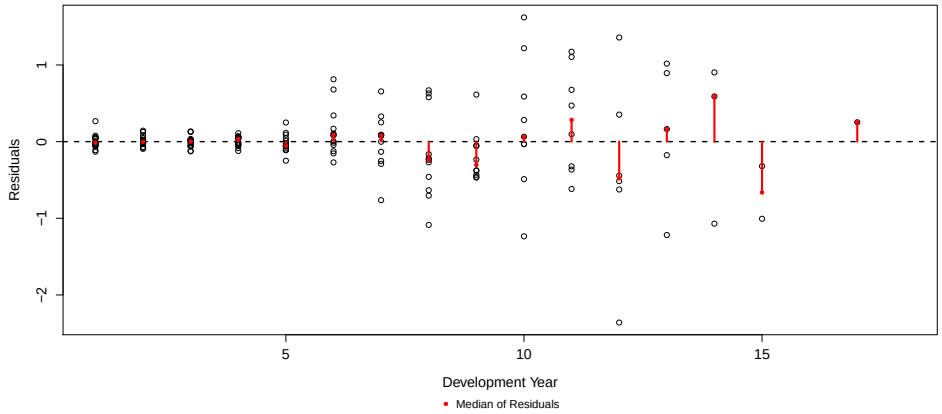

Figure 27: (Unstandardized) Residuals by development year.

Comparison of Cumulative Payments As previously mentioned, the loss triangle used to illustrate the change point model has a non-negligible number of zero incremental payments. Figure 28 uses the function `finalCumulativeDiff` to examine the impact of treating these values as missing.

```
> finalCumulativeDiff(break.model.output)
```

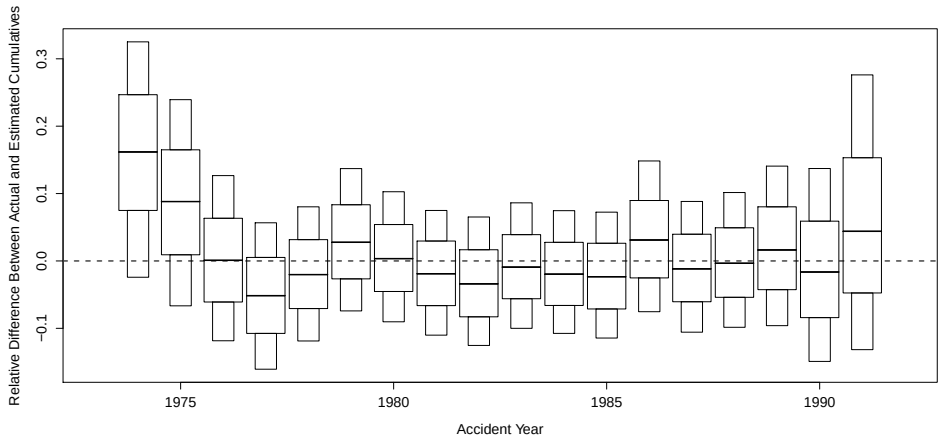

Figure 28: Difference in Final Observed Cumulative Payments.

4.4.3 Extracting Inference and Results

Just as was done for the standard model example, the user will want to draw inferences from the change point model. All of the functions discussed in Section 3.4.3 are available for this purpose– though some will plot slightly different charts and return answers in slightly different ways. In addition, a few functions are made available, which deal with the change point. These functions have no meaning for the standard model discussed in Section 3.

Predicted Payments Figure 29 again uses the function `predictedPayments` to plot the predicted incremental payments vs the observed incremental payments. The impact of treating incremental payments of zero as missing values is most noticeable in this chart.

```
> predictedPayments(break.model.output, type = "incremental", logScale = TRUE)
```

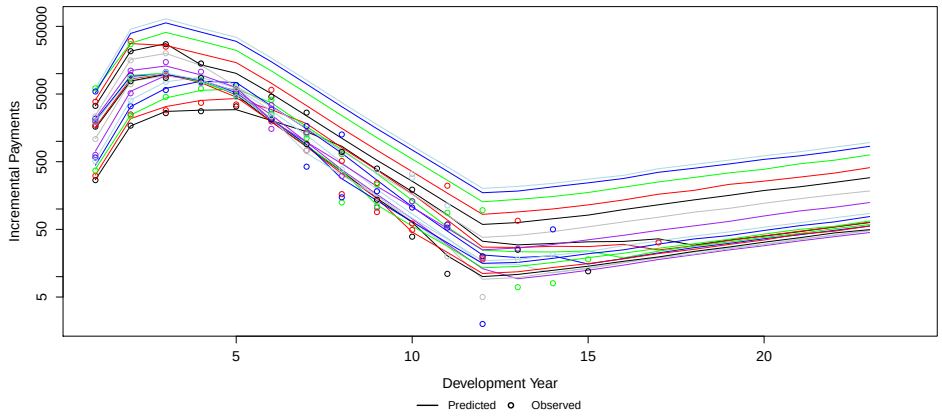

Figure 29: Predicted Incremental Payments.

Consumption Path Figure 30 plots the consumption path for the change point model, again using the function `consumptionPath`. Notice that now two consumption paths are plotted – one for the pre-break path and one for the post-break path. Both the pre and post break paths are normalized to the first exposure year level.

```
> consumptionPath(break.model.output)
```

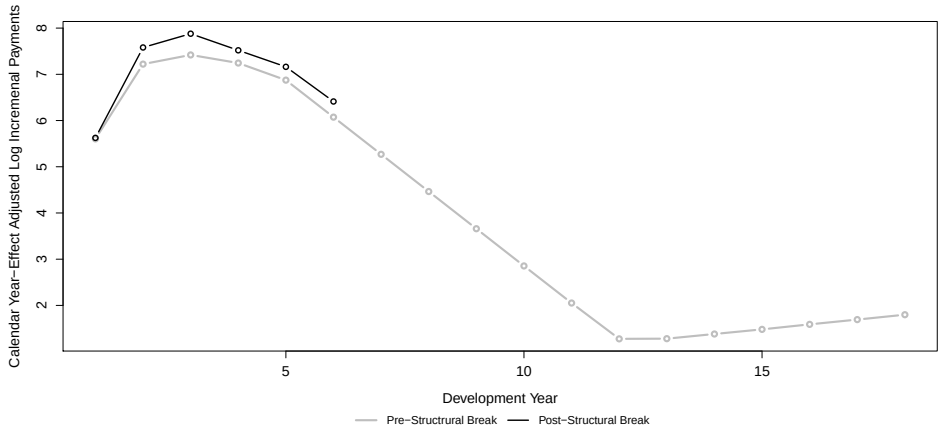

Figure 30: Consumption Path.

Knots in the Consumption Path Figure 31 displays the posterior number of knots for the change point model example, again using the function `numberOfKnots`. Notice that both the number of knots in the pre-break consumption path and the post-break consumption path are plotted.

```
> numberOfKnots(break.model.output)
```

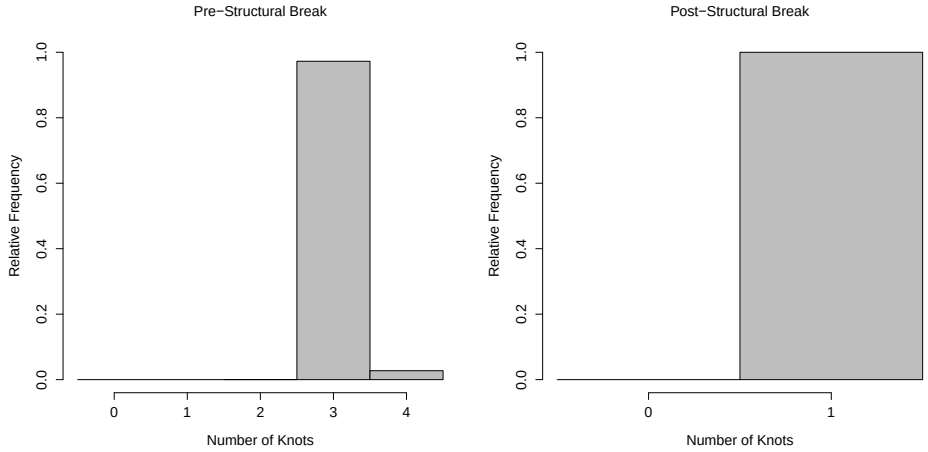

Figure 31: Number of Knots.

Rate of Decay Figure 32 uses the function `rateOfDecay` to plot the rate of decay from one development year to the next for both the pre and post break regimes. This can be useful in assessing the impact of changes in the environment effecting loss development.

```
> rateOfDecay(break.model.output)
```

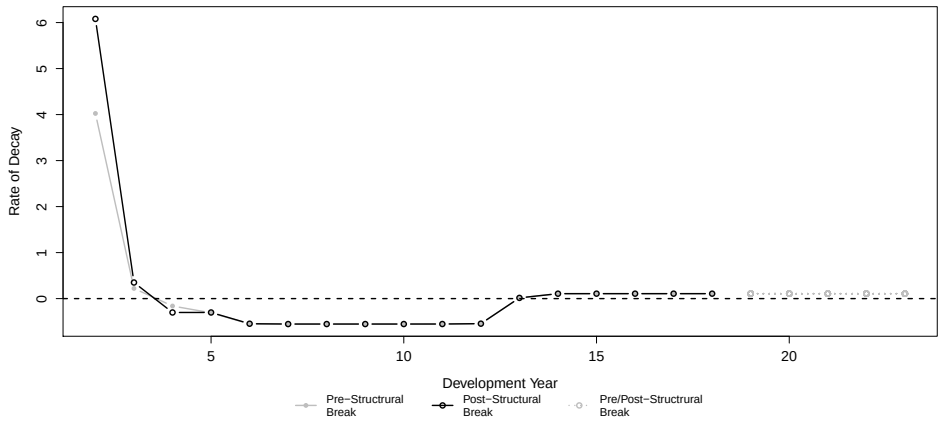

Figure 32: Rate Of Decay.

Calendar Year Effect Figure 33 uses the function `calendarYearEffect` to plot the calendar year effect for the change point model. By default, `calendarYearEffect` will plot the calendar year effect for all (observed and projected) incremental payments. Setting the argument `restrictedSize` to TRUE will only plot the calendar year effect for the observed incremental payments and the projected incremental payments needed to "square" the triangle.

Figure 34 shows the chart resulting from calling the function `calendarYearEffectErrors`.

Autocorrelation in Calendar Year Effect The autocorrelation exhibited in Figure 34 is too strong to ignore. As such, Figure 35 illustrates the use of `makeBreakAnnualInput`'s argument

```
> calendarYearEffect(break.model.output)
```

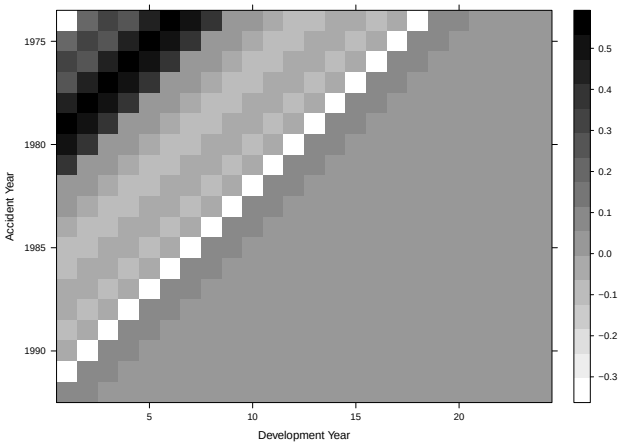

Figure 33: Calendar Year Effect.

`use.ar1.in.calendar.year`.

```
> calendarYearEffectErrors(break.model.output)
```

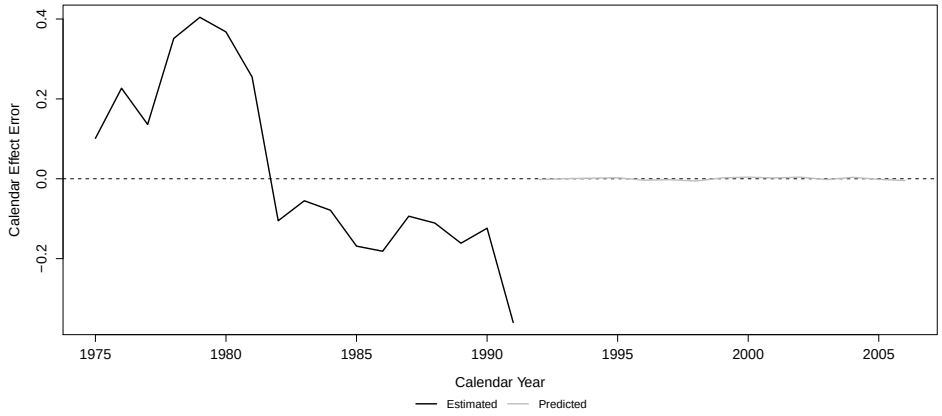

Figure 34: Calendar Year Effect Errors (Without AR1).

Setting `use.ar1.in.calendar.year` to TRUE enables the use of an additional function `autoregressiveParameter`. This function plots the autoregressive parameter associated with the calendar year effect. Figure 36 illustrates the use of this function.

Skewness Parameter Figure 37 displays the skewness parameter for the change point model example by using the function `skewnessParemeter`. The result of setting `bound.for.skewness.parameter` to 5 is visible in the chart.

First Year in New Regime The posterior for the first year in which the post-break consumption path applies can be obtained via the function `firstYearInNewRegime`. Figure 38 shows the posterior (and prior) for the first year in the new regime in the specified example. Note how the

```
> break.model.input.w.ar1 <- makeBreakAnnualInput(cumulative.payments = CumulativeAutoBodilyInjur
+     stoch.inflation.weight = 1, non.stoch.inflation.weight = 0,
+     stoch.inflation.rate = MCPI.rate, first.year.in.new.regime = c(1986,
+         1987), priors.for.first.year.in.new.regime = c(2, 1),
+     exp.year.type = "ay", extra.dev.years = 5, use.skew.t = TRUE,
+     bound.for.skewness.parameter = 5, use.ar1.in.calendar.year = TRUE)
> break.model.output.w.ar1 <- runLossDevModel(break.model.input.w.ar1,
+     burnIn = 30000, sampleSize = 40000, thin = 10)

Compiling data graph
   Resolving undeclared variables
   Allocating nodes
   Initializing
   Reading data back into data table
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 14498

[1] "Update took 2.32852805554867"

> calendarYearEffectErrors(break.model.output.w.ar1)
```

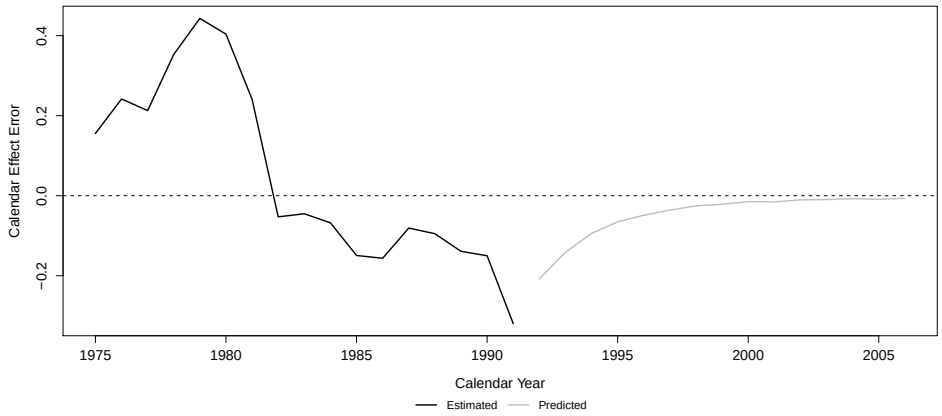

Figure 35: Calendar Year Effect Errors (With AR1).

```
> autoregressiveParameter(break.model.output.w.ar1)
```

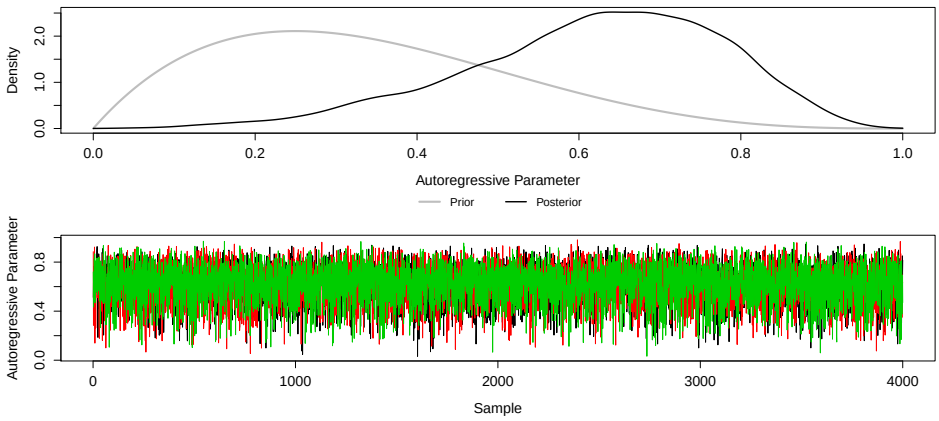

Figure 36: Calendar Year Effect Autoregressive Parameter.

```
> skewnessParameter(break.model.output)
```

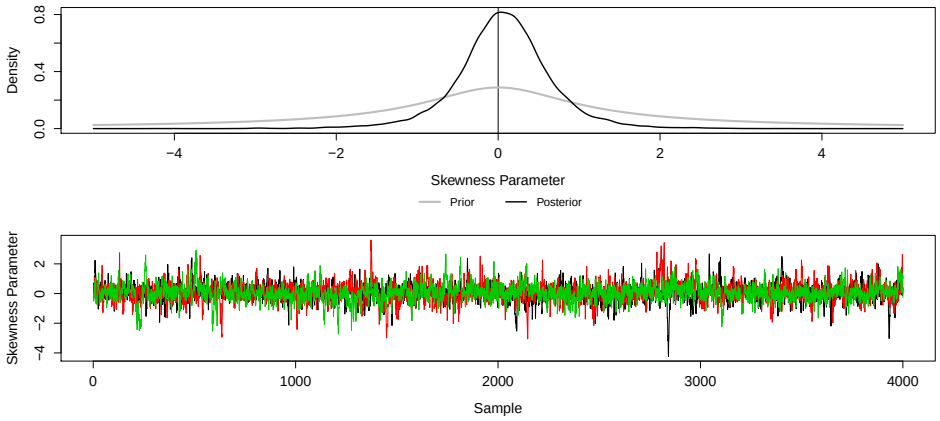

Figure 37: Skewness Parameter.

choice of the argument `priors.for.first.year.in.new.regime` to `makeBreakAnnualInput` has affected the prior.

```
> firstYearInNewRegime(break.model.output)
```

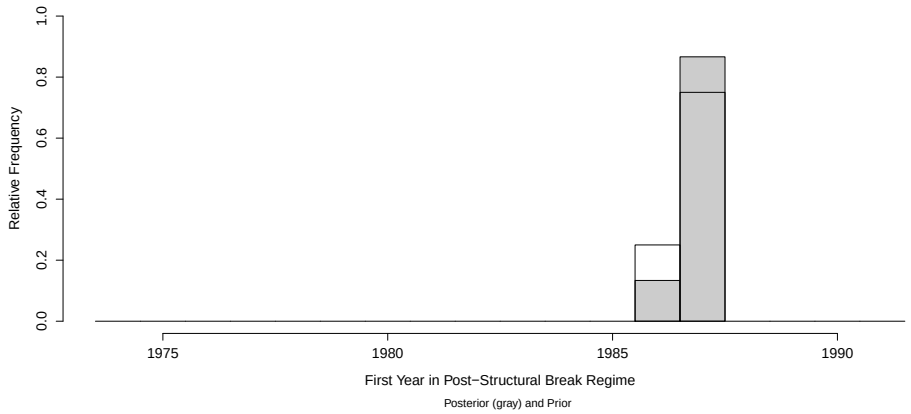

Figure 38: First Year in New Regime.

5 Accounting for Incremental Payments of Zero

As mentioned in Section 4.2.1 and illustrated in Figure 29, the triangle used as an example for the break model contains many incremental payments of zero which, if ignored, would cause the predicted losses to be overestimated.

`lossDev` provides a means to account for these zero payments. This is done by estimating a secondary, auxiliary model to determine the probably that a payment will be greater than zero. Predicted payments are then weighted by this probability.

5.1 Estimating the Auxiliary Model

To account for zero payments, the function `accountForZeroPayments` is called with the first argument being an object returned from a call to `runLossDevModel`. This function then returns another object which, when called by certain functions already mentioned, incorporates into the calculation the probability that any particular payment is zero.

```
> break.model.output.w.zeros <- accountForZeroPayments(break.model.output)

Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 3073

[1] "Update took 1.57576666673025"
```

5.2 Assessing Convergence of the Auxiliary Model

The MCMC run used to estimate the auxiliary model must also be checked for convergence. `lossDev` provides the function `gompertzParameters` to this end.

The auxiliary model uses a two parameter Gompertz function to model the zero incremental payments. Which of these parameters is plotted by `gompertzParameters` is determined by the argument `parameter`.

Figure 39 plots the parameter that determines the steepness of the curve and is obtained by setting `parameter` equal to "scale."

```
> gompertzParameters(break.model.output.w.zeros, parameter = "scale")
```

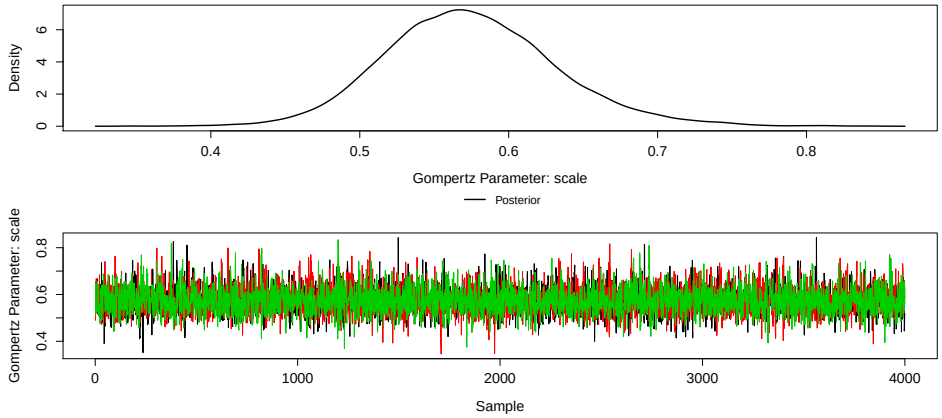

Figure 39: Gompertz Scale Parameter.

Figure 40 plots the parameter which determines the point in development time at which the curve assigns equal probability to payments being zero and payments being greater than zero and is obtained by setting `parameter` equal to "fifty.fifty."

```
> gompertzParameters(break.model.output.w.zeros, parameter = "fifty.fifty")
```

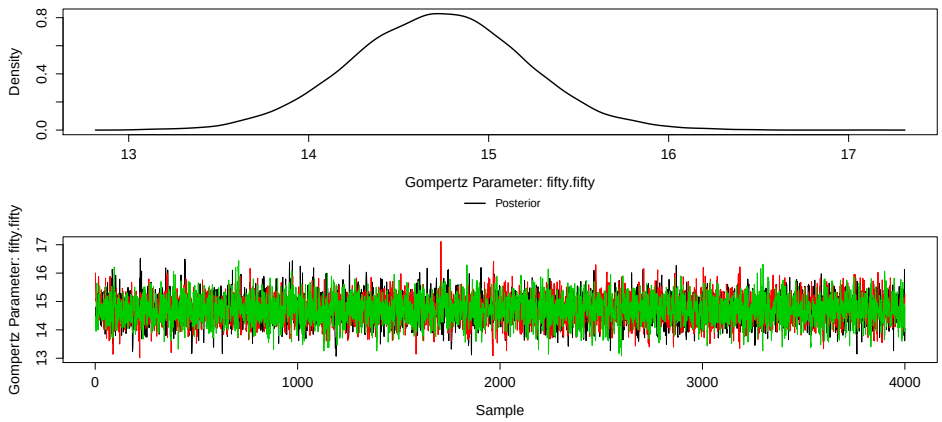

Figure 40: Gompertz Location Parameter.

5.3 Assessing Fit of the Auxiliary Model

One can plot the observed empirical probabilities of payments being greater than zero against the predicted (and projected) probabilities. This is done with the function `probablityOfPayment`.

Figure 41 plots this chart.

```
> probablityOfPayment(break.model.output.w.zeros)
```

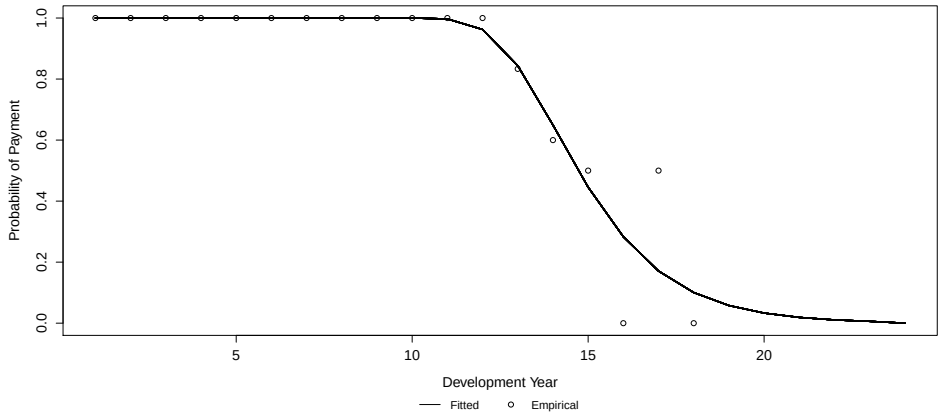

Figure 41: Probability of Non-Zero Payment.

5.4 Incorporating the Probability of Non-Zero Payment

Once the auxiliary model has been estimated and its output verified. The functions `predictedPayments`, `finalCumulativeDiff`, and `tailFactor` will incorporate this information into their calculations.

Figure 42 displays the predicted incremental payments after accounting for the probability that they may be zero. This should be compared with Figure 29, which does not account for the probability that payments may be zero.

```
> predictedPayments(break.model.output.w.zeros, type = "incremental",
+     logScale = TRUE)
```

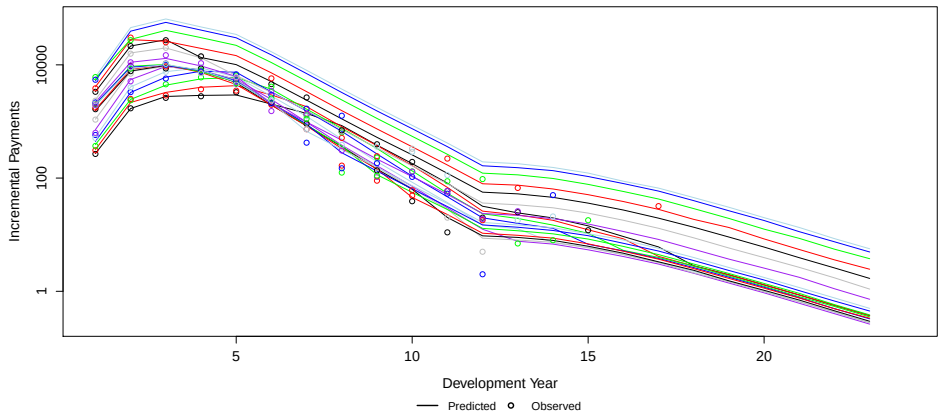

Figure 42: Predicted Incremental Payments (Accounting for Zero Payments).