`NGFMfitDistr` Workflow

Charlie Wusuo Liu

September 5, 2023

Abstract

This document contains code examples of producing NGFM-compliant PMF table using R package `NGFMfitDistr`.

Contents

1 Introduction 1
1.1 Import data 1
1.2 Model P_0 1
1.3 Compute "empirical" PMFs without P_0 1
1.4 Fit TrB to 18999 "empirical" PMFs 1
1.5 Discretization 1
2 Code example I 1

1 Introduction

A new methodology for generating NGFM-compliant distributions is implemented in R package `NGFMfitDistr`. The software facilitates an end-to-end computing pipeline from claims data to PMF table. It also manages deductibles and incorporates Bayesian updates. Main steps of the pipeline are:

1. Import data: MDRs (Modeled Damage Ratio), damage ratios.
2. Model P_0.
3. Compute e.g. 18999 "empirical" PMFs without P_0.
4. Fit Transformed Beta (TrB) to the 18999 PMFs.
5. Discretization.

For coverages A, B, C, the NGFM-compliant distribution table contains PMFs associated to 19001 `targetMDRs` $= \{0, 0.00001, 0.00002, \ldots, 0.09999, 0.1, 0.1001, 0.1002, \ldots, 0.9999, 1\}$. Distributions associated to MDR=0 and MDR=1 are assumed degenerate, thus ignored during fitting.

The MDR unit in coverage D is day: `targetMDRs`$= \{0, 0.00001, 0.00002, \ldots, 0.09999, 0.1, 0.1001, 0.1002, \ldots, 0.9999, 1, 1.001, 1.002, \ldots, 9.999, 10, 10.01, 10.02, \ldots, 99.99, 100, 100.1, 100.2, \ldots, 999.9, 1000, 1001, \ldots, 1096\}$.

1.1 Import data

Claims data are given as a two-column matrix where each row is an MDR and a DR (damage ratio). We order rows of the matrix by MDR, and remove rows where MDR is 0 or 1.

1.2 Model P_0

Users will define a sliding window size and a sliding speed over the data. For instance, the sliding window size can be 1% of the number of claims, and the sliding speed can be 1% of the window size. This will yield about 10,000 overlapped windows of data. In each window, we average MDRs as mMDR, and compute the proportion of zero DRs, namely P_0.

The pairs of (mMDR, P_0) in all windows are gathered. Our goal is to estimate $\hat{P}_0$ for every element in `targetMDRs`. Additionally, NGFM standard dictates that $\hat{P}_0$ should be nonincreasing with increasing `targetMDRs`. To this end, we (i) find the longest nonincreasing subsequence of P_0s along with increasing mMDRs, (ii) interpolate pairs of (mMDR, P_0) in the subsequence to obtain a nonincreasing function, (iii) supply `targetMDRs` to the function for $\hat{P}_0$s. Please run `vignette("slides", package = "NGFMfitDistr")` to see more details in the slide vignette.

1.3 Compute "empirical" PMFs without P_0

Let $\hat{P}_0(\texttt{targetMDR})$ denote the modeled P_0 for a `targetMDR`. The target mean for the main part of the distribution is

$$\texttt{targetMDR}_{\text{main}} = \frac{\texttt{targetMDR}}{1 - \hat{P}_0(\texttt{targetMDR})}.$$

We delete all rows where DR=0 from the claims data, and reapply the 1% window size and 0.01% sliding speed to obtain about 10,000 windows of data. In each window, we compute the mean MDR and the "empirical" distribution of DRs. Let (mMDR, PMF) denote the result. Computing the PMF is similar to making a histogram except that we use regriding instead of binning and frequency normalization. Please check `keyALGs` in `//air-worldwide.com/data/financialmodule/Rpackages` for details about regriding.

The quotes on "empirical" suggest the distribution is not entirely empirical, because PMF is a lossy characterization of the data. Typically we set 64 as the PMF's support size (number of points).

There are 18,999 $\texttt{targetMDR}_{\text{main}}$s. Our goal is to associate each of them with an "empirical" PMF, and then to fit the PMF with a TrB model. The approach is as follows. In the sequence of mMDRs, we search the adjacent pair that $\texttt{targetMDR}_{\text{main}}$ falls between. The "empirical" PMF for $\texttt{targetMDR}_{\text{main}}$ is then made as a mixture of the two PMFs associated with the mMDRs. The mixture weight is determined by the relative position of $\texttt{targetMDR}_{\text{main}}$ in $[\text{mMDR}_{\text{low}}, \text{mMDR}_{\text{high}}]$. Please see `vignette("slides", package = "NGFMfitDistr")` for more details.

1.4 Fit TrB to 18999 “empirical” PMFs

We first fit the PMF associated to the median MDR in the original data, e.g. 0.05. Using the optimized TrB parameters as initialization, we next fit the PMF associated to 0.0501. The optimized parameters for 0.0501 are then used to initialize the fitting for 0.0502.

After the PMF for 0.9999 is fitted, we reverse direction. We use the optimized parameters for 0.05 to initialize the fitting of PMF for 0.0499, and then 0.0498, and so on, down to 0.00001.

The fitting method is referred to as *bidirectional sequential fitting*. The motivation is to promote smooth parameter transition between close `targetMDR`$_{\text{main}}$s. The package also implements naive fitting procedures that do not exploit the sequential locality. The naive approaches usually run much slower and tend to produce erratic parameter behaviors along the `targetMDR`$_{\text{main}}$ axis.

TrB has four parameters a, b, c, d. Because the TrB’s mean needs to equal `targetMDR`$_{\text{main}}$, we can solve the scale parameter d on the fly given `targetMDR`$_{\text{main}}$ and the values of a, b, c at the time. This effectively reduce the problem’s dimensionality to three for the L-BFGS-B optimizer. We choose L-BFGS-B for its efficiency and robustness. It is widely recognized as the go-to quasi-Newton method for continuous optimization under box constraints.

For more details, please see `vignette("slides", package = "NGFMfitDistr")`.

1.5 Discretization

An *inflated TrB distribution* comprises $\hat{P}_0$ and the TrB model for the main part. We have 18,999 such distributions to be discretized on 42/64-point supports. The final PMF table should also satisfy the following monotonicity constraints:

- Respectively, $\hat{P}_0$s, maxes, $\hat{P}_{\text{max}}$s, the coefficients of variation are nonincreasing, nondecreasing, nondecreasing, nonincreasing with increasing `targetMDR`s.

Here, max is the high endpoint on PMF’s support and $\hat{P}_{\text{max}}$ is the probability associated with it. Coefficient of variation is the PMF’s standard deviation divided by its mean.

In general, we first set the max equal to some high percentile (e.g. the 99th) of the distribution. Then we discretize TrB onto a fine regular grid of e.g. 2,000 points. Finer grid incurs less error between the PMF’s mean and the TrB’s mean. The fine PMF is then regrided onto a 42/64-point support. The regriding algorithm preserves the mean exactly.

After the regriding, the monotonicity requirements are hardly fully satisfied on a micro scale. We massage the probabilities in the PMF table to meet the requirements. The massaging is based on interpolation of the longest monotonic subsequence. Please see `vignette("slides", package = "NGFMfitDistr")` for more details.

2 Code example I

Load New Zealand coverage A data included in `NGFMfitDistr`. Rename it for simplicity. Check the data’s structure:

```r
data("NewZealandCvgA", package = 'NGFMfitDistr')
dat = NewZealandCvgA
str(dat)
# 'data.frame':  178398 obs. of  2 variables:
# £ MDR: num  1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 1e-04 ...
```

```
# £ CDR: num  0 0 0 0 0 0 0 0 0 0 ...
```

Feed `targetMDRs` and the data to `NGFMfitDistr::fullFit()`:

```
targetMDRs = c(1:10000, seq(1e4L + 10L, 1e5L - 10L, by = 10L)) / 1e5

optRst = NGFMfitDistr::fullFit(
  mdr = dat$MDR,
  cdr = dat$CDR,
  windowSize = as.integer(round(nrow(dat) * 0.01)),
  slidingSpeed = as.integer(round(nrow(dat) * 1e-4)),
  sampleSize = round(nrow(dat) * (1 - 1 / exp(1))),
  NsampleSets = 100L,
  interpolationMethod = "linear",
  linearIntpoThenCpp = TRUE,
  targetMDRs = targetMDRs,
  randomSeed = 42L,
  maxCore = parallel::detectCores(),
  deductible = NULL,
  isDedFranchise = TRUE,
  givenP0 = NULL,
  nonDegenerateOldDistrs = NULL,
  sampleWeightOnOldDistrs = 0.8,
  empDistrSupportSize = 64L,
  regridMethod = "lr",
  tempDir = "../tempFiles/CharlieMP/C",
  figureDir = "../tmpfigure",
  fitToBiasCorrectedPMFs = FALSE,
  abc = matrix(c(4, 5, 6)),
  abcLB = c(1.01, 0.1, 0.1),
  abcUB = c(30, 30, 30),
  startingPmf = 0L,
  scaleEps = 1e-8,
  scaleMaxit = 100L,
  distanceFun = "likelihood",
  max_iterations = 100L,
  RIBlib = "Numerical Recipes",
  sequentialUpdate = -1,
  hgrad = 0,
  centralDiff = TRUE,
  verbose = TRUE,
  m = 6,
  epsilon = 1e-5,
  epsilon_rel = 1e-5,
  past = 1,
  delta = 1e-10,
  max_submin = 10,
  max_linesearch = 20,
  min_step = 1e-20,
  max_step = 1e+20,
  ftol = 1e-4,
  wolfe = 0.9
)
```

Most of the function arguments have default values that usually need no changes. Please refer to the package manual for more details on the function.

In the sibling directory `figureDir = "../tmpfigure"`, we can find `p0models.png` and

`TrBtransitionParam.png`:

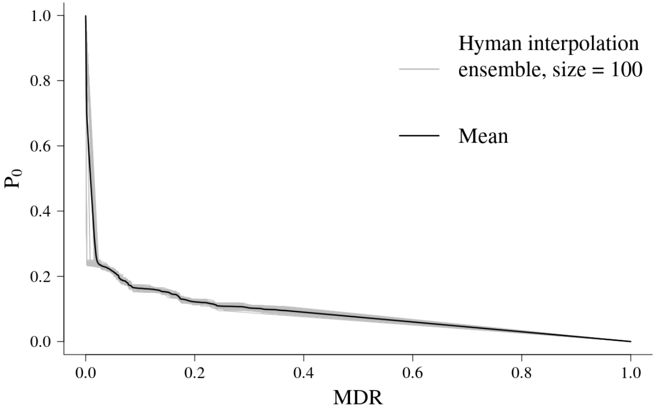

Figure 1: `p0models.png` in `../tmpfigure`. It shows the ensemble model for P_0 and mean of the ensemble.

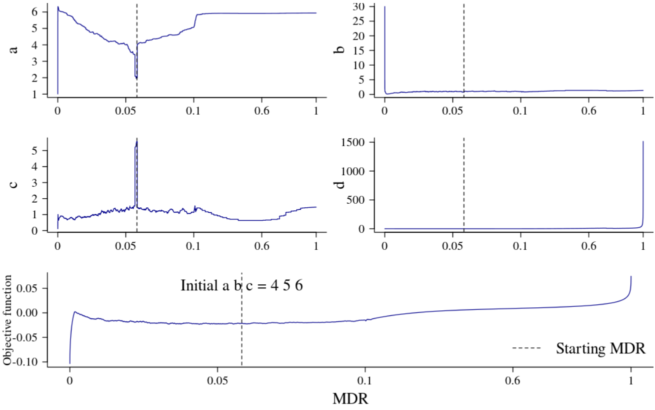

Figure 2: `TrBtransitionParam.png` in `../tmpfigure`. It shows the fitted TrB parameters a, b, c, d and the objective function value.

Figure 2 shows a cliff in a and a spike in c around the starting MDR (see Section 1.4). The behavior is due to: (i) the initialization guided the optimizer into some local optimum, (ii) when the cumulative change in the "empirical" PMF becomes large enough, the optimizer will slide into another local optimum that is far away from the original. Note that this does not imply nonnegligible changes in probability densities or the objective function value. Although unnecessary, we can trial-and-error a different initialization to smooth the transition.

In Figure 2, the value of a around the cliff is about 4. The corresponding value of b is about 1. The value of c around the spike is about 1.5. Therefore we may try 4, 1, 1.5 as the initialization for a, b, c:

```
optRst = NGFMfitDistr::fullFit(
  mdr = dat$MDR,
  cdr = dat$CDR,
  windowSize = as.integer(round(nrow(dat) * 0.01)),
  slidingSpeed = as.integer(round(nrow(dat) * 1e-4)),
  sampleSize = round(nrow(dat) * (1 - 1 / exp(1))),
  abc = matrix(c(4, 1, 1.5)),
  figureDir = "../tmpfigure2"
)
```

Figure 3 shows the new TrB parameter transition. The objective function value has no visible change.

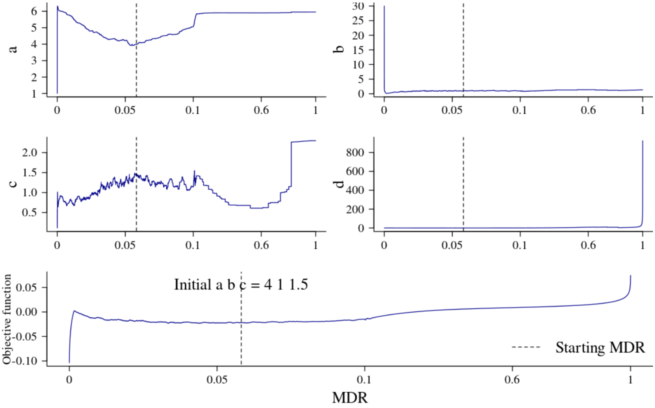

Figure 3: `TrBtransitionParam.png` in `../tmpfigure2`. Initial $a, b, c = 4, 1, 1.5$.

Before discretization, we set maxes of the PMFs' supports as the 99th percentiles of the TrBs. The 99th percentile is not the only choice for making maxes.

```
# Assemble the matrix of a, b, c, d:
param = optRst$optRst$param # param contains a,b,c and the main part's target mean.
d = NGFMfitDistr::solve_d(param, eps= 1e-8, maxit = 100)
abcd = param
abcd[4, ] = d

# Compute the maxes as the 99th percentile of the TrB distribution.
maxQuantile = 0.99
tailProbThreshold = 1 - maxQuantile
maxes = pmin(1, NGFMfitDistr::actuar_qtrbeta(
```

```
  1 - tailProbThreshold / (1 - optRst$P0), abcd))

# Non-finite values could be resulted due to limited numeric precision.
maxes[!is.finite(maxes)] = 0
maxLowerBound = 0.05 # Put a lower bound on the maxes.
maxes = pmax(maxLowerBound, maxes)

# Find the longest nondecreasing subsequence.
ind = NGFMfitDistr::longestNonDecreasingSubseq(maxes)

# Interpolation.
maxFun = splinefun(x =  targetMDRs[ind], y = maxes[ind], method = "linear")
maxes = maxFun(targetMDRs) # Obtain all the maxes.
```

Making the maxes can also be simplified to one function call:

```
maxes = NGFMfitDistr::makeMax(
  MDR = targetMDRs, P0 = optRst$P0, abcd = abcd, tailThreshold = 1 - 0.99,
  minMax = 0.05, interpolationMethod = "linear")
```

The final step is to input the maxes, $\hat{P}_0$ and TrB parameters to the discretization function:

```
TrBtable = rbind(targetMDRs, maxes, optRst$P0, abcd)
pmftable = NGFMfitDistr::fullDiscretize(
  TrBtable = TrBtable,
  supportSizes = c(42L, 64L), # Discretize TrBs on both 42 and 64-point supports.
  regridMethod = "lr",
  RIBlib = "Numerical Recipes",
  outputProbsInRows = TRUE,
  fineDiscretizationSize = 2000,
  maxCore = 1000,
  verbose = TRUE,
  downScaleSupport = FALSE,
  figureDir = "../tmpfigure",
  empDistrsLists = optRst$empDistrsLists,
  mdrRangeInData = optRst$mdrRangeInData,
  claimsDataForScoring = NULL
)
```

The TrB parameter table and the 42/64-point PMF tables are saved as follows:

```
tb = as.data.frame(pmftable$tuned$p42)
colnames(tb) = c("MDR", "max", paste0("P", 1:(ncol(tb) - 2L)))
data.table::fwrite(tb, row.names = TRUE,
                 file = "../tempFiles/examplePMF42.csv")

tb = as.data.frame(pmftable$tuned$p64)
colnames(tb) = c("MDR", "max", paste0("P", 1:(ncol(tb) - 2L)))
data.table::fwrite(tb, row.names = TRUE,
                 file = "../tempFiles/examplePMF64.csv")

tb = as.data.frame(t(TrBtable))
colnames(tb) = c("MDR", "max", "P0", "a", "b", "c", "d")
data.table::fwrite(tb, file = "../tempFiles/exampleTrBtable.csv")
```

Historically, the PMF table had an extra column of row indices, so we set `row.names = TRUE`.

P_0s and maxes in `examplePMF64.csv`, `examplePMF42.csv` and `exampleTrBtable.csv` are not necessarily equal due to unavoidable errors in discretization and fine tuning for imposing monotonicity.