A Comparison of Resampling Methods for Bootstrapping Triangle GLMs

by Thomas Hartl

ABSTRACT

Bootstrapping is often employed for quantifying the inherent variability of development triangle GLMs. While easy to implement, bootstrapping approaches frequently break down when dealing with actual data sets. Often this happens because linear rescaling leads to negative values in the resampled incremental development data. We introduce two computationally efficient methods for avoiding this pitfall: split-linear rescaling and parametric resampling using a limited Pareto distribution. After describing the essential mathematical properties of the techniques, we present a performance comparison based on a VBA for Excel bootstrapping application. The VBA application is available on request from the author.

KEYWORDS

Bootstrapping and resampling methods, generalized linear modeling, efficient simulation, stochastic reserving, regression

1. Introduction

Several authors have described the use of bootstrapping in the context of GLM-based stochastic reserving models for property and casualty loss development triangles; e.g., England and Verrall (2002), Pinheiro, Silva and Centeno (2003), and Hartl (2013). A more general textbook treatment of the bootstrapping procedure for GLMs can be found in Davison and Hinkley (1997, Section 7.2). While alternative approaches are mentioned, practical examples that are based on linear rescaling of Pearson residuals are often presented. This approach has the appeal of straightforward mathematics: all you need is one addition and one multiplication. As a result, it can be implemented with relative ease in a spreadsheet.

For real data, however, the linear rescaling of Pearson residuals often leads to negative values in the resampling distribution for smaller data values, thus violating the model assumptions of the GLM we started with. In other words, the bootstrapping procedure fails. Both the details of this breakdown and the alternative method of deviance residual resampling are discussed in Hartl (2010). Other authors, e.g., Gluck and Venter (2009), employ parametric resampling and thus sidestep the problem. While parametric resampling circumvents the issue of negative values in the resampling distribution, there is an increased computational cost because generating pseudo-random variates from typical exponential family distributions requires the evaluation of logs, exponentials or even more computationally intensive functions. As mentioned above, linear rescaling of Pearson residuals, by contrast, only requires one addition and one multiplication.

In this paper we propose two alternative methods for generating pseudo data: split-linear rescaling and sampling from a limited and shifted Pareto distribution. During the time-critical Monte Carlo iteration phase of the bootstrapping procedure, both of the proposed alternative methods are about the same or better in terms of computational cost than linear rescaling of Pearson residuals. In Section 2 we describe basic mathematical properties of split-linear rescaling, and sampling from a limited and shifted Pareto distribution. Proofs of some key formulas from this section are given in Appendix A. Section 3 is more technical in nature and can be skipped on a first reading, but it does contain information of interest to readers who want to implement the proposed methods themselves. The algorithms and memory requirements needed for linear rescaling, split-linear rescaling, and sampling from a limited and shifted Pareto distribution are discussed. We present performance comparisons based on a VBA for Excel implementation, and thus demonstrate that the new methods are computationally as efficient as linear rescaling. The VBA application is available from the author on request. Conclusions are discussed in Section 4.

2. Two schemes for generating pseudo data

Before delving into the mathematical properties of the proposed methods, we briefly review resampling for bootstrapping purposes and introduce the notation used in this paper.

2.1. Resampling basics

Bootstrapping is a Monte Carlo simulation technique that is employed to approximate the sampling distribution of a quantity that is estimated from a data sample. To do so, one repeatedly generates pseudo data and re-estimates the quantity in question.

The key idea of non-parametric bootstrapping is to sample the standardized observed residuals relative to the stochastic model that is assumed to generate the original data. The sampled standardized observed residuals are then used to generate the pseudo data. As pointed out in Davison and Hinkley (1997), due to the non-unique definition of residuals for GLMs, there are multiple non-parametric residual resampling methods. We also observe that the various residual standardization schemes typically

fail to strictly standardize the residuals.[1] To proceed with the bootstrap we therefore we must consider the iid assumption for the standardized residuals as being "approximately" true.

How these ideas are applied to GLM-based stochastic reserving is described in England and Verrall (2002) and Pinheiro, Silva, and Centeno (2003). An alternative to non-parametric bootstrapping is parametric resampling, where the pseudo data are generated by sampling from an assumed parametric distribution. One version of this approach is used in Gluck and Venter (2009). In either case, pseudo data are repeatedly generated and the model is refitted to the pseudo data. In this way, one generates a Monte Carlo sample of the distribution of fitted parameters or other derived quantities, such as the projected reserve.

2.2. Notation

In the context of our discussion we will have to refer both to individual values and to collections of similar values. To do so we follow the convention that **boldface** symbols refer to vector quantities while non-boldface symbols refer to scalar quantities. For example, the symbol y refers to a generic element of $\boldsymbol{y}$. Where we do need to distinguish between different elements of $\boldsymbol{y}$, we will employ subscripts such as y_i and y_j.

More specifically, $\boldsymbol{y}$ refers to the vector of original data. The hat symbol, as in $\hat{\boldsymbol{y}}$ or $\hat{y}$, indicates that we are talking about fitted values. The asterisk symbol, finally, is used to refer to a full set of pseudo data, $\boldsymbol{y}^*$, or an individual pseudo datum, y^*.

For residual-based bootstrapping, we need a set of standardized residuals, which we denote by $\boldsymbol{s}$. To avoid bias as a result of the resampling process, we require the mean of $\boldsymbol{s}$ to be exactly zero. This is in addition to the usual requirement that the variance of s equals one. It is useful to think of a standardized residual as a function of an original data value and the fitted value predicted by the GLM

$$s = H_{\hat{y}}(y). \tag{2.1}$$

The function $H_{\hat{y}}(\cdot)$ depends on the definition of residual and the standardization procedure employed. This is discussed in detail in McCullagh and Nelder (1989), Davison and Hinkley (1997), or Pinheiro, Silva, and Centeno (2003). For residual-based bootstrapping, obtaining a pseudo datum (a.k.a. resampled data value) is a two-step process: we randomly sample from the standardized residuals, and then solve the following equation for y^*

$$\text{Rnd}(\boldsymbol{s}) = H_{\hat{y}}(y^*).$$

Note that the function $\text{Rnd}(\boldsymbol{s})$ denotes the operation of randomly sampling from the set of standardized residuals. We summarize the resampling procedure for an individual data point as

$$y^* = H_{\hat{y}}^{-1}(\text{Rnd}(\boldsymbol{s})). \tag{2.2}$$

For a concrete example, we consider Pearson residual resampling for an over dispersed Poisson model. In this case equation (2.2) becomes

$$y^* = \hat{y} + \sqrt{\phi \hat{y}} \cdot \text{Rnd}(\boldsymbol{s}), \tag{2.3}$$

where ϕ is the dispersion factor. From this equation, one can easily see how a negative standardized residual and sufficiently small values of $\hat{y}$ will lead to negative resampling values. This breakdown is frequently encountered when using Pearson residuals for bootstrapping triangle GLMs. A more detailed discussion and examples of this can be found in Hartl (2010). As noted in subsection 2.1, non-parametric residual resampling relies on the assumption that standardized residuals are "approximately" iid. When

[1] An anonymous reviewer raised the point that the idd assumption often fails and shared the empirical observation that standardized residuals from the first column of a development triangle often appear to be from a different distribution than the rest. We observe that this is not just a typical empirical occurrence, but a mathematical truth: the standardized Pearson residuals for GLMs with different expected (positive) means for different observations cannot be identically distributed unless we assume a gamma variance function $V(\mu) = \mu^2$; this is clearly the case since the standardized Pearson residuals for different observations have different lower bounds.

the described breakdown of linear Pearson residual rescaling occurs, we can interpret this as saying that this approximation is not good enough for this data set, prompting one to look for alternatives.

2.3. Split-linear rescaling

Our goal is to avoid the breakdown of the resampling scheme that results from zero or negative pseudo data being generated. To this end we introduce a new parameter, π_{min}, that defines the smallest allowable pseudo data values as a percentage of the expected mean. Conceptually this could be different for each data point, but we use a ratio that is uniformly applied to all data points. One might argue that just requiring the pseudo data to be positive is sufficient, and that introducing π_{min} is overly restrictive. At the same time, since we are dealing with finite precision computer-based simulations, we may get into numerical trouble if we get too close to zero. Practitioners who want to explore what happens at the limits are welcome to choose very small values for π_{min}.

The intuitive idea behind split-linear rescaling is as follows: if the standard Pearson residual rescaling procedure results in resampling values that are below π_{min} times the expected mean value, we split the standard Pearson residual resampling values into a lower set and an upper set. Next we "squeeze" the lower resampling values together, so that they no longer dip below π_{min} times the expected mean value. The "squeezing" operation does preserve the mean, but it will lower the variance of the resampling distribution. To offset this, we apply a mean preserving "expansion" operation to the upper resampling values.

We proceed by defining what we mean by "squeezing" and "expanding." Given any set of values, $\boldsymbol{z}$, with mean μ, we can obtain a new set of values, $\boldsymbol{z}'$, that has the same mean. This is accomplished by applying the following linear transformation to each element z of $\boldsymbol{z}$

$$z' = \mu + c(z - \mu), \tag{2.4}$$

where the scaling factor, c, is a positive constant. If $c < 1$, we end up "squeezing" the values. If $c > 1$, the values are "expanded." The proof that transformation (2.4) preserves the mean is left as an easy exercise for the reader. Figure 1 graphically represents the impact of different values of c on a set of data points with values of 4.5, 6.5, and 13 and a mean of 8.

If σ_z^2 denotes the variance of $\boldsymbol{z}$, it is also easy to show that

$$\sigma_{z'}^2 = c^2 \sigma_z^2. \tag{2.5}$$

So we can see that if we "squeeze" a set of values, we will reduce the variance. Conversely, if we expand the values, we will increase the variance. For split-linear rescaling, however, we apply transformation (2.4) to two disjoint subsets of the original data set, so we need to consider the impact on the overall variance resulting from the transformation on each subset.

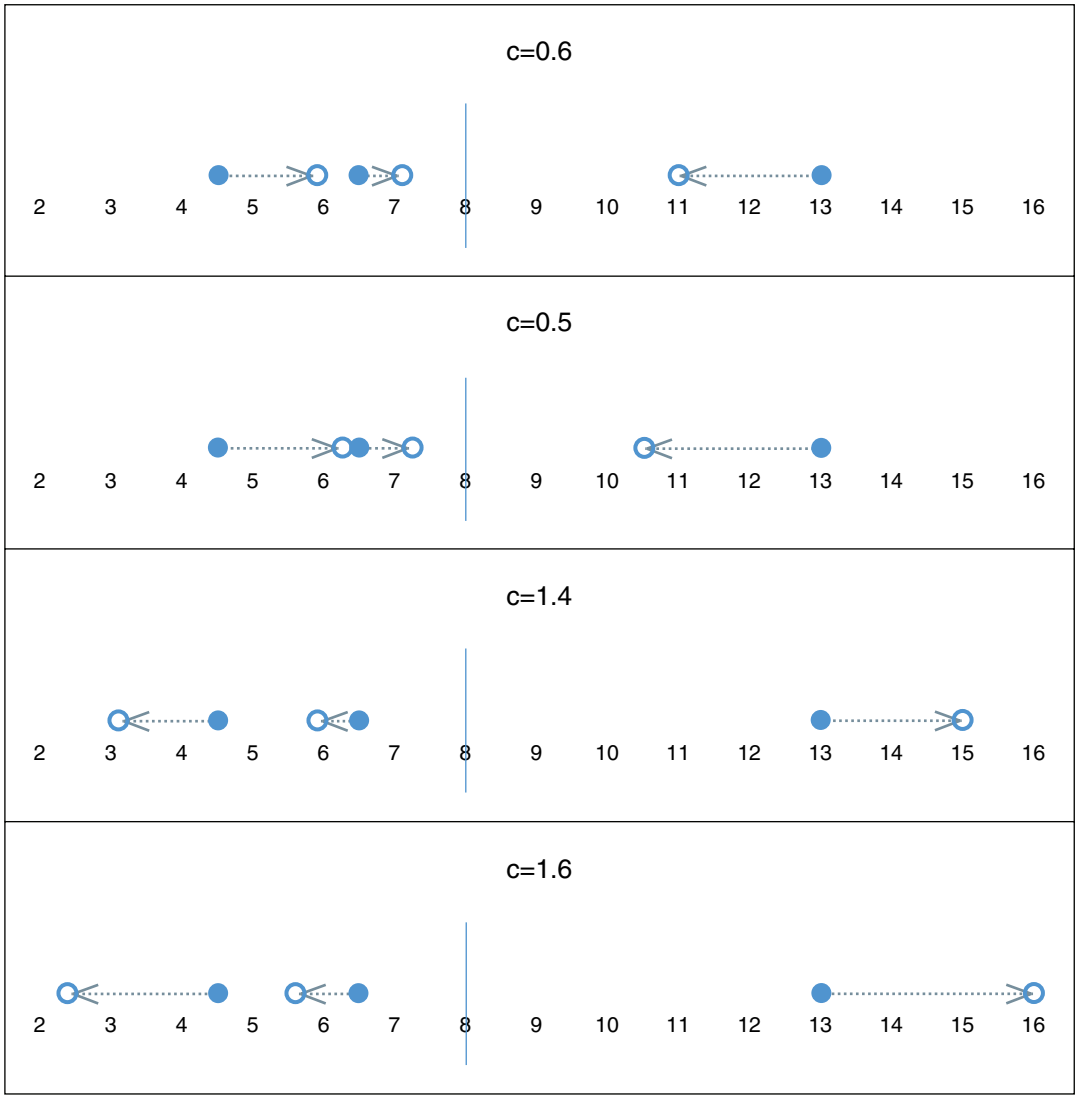
To be specific, let us assume that the original resampling distribution, $\boldsymbol{y}^*$, has m data values and mean μ. Let us furthermore assume that we have partitioned $\boldsymbol{y}^*$ into two subsets, $\boldsymbol{y}_l^*$ and $\boldsymbol{y}_u^*$, with q and r data values, respectively. The corresponding means are μ_l and μ_u. For convenience we will assume that the values in the lower subset are less than the values in the upper subset. Furthermore, we can assume without loss of generality that the original data set and both subsets are indexed in ascending order. In particular, the first element of each subset can be assumed to be the smallest element. The split-linear transformation can be summarized by the following equation,

$$y^{*\prime} = \begin{cases} \mu_l + c_l (y^* - \mu_l), \text{ if } y^* \in \boldsymbol{y}_l^* \\ \mu_u + c_u (y^* - \mu_u), \text{ if } y^* \in \boldsymbol{y}_u^{*\prime} \end{cases}, \tag{2.6}$$

where c_l and c_u, are the scaling factors for the lower and upper subset. Since transformation (2.4) preserves the mean of each subset, the overall mean is also preserved. In A.1 we show that the variance of the transformed resampling distribution can be expressed as

$$\sigma_{y^{*\prime}}^2 = \frac{1}{m}\begin{pmatrix} q(\mu_l - \mu)^2 + q c_l^2 \sigma_{y_l^*}^2 \\ + r(\mu_u - \mu)^2 + r c_u^2 \sigma_{y_u^*}^2 \end{pmatrix}. \tag{2.7}$$

Figure 1. Impact of different values of C



In practice we proceed by finding a suitable partition of $\boldsymbol{y}^*$ and a value of c_l such that we satisfy the condition that no value of $\boldsymbol{y}^{*\prime}$ drops below $\pi_{min}\mu$. To get a variance preserving split-linear transformation we then set the right-hand side of equation (2.7) equal to the variance of $\boldsymbol{y}^*$ and solve the resulting equation for c_u. We summarize the general procedure for split-linear rescaling as follows.

Step 0 *Generate* $\boldsymbol{y}^*$ *using Pearson residual rescaling. If* $y_1^* \geq \pi_{min}\mu$, *use* $\boldsymbol{y}^*$*—there is no need to use split-linear rescaling.*

Step 1 *Find a suitable partition of* $\boldsymbol{y}^*$ *such that* $\mu_l > \pi_{min}\mu$.

Step 2 *Calculate* c_l *using the following formula*

$$c_l = \frac{\mu_l - \pi_{min}\mu}{\mu_l - y^*_{l_1}}.$$

Step 3 *Calculate* c_u *according to*

$$c_u = \sqrt{1 + \left(1 - c_l^2\right)\frac{q\sigma^2_{y_l^*}}{r\sigma^2_{y_u^*}}}.$$

Step 4 *Generate* $\boldsymbol{y^{*\prime}}$ by applying transformation (2.6)

$$y^{*\prime} = \begin{cases} \mu_l + c_l\,(y^* - \mu_l), \text{ if } y^* \in \boldsymbol{y_l^*} \\ \mu_u + c_u\,(y^* - \mu_u), \text{ if } y^* \in \boldsymbol{y_u^*} \end{cases}. \qquad (2.8)$$

Proofs for the formulas given in equation (2.8) can be found in A.2 and A.3. The condition in Step 1 that $\mu_l > \pi_{min}\mu$ is necessary because the smallest value in the lower set, y_1^*, gets "squeezed" towards μ_l, so the transformed value $y_1{}^{*\prime}$ can only reach $\pi_{min}\mu$ if $\mu_l > \pi_{min}\mu$.

We also observe that for Step 1 there can be more than one of partition of $\boldsymbol{y^*}$ such that $\mu_l > \pi_{min}\mu$. So, which one should we choose? One strategy might be to simply take the first one that comes along, with the smallest number of elements in $\boldsymbol{y_l^*}$. The strategy we are employing is to find the partition that makes $c_u^2 - 1$ as close to $1 - c_l^2$ as possible, by minimizing the absolute difference between $q\sigma^2_{y_l^*}$ and $r\sigma^2_{y_u^*}$. This heuristic decision is motivated by a notion that the adjustment should be as evenly spread over the entire distribution as possible, rather than just "bunching" values in the left tail. Tracking $q\sigma^2_{y_l^*}$ and $r\sigma^2_{y_u^*}$ proved to be convenient because they can be updated recursively as we move values from the upper set to the lower set, and the optimal partition can be determined without testing all possible partitions.

Finally, we note that there are two situations in which the split-linear rescaling procedure does not work: breakdowns may occur during Step 3 or during Step 4. For Step 3 the breakdown happens if $\sigma^2_{y_u^*} = 0$. This means that all the values in $\boldsymbol{y_u^*}$ are the same (including the degenerate case where there are no values in $\boldsymbol{y_u^*}$). In this case we cannot expand the upper subset. Passing the $\sigma^2_{y_u^*} > 0$ hurdle, however, still leaves open the possibility that the procedure breaks down in Step 4 because $y^{*\prime}_{u_1} < \pi_{min}\mu$. This means that the smallest value in $\boldsymbol{y_u^*}$ gets mapped to a value below $\pi_{min}\mu$, thus violating the very constraint we wanted to satisfy by using split-linear rescaling. Working with real data we have encountered both of the breakdowns discussed, but we also have found that split-linear resampling succeeds in many cases where linear Pearson resampling fails.

2.4. Sampling from a limited and shifted Pareto distribution

We briefly discuss the justification for the method, before presenting the mathematical properties of the distribution.

2.4.1. Rationale for parametric resampling

The original idea of bootstrapping is that that the residuals can be interpreted as a sample from the error distribution, and that we can therefore use it to approximate the error distribution itself. In the context of development triangle GLMs, where we only have one data point for each combination of covariate levels, this assumption may be a little shaky. To see why bootstrapping seems to work anyway, we note that from a pseudo likelihood perspective (see chapter 9 in McCullagh and Nelder 1989), the fitted values of the GLM only depend on the assumed relationship between the fitted means and the corresponding expected variances. I.e., the specific shape of the error distribution beyond the second moment assumption does not affect the fitted parameter values, and we are dealing with a semi-parametric model for the data. The residual rescaling procedure ensures that for each data point we sample from a distribution with the correct mean-variance relationship. Viewed this way, residual bootstrapping is just one way to run a Monte Carlo simulation to determine the shape of the sampling distribution of the model parameters.

Unless we strongly believe that the observed residuals contain good information about the error structure (beyond a way of estimating the dispersion parameter), it seems equally valid to perform the Monte Carlo simulation using another resampling distribution that satisfies the mean-variance relationship, and is thus consistent with the semi-parametric model assumption for the data that the original model fit is based on. A more detailed discussion of the case for parametric resampling in the context of development triangle GLMs can be found on pages 16 and 17 of Hartl (2010). In this subsection we present one such resampling scheme that is efficient from a computational point of view and works well for small fitted values and large dispersion factors.

2.4.2. Properties of the limited and shifted Pareto distribution

We are only considering distributions with a Pareto index of one. The cumulative distribution function is given by

$$F(x)=\begin{cases}0 & x+c<a\\ 1-\dfrac{a}{x+c} & a\le x+c<b,\\ 1 & b\le x+c\end{cases} \qquad (2.9)$$

where $0<a<b$, and $c\in\mathbb{R}$. Note that this is a mixed distribution with $P(X=b-c)=a/b$. The following formulas for the expectation and variance are derived in A.4:

$$E(X)=a\left(1-\ln\frac{a}{b}\right)-c,$$
$$\mathrm{Var}(X)=2ab-a^2-a^2\left(1-\ln\frac{a}{b}\right)^2. \qquad (2.10)$$

This distribution has three parameters, which generally allows us to match the required mean and variance, while maintaining the minimum percentage of mean condition. Actually we employ two strategies.

First, we argue heuristically that it is desirable that a/b is small, but not too small. This ensures that the distribution is largely continuous, while there is also a reasonable chance that the maximum value is predictably attained during a 10,000 or 50,000 run simulation. By setting $a/b=0.001$ (any other small, but not too small value would do, as well) we end up with

$$\hat{y}=7.908a-c,\quad V(\hat{y})=1{,}936.467a^2, \qquad (2.11)$$

so the parameters *a, b, c* can readily be calculated in closed form (each fitted data point has its own set of parameters). Often the distribution obtained using this strategy will satisfy the minimum percentage of mean condition (i.e. $a-c\ge\pi_{min}\hat{y}$).

Second, if the first strategy fails to satisfy the minimum percentage of mean condition, we solve the following system of equations:

$$a-c=\pi_{min}\hat{y}$$
$$a\left(1-\ln\frac{a}{b}\right)-c=\hat{y}$$
$$2ab-a^2-a^2\left(1-\ln\frac{a}{b}\right)^2=V(\hat{y}). \qquad (2.12)$$

It is not possible to solve this non-linear system of equations in closed form, but it can be reduced to the following equation with one unknown

$$1+\gamma+\frac{1+k}{2}\gamma^2-e^{\gamma}=0, \qquad (2.13)$$

where $k=V(\hat{y})(1-\pi_{min})^{-2}\hat{y}^{-2}$. A derivation of equation (2.13) is provided in A.5.

In A.6 we show that equation (2.13) has a unique positive solution for all $k>0$. For a brief discussion on how to find the solution the reader is referred to A.7. While it is always possible to use parametric resampling from a limited and shifted Pareto distribution, we caution that the distribution may become nearly degenerate for small values of k ($P(X=b-c)$ goes to 1) or nearly unlimited for large values of k ($b-c$ becomes large, and $P(X=b-c)$ goes to 0).

3. Relative performance

When considering the code needed to implement a bootstrapping scheme, one can distinguish three phases: initialization, Monte Carlo phase, and processing of output. During initialization we perform the initial model fit and create the data structures needed to efficiently execute the resampling and refitting during the Monte Carlo phase. Typically we perform a bootstrapping simulation to get a robust measure of the variance or a tail index such as TVar. So, as a general rule, we want to run as many Monte Carlo iterations as possible (given time and memory constraints). For this reason the time spent during initialization and processing of output is small compared to the time spent during the Monte Carlo phase. Therefore we restrict our-

selves to analyzing performance during the Monte Carlo phase.

3.1. Pseudo code for resampling step

For all our methods we assume that the resampling parameters were pre-computed and stored during the initialization phase. Furthermore we assume that there are m data points and r standardized residuals.

The memory requirements for linear Pearson residual resampling are

```
Dim Residual(1 To r) As Double
Dim Base(1 To n) As Double
Dim Scale(1 To n) As Double
```

The pseudo code for resampling the *i*th data point is

```
Function ResampleLinearPearson
  (i As Integer) As Double
Dim j As Integer
j = 1 + int(RandUniform(0,1) * r)
  ' RandInteger(1,r)
Result = Base(i) + Scale(i) * Residual(j)
End Function                                (3.1)
```

The memory requirements for split-linear residual resampling are

```
Dim Residual(1 To r) As Double
Dim FirstUp(1 To n) As Integer
Dim BaseLow(1 To n) As Double
Dim BaseUp(1 To n) As Double
Dim ScaleLow(1 To n) As Double
Dim ScaleUp(1 To n) As Double
```

The pseudo code for resampling the *i*th data point is

```
Function ResampleSplitLinear
  (i As Integer) As Double
Dim j As Integer
j = 1 + int(RandUniform(0,1) * r)
  ' RandInteger(1,r)
If j < FirstUp(i)
  Result = BaseLow(i) + ScaleLow(i)
  * Residual(j)
Else
  Result = BaseUp(i) + ScaleUp(i)
  * Residual(j)
End If
End Function                                (3.2)
```

Note that by precomputing BaseLow, ScaleLow, BaseUp, and ScaleUp we are effectively combining the original linear Pearson rescaling with the split-linear transformation (2.6) into one linear transformation (technically one transformation for the lower set and one for the upper set). Details can be found in A.8. We observe that only one line of code from the If . . . then . . . else block will actually be executed for any given data point, so the executed operations for split-linear resampling are almost the same as linear Pearson resampling, except for the comparison and branching operation needed for the If . . . then . . . else block.

The memory requirements for limited Pareto sampling are

```
Dim a(1 To n) As Double
Dim bmc(1 To n) As Double  ' bmc = b minus c
Dim c(1 To n) As Double
Dim p(1 To n) As Double
```

The pseudo code for resampling the *i*th data point is

```
Function ResampleLimitedPareto(i As Integer)
  As Double
Dim u As Double
u = RandUniform(0,1)
If u <= p(i) Then
  Result = bmc(i)
Else
  Result = a(i)/u - c(i)
End If
End Function                                (3.3)
```

This formula is simply the inverse transform method applied to the CDF given by (2.9).

Note that a typical random number generator will produce a uniform random number between 0 and 1. Generating a random integer (between given bounds) is usually accomplished as indicated in the pseudo code above, even if a particular implementation may hide this from the user by providing a RandInteger function. For our operation counts in Table 1 we will exclude the addition in RandInteger, because this can be avoided by using 0-based instead of 1-based arrays.

3.2. Performance comparisons

Table 1 summarizes the memory requirements and provides operation counts for each of the three methods. Please note that in the case of limited Pareto sampling, the operation counts depend on the random number generated, and therefore two columns of operation counts are provided. With probability $(1 - p)$ the

Table 1. Memory requirement and operation counts

	Linear Pearson	Split-Linear	Limited Pareto	
Memory	$(2n + r) \times$ dbl	$(4n + r) \times$ dbl + $n \times$ int	$4n \times$ dbl	
array access	3	4	3	2
RndUni()	1	1	1	1
int()	1	1	–	–
dbl + or –	1	1	1	–
dbl * or ÷	2	2	1	–
comparison	–	1	1	1

first column applies, and with probability p, the second column applies. The "int()" row refers to the truncation of the fractional part used in RandInteger; the "comparison" row refers the If . . . then . . . else block. For the memory row, dbl refers to a 64 bit floating point number, and int refers to a 32 bit integer (some systems may allocate 64 bits for an integer).

The computational cost for the various operations does depend on the programming language and platform used. However, we can unambiguously conclude that split-linear rescaling requires slightly more operations (one additional array access, and one comparison) than linear Pearson rescaling. Comparing linear Pearson rescaling to limited Pareto sampling, we note that linear Pearson rescaling requires an additional multiplication operation, while limited Pareto sampling involves an additional comparison. The complexity of the comparison is similar to a subtraction operation, so we expect that on most platforms, limited Pareto sampling will perform slightly better than linear Pearson rescaling. Limited Pareto rescaling additionally benefits from the reduced complexity in case we sample the upper limit.

To provide some concrete examples, we have tested the methods on two actuarial development triangles. The first data set (A) is a US Industry Auto paid loss triangle taken from Friedland (2010, p. 107). This triangle was chosen because all three methods work, allowing us to benchmark them against each other. The second data set (B) is taken from the appendix to Taylor and Ashe (1983) and has been used by a number of authors to demonstrate stochastic reserving models. Data set (B) is presented in the triangle form familiar to P&C actuaries in Pinheiro, Silva, and Centeno (2003). Both triangles are included in appendix B for reference purposes. Linear Pearson rescaling fails for this second triangle (see Hartl 2010 for a detailed discussion). We are using it to demonstrate that the two alternative methods proposed in this paper do extend the applicability of the bootstrapping approach to triangle GLMs. To further demonstrate the flexibility of the GLM approach, the second model is fitted to the last five diagonals of the triangle (instead of the full triangle). Note that for data set B we only have 40 data points vs 55 data points for data set A (because we are only using the most recent five diagonals). In each case we are also sampling 45 future cells to simulate the process error component.

All the test results cited in this paper were produced with a VBA for Excel application that is available from the author on request. Because VBA for Excel is an interpreted language that is embedded in a complex environment, reliably testing the performance is not straightforward. Since we cannot control for the variation in the environment, we have used a randomized sampling scheme, where we executed a fixed number of test runs and randomly chose which particular method to use for each test run. We ran two different test scripts for each triangle. The first test script (sampling) tested the resampling operation in isolation. Given our total number of test runs, we expect about 3,333 runs for each method, where each test run consists of resampling the triangle and simulating the reserve runoff 100 times. The second test script (full) tested the resampling methods in the context of the full bootstrapping simulation (i.e., including refitting the model, projecting the reserve based on the refitted model, and simulating the runoff of the reserve). Again, we expect about 3,333 test runs for each method, with 100 bootstrapping iterations each. The scripts were run on the author's Lenovo X230 tablet (Intel Core i5, 2.6GHz, 4GB ram), under Windows 7(64 bit), with Excel 2010.

Table 2 summarizes the average run time and the standard deviation for 100 iterations, for each data

Table 2. Performance statistics

Run time (seconds) per 100 iterations (sample size ≈ 3,333)						
	Linear Pearson		Split-Linear		Limited Pareto	
Data – Script	Avg	StDev	Avg	StDev	Avg	StDev
A – sampling	.035865	1.4732e-5	.036634	1.4646e-5	.034683	1.4305e-5
A – full	.28042	8.1971e-5	.28123	8.0853e-5	.24674	8.0390e-5
B – sampling	n/a	n/a	.030226	1.3084e-5	.028707	1.2344e-5
B – full	n/a	n/a	.26993	8.1661e-5	.22490	7.7293e-5

set and test script. All run times are in seconds and pertain only to the Monte Carlo phase.

The VBA for Excel based performance comparison confirms the relative performance we expected based on our operation counts in Table 1.

4. Conclusions

We presented two new methods for generating pseudo data for bootstrapping a GLM. Split-linear rescaling is a residual based resampling technique that extends the class of triangle GLMs that can be bootstrapped, but it can fail for specific triangles. Sampling from a limited and shifted Pareto distribution is always possible (subject to practical limitations when the distribution becomes almost degenerate or effectively unlimited). Both methods are computationally inexpensive and comparable to linear Pearson residual rescaling. Further research is needed to explore the sensitivity of bootstrapped confidence intervals or tail measures to the resampling distribution employed.

Acknowledgments

This paper was written and revised while the author worked as an Assistant Professor at Bryant University, RI. The author is also grateful for the constructive comments from the anonymous reviewers.

Bibliography

Davison, A. C., and D. V. Hinkley, *Bootstrap Methods and their Application*, New York: Cambridge University Press, 1997.

England, P. D., and R. J. Verrall, "Stochastic Claims Reserving in General Insurance," *British Actuarial Journal* 8: 3, 2002, pp. 443–518.

Gluck, S. M., and G. G. Venter, "Stochastic Trend Models in Casualty and Life Insurance," in *Enterprise Risk Management Symposium*, 2009.

Friedland, J. F., *Estimating Unpaid Claims Using Basic Techniques*, 3rd ed., Arlington, Va.: Casualty Actuarial Society, 2010.

Hartl, T., "Bootstrapping Generalized Linear Models for Development Triangles Using Deviance Residuals," CAS *E-Forum*, Fall 2010.

Hartl, T., "GLMs for Incomplete Development Triangles," presented at the Casualty Loss Reserving Seminar, 2013.

McCullagh, P., and J. A. Nelder, *Generalized Linear Models*, 2nd ed., New York: Chapman and Hall, 1989.

Pinheiro, P. J. R., J. M. A. Silva, and M. D. L. Centeno, "Bootstrap Methodology in Claim Reserving," *Journal of Risk and Insurance* 70: 4, Dec. 2003, pp. 701–714.

Taylor, G. C., and F. R. Ashe, "Second Moments of Estimates of Outstanding Claims," *Journal of Econometrics* 23: 1, 1983, pp. 37–61.

Appendix A

A.1. Derivation of equation (2.7):

The variance of $\boldsymbol{y}^{*\prime}$ is given by

$$\sigma^2_{y^{*\prime}} = \frac{1}{m}\sum (y^{*\prime} - \mu)^2 .$$

This can be split into separate summations for the lower and upper subset. After substituting the expressions from equation (2.6) we get

$$\sigma^2_{y^{*\prime}} = \frac{1}{m}\sum (\mu_l + c_l (y^*_l - \mu_l) - \mu)^2 + \frac{1}{m}\sum (\mu_u + c_u (y^*_u - \mu_u) - \mu)^2. \quad \text{(A.1)}$$

Note that the first summation ranges over the q elements of $\boldsymbol{y}_l^*$ and the second summation ranges over the r elements of $\boldsymbol{y}_u^*$. For the time being, we concentrate on simplifying the summation term for the lower subset. We start by grouping all the constant terms together, continue by expanding the square, and then simplify

$$\begin{aligned}
&\sum (c_l y_l^* + (\mu_l - c_l\mu_l - \mu))^2 \\
&= \sum \left((c_l y_l^*)^2 + 2c_l y_l^* (\mu_l - c_l\mu_l - \mu) + (\mu_l - c_l\mu_l - \mu)^2 \right) \\
&= \left(c_l^2 \sum y_l^{*2} \right) + q\mu_l (2c_l (\mu_l - c_l\mu_l - \mu)) \\
&\quad + q(\mu_l - c_l\mu_l - \mu)^2 \\
&= q(\mu_l - c_l\mu_l - \mu)(2c_l\mu_l + \mu_l - c_l\mu_l - \mu) + c_l^2 \sum y_l^{*2} \\
&= q((\mu_l - \mu) - c_l\mu_l)((\mu_l - \mu) + c_l\mu_l) + c_l^2 \sum y_l^{*2} \\
&= q(\mu_l - \mu)^2 - qc_l^2\mu_l^2 + c_l^2 \sum y_l^{*2} \\
&= q(\mu_l - \mu)^2 + c_l^2 \left(\sum y_l^{*2} - \sum \mu_l^2 \right) \\
&= q(\mu_l - \mu)^2 + qc_l^2\sigma_{y_l^*}^2.
\end{aligned}$$

An analogous derivation shows that

$$\sum \left(c_u y_u^* + (\mu_u - c_u\mu_u - u) \right)^2 = r(\mu_u - \mu)^2 + rc_u^2\sigma_{y_l^*}^2.$$

Substituting these last two expressions into equation (A.1) we end up with

$$\sigma_{y^{*\prime}}^2 = \frac{1}{m}\left(q(\mu_l - \mu)^2 + qc_l^2\sigma_{y_l^*}^2 + r(\mu_u - \mu)^2 + rc_u^2\sigma_{y_u^*}^2 \right),$$

which completes the derivation of equation (2.7) ■

A.2. Derivation of equation for step 2 of (2.8):

To guarantee the minimum percentage of mean condition we require

$$y_{l_1}^{*\prime} = \mu_l + c_l \left(y_{l_1}^* - \mu_l \right) = \pi_{min}\mu.$$

Solving for c_l we get

$$c_l = \frac{\pi_{min}\mu - \mu_l}{y_{l_1}^* - \mu_l} = \frac{\mu_l - \pi_{min}\mu}{\mu_l - y_{l_1}^*},$$

which completes the derivation of the equation for step 2 ■

A.3. Derivation of equation for step 3 of (2.8):

Note that by letting $c_l = c_u = 1$ transformation (2.6) maps $\boldsymbol{y}^*$ onto itself. As a special case of equation (2.7) we therefore get

$$\sigma_{y^*}^2 = \frac{1}{m}\left(q(\mu_l - \mu)^2 + q\sigma_{y_l^*}^2 + r(\mu_u - \mu)^2 + r\sigma_{y_u^*}^2 \right).$$

Requiring the general transformation (2.6) to be variance preserving (i.e., $\sigma_{y^{*\prime}}^2 = \sigma_{y^*}^2$), thus leads to

$$\begin{aligned}
&\frac{1}{m}\left(q(\mu_l - \mu)^2 + qc_l^2\sigma_{y_l^*}^2 + r(\mu_u - \mu)^2 + rc_u^2\sigma_{y_u^*}^2 \right) \\
&\qquad = \frac{1}{m}\left(q(\mu_l - \mu)^2 + q\sigma_{y_l^*}^2 + r(\mu_u - \mu)^2 + r\sigma_{y_u^*}^2 \right).
\end{aligned}$$

This expression can readily be simplified to yield

$$qc_l^2\sigma_{y_l^*}^2 + rc_u^2\sigma_{y_u^*}^2 = q\sigma_{y_l^*}^2 + r\sigma_{y_u^*}^2.$$

Solving for c_u we end up with

$$c_u = \sqrt{1 + (1 - c_l^2)\frac{q\sigma_{y_l^*}^2}{r\sigma_{y_u^*}^2}},$$

completing the derivation of the equation for step 3 ■

A.4. Derivation of equation (2.10)

The support of the limited and shifted Pareto random variable defined by (2.9) is $(a - c, b - c]$, with

$$f(x) = \frac{a}{(x + c)^2} \text{ for } x \in (a - c, b - c),$$

$$P(X = b - c) = \frac{a}{b}.$$

The derivation of the formulas for the expected value can be simplified by considering the random variable $Y = X + c$ (i.e., by reversing the shift). We obtain

$$f(y) = \frac{a}{y^2} \text{ for } y \in (a, b), \quad P(Y = b) = \frac{a}{b}.$$

For the expected value of Y we have

$$\begin{aligned}\mathrm{E}(Y) &= \int_a^b y\,\mathrm{f}(y)\,dy + b\,\mathrm{P}(Y=b)\\ &= \int_a^b \frac{ay}{y^2}\,dy + b\frac{a}{b} = a\int_a^b \frac{1}{y}\,dy + a\\ &= a\ln b - a\ln a + a = a\left(1-\ln\frac{a}{b}\right).\end{aligned}$$

For the second raw moment of Y we get

$$\begin{aligned}\mathrm{E}\left(Y^2\right) &= \int_a^b y^2\,\mathrm{f}(y)\,dy + b^2\,\mathrm{P}(Y=b)\\ &= \int_a^b \frac{ay^2}{y^2}\,dy + b^2\frac{a}{b} = \int_a^b a\,dy + ab\\ &= a(b-a)+ab = 2ab-a^2.\end{aligned}$$

From this we readily obtain the formula for the variance of Y:

$$\begin{aligned}\mathrm{Var}(Y) &= \mathrm{E}\left(Y^2\right) - \mathrm{E}(Y)\\ &= 2ab - a^2 - a^2\left(1-\ln\frac{a}{b}\right)^2.\end{aligned}$$

Noting that $X = Y - c$ implies $\mathrm{E}(X) = \mathrm{E}(Y) - c$, and $\mathrm{Var}(X) = \mathrm{Var}(Y)$, we can see that the following formulas hold:

$$\mathrm{E}(X) = a\left(1-\ln\frac{a}{b}\right) - c,$$

$$\mathrm{Var}(X) = 2ab - a^2 - a^2\left(1-\ln\frac{a}{b}\right)^2.$$

This completes the derivation of equation (2.10) ■

A.5. Derivation of equation (2.13)

For convenience we repeat the system of equations (2.12)

$$a - c = \pi_{min}\hat{y}$$

$$a\left(1-ln\frac{a}{b}\right) - c = \hat{y}$$

$$2ab - a^2 - a^2\left(1-ln\frac{a}{b}\right)^2 = V(\hat{y})$$

We can eliminate the first equation by setting $c = a - \pi_{min}\hat{y}$. Substituting $U = (1-\pi_{min})\hat{y}$, $p = a/b$, and $V = \mathrm{V}(\hat{y})$ leaves us with

$$a(1-\ln p) = a + U$$

$$\frac{2a^2}{p} - a^2 - a^2(1-\ln p)^2 = V.$$

Squaring the first equation and substituting into the second, we can eliminate the second equation by setting

$$\frac{1}{p} = \frac{a^2 + (a+U)^2 + V}{2a^2} = 1 + \frac{U}{a} + \frac{U^2}{2a^2} + \frac{V}{2a^2},$$

where the second expression is obtained by expanding $(a + U)^2$ and then dividing all terms by $2a^2$. This leaves us with only one remaining equation to satisfy

$$a - a\ln p = a + U.$$

Note that at this point p is considered a function of a. We eliminate the additive a term, apply the exponential to both sides, and finally collect all terms on the left to get the equivalent equation

$$\frac{1}{p} - e^{U/a} = 0.$$

Substituting the above expression for $1/p$, $\gamma = U/a$, and $k = V/U^2$ we finally arrive at

$$1 + \gamma + \frac{1+k}{2}\gamma^2 - e^{\gamma} = 0.$$

Note that since U, $a > 0$, we are only interested in solutions with $\gamma > 0$. Also note that $k > 0$. This completes the derivation of equation (2.13) ■

A.6. Existence of a unique solution to equation (2.13)

Our analysis is aided by thinking of the LHS of equation (2.13) as defining a function $G(\gamma)$

$$\mathrm{G}(\gamma) = 1 + \gamma + \frac{1+k}{2}\gamma^2 - e^{\gamma}.$$

We also need the first derivative

$$G'(\gamma) = 1 + (1+k)\gamma - e^{\gamma},$$

and the second derivative

$$G''(\gamma) = 1 + k - e^{\gamma}.$$

We note the boundary conditions of $G(0) = G'(0) = 0$, while $G''(0) = k > 0$.

Clearly $G''(\ln(1+k)) = 0$, $G''(\gamma) > 0$ for $0 < \gamma < \ln(1+k)$, and $G''(\gamma) < 0$ for $\gamma > \ln(1+k)$. Given that $G'(0) = 0$, we can conclude that there exists a unique value $\gamma_{max} > \ln(1+k)$ such that $G'(\gamma_{max}) = 0$, $G'(\gamma) > 0$ for $0 < \gamma < \gamma_{max}$, and $G'(\gamma) < 0$ for $\gamma > \gamma_{max}$. Since we also have $G(0) = 0$, the same argument also implies that there exists a unique value $\gamma_{root} > \gamma_{max}$ such that $G(\gamma_{root}) = 0$, $G(\gamma) > 0$ for $0 < \gamma < \gamma_{root}$, and $G(\gamma) < 0$ for $\gamma > \gamma_{root}$.

This establishes that γ_{root} is the unique solution to equation (2.13) ■

A.7. Remarks on numerically solving equation (2.13)

Using the definitions from Appendix A.6, we can apply the Newton-Raphson method to solve equation (2.13) using the iteration

$$\gamma_{i+1} = \gamma_i - \frac{G(\gamma_i)}{G'(\gamma_i)}.$$

Without providing the details, we comment that the above iteration is guaranteed to converge to γ_{root}, provided we can find an initial guess such that $\gamma_0 > \gamma_{max}$. Based on our analysis in Appendix A.6, we know that $\gamma_{root} > \gamma_{max} > \ln(1+k)$ and that $G'(\gamma) < 0$ if and only if $\gamma > \gamma_{max}$. So a brute force strategy for finding a suitable γ_0, is the following procedure (pseudo code):

Step 0 $\gamma_0 = \ln(1+k)$ ′ Initialize γ_0
Step 1 $\gamma_0 = 2\gamma_0$ ′ Double γ_0
Step 2 If $G'(\gamma_0) \geq 0$ ′ Repeat until $\gamma_0 > \gamma_{max}$
then go back to Step 1.

In practice the basic Newton-Raphson algorithm described here may be modified to increase numerical stability or the speed of convergence. Also note that the particular form of equation (2.13) has been chosen to streamline the proof of the existence and uniqueness of the solution to the system of equations (2.12). Numerical methods for solving (2.12) may also work with other equivalent representations (e.g., directly solving for a) ■

A.8. Derivation of the formula for pseudo code (3.2)

The linear Pearson rescaling transformation for a GLM with variance function $V(\hat{y})$ and dispersion factor ϕ is given by

$$y^* = \hat{y} + \sqrt{\phi V(\hat{y})} \cdot \mathrm{Rnd}(\boldsymbol{s}),$$

while the split-linear transformation (2.6) is given by

$$y^{*\prime} = \begin{cases} \mu_l + c_l(y^* - \mu_l), \text{if } y^* \in \boldsymbol{y}_l^* \\ \mu_u + c_u(y^* - \mu_u), \text{if } y^* \in \boldsymbol{y}_u^* \end{cases}.$$

Without loss of generality we can assume that the elements of $\boldsymbol{s}$ have been indexed in ascending order, starting with index 1 for the smallest (i.e., most negative) residual. As before, we assume that $\boldsymbol{y}_l^*$ and $\boldsymbol{y}_u^*$, have q and r data values, respectively. We can combine both transformations into one transformation using the following definition:

$$y_l^{*\prime} = \begin{cases} \mathrm{base}_l + \mathrm{scale}_l s_l, \text{ if } i \leq q \\ \mathrm{base}_u + \mathrm{scale}_u s_l, \text{ if } i > q \end{cases}.$$

The reader can easily verify that the parameters for this version are given by the following formulas:

$$\mathrm{base}_l = (1 - c_l)\mu_l + c_l\hat{y}$$

$$\mathrm{scale}_l = c_l\sqrt{\phi V(\hat{y})}$$

$$\mathrm{base}_u = (1 - c_u)\mu_u + c_u\hat{y}$$

$$\mathrm{scale}_u = c_u\sqrt{\phi V(\hat{y})}.$$

This completes the derivation of the formula used in pseudo code (3.2) ■

Appendix B

For reference we provide the data used to test the methods in section 3. Both data sets are given in incremental format, organized as development triangles with rows corresponding to accident years and columns corresponding to development years.

B.1. Triangle A

This data set is taken from Friedland (2010, p. 107). We divided the amounts by 1,000.

18,539	14,692	6,831	3,830	2,004	869	456	226	109	89
20,410	15,680	7,169	3,900	2,049	954	464	253	122	
22,121	16,855	7,413	4,173	2,173	1,005	544	249		
22,992	17,104	7,672	4,326	2,270	1,015	500			
24,093	17,703	8,108	4,449	2,401	1,053				
24,084	17,315	7,671	4,514	2,346					
24,370	17,120	7,747	4,538						
25,101	17,602	7,943							
25,609	17,998								
27,230									

B.2. Triangle B

This data set is taken from the appendix of Taylor and Ashe (1983). Note that the original data set is provided in claim count and severity format. We have rounded total paid claim amounts to the nearest integer.

357,848	766,940	610,542	482,940	527,326	574,398	146,342	139,950	227,229	67,948
352,118	884,021	933,894	1,183,289	445,745	320,996	527,804	266,172	425,046	
290,507	1,001,799	926,219	1,016,654	750,816	146,923	495,992	280,405		
310,608	1,108,250	776,189	1,562,400	272,482	352,053	206,286			
443,160	693,190	991,983	769,488	504,851	470,639				
396,132	937,085	847,498	805,037	705,960					
440,832	847,631	1,131,398	1,063,269						
359,480	1,061,648	1,443,370							
376,686	986,608								
344,014									