Fitting a GLM to Incomplete Development Triangles

Thomas Hartl, ACAS

Abstract: When fitting a generalized liner model (GLM) to a development triangle is discussed in the existing actuarial literature, reference is usually made to statistical packages for accomplishing this task. This paper presents a practical discussion of how to use Visual Basic to fit a GLM to a triangle with special emphasis on how to deal with incomplete data. Interested readers can contact the author to request a copy of an MS Excel application that implements the algorithms discussed in this paper. The application of GLMs to incomplete development triangles is motivated by translating judgments of practicing actuaries (e.g., use last *n* diagonals) into a rigorous regression framework. The key original contribution of this paper is the discussion of how graph theory can be used to analyze the topology of an arbitrary selection of triangle cells, and how to use the information gained to set up a regression model that is suitable for projecting future development. Once properly specified, fitting a GLM using maximum likelihood estimation (MLE) is straight forward, and we describe how this can be accomplished from a practical point of view in Visual Basic. To round off our discussion of model fitting, we briefly describe the standardization of residuals, and how to plot them for graphically evaluating goodness of fit. Finally we briefly discuss how the described class of GLMs for development triangles compares to some other stochastic models proposed in the actuarial literature.

Keywords. Generalized Linear Modeling, Reserving Methods, Regression, Data Diagnostics, Data Visualization, Bootstrapping, and Resampling Methods.

1. INTRODUCTION

In the context of stochastic reserving, several authors (e.g., [4], [7], and [9]) have stressed the need of casting the task of projecting reserves in a rigorous way as a regression problem. These authors have also pointed out that performing an all years volume weighted link-ratio estimate leads to the same result as fitting a GLM with the logarithmic link function and the identity variance function. Many practitioners have exploited this equivalence to implement spreadsheet-based applications for deriving a distribution of possible reserve outcomes based on bootstrap simulations by repeated resampling and application of the link-ratio estimate. While suitable to illustrate the concept of bootstrapping, these applications are typically not flexible enough to deal with practical judgments reserving analysts have to make about which cells of the triangle are deemed to be representative of future development (e.g., use data from last *n* diagonals or exclude obvious abnormalities). At other times practicing actuaries are also faced with data that are simply incomplete to start with. The important question here is how incomplete can a triangle ultimately be, while still providing information that is useful for the purpose of projecting future development? The key result presented in this paper is that concepts from an area of mathematics know as graph theory can be used to answer this question. Once we have analyzed some key aspects of the graph topology of a set of triangle cells, we can easily set up a well-defined regression problem and gain further insights into what information about the variability of the underlying stochastic process can be gained from the resulting model.

1.1 Research Context

Several papers (e.g., [4], [7], and [9]) in the actuarial literature do describe in abstract terms how to apply GLM theory to fitting a model to an actuarial development triangle. The actual algorithm for fitting a GLM follows the description in McCullagh and Nelder's classic *Generalized Linear Models* (2nd Edition) [6]. When fitting a GLM to an incomplete development triangle, however, the question of what constitutes a valid regression model specification naturally arises. We discuss how to use graph theory to algorithmically deal with this issue. By fitting a GLM to incomplete development triangles we furthermore extend the scope of traditional triangle-based reserving techniques: often a reserve projection can be made even if we only have partial information about the past development history. While this paper deals with the fitting of a regression model, the graph topology of the selected set of triangle cells also determines what information about the variability of the underlying stochastic process can be gleaned from the data. As it turns out, even when a data set supports projections for all development periods, different regions of an incomplete triangle may split into areas that are effectively fit without any influence from other areas. So, we can have multiple weakly connected regression models, rather than one comprehensive model for all selected data points. We use some of this information in our description of how to standardize residuals and how to plot them for diagnostic purposes. This insight also has implications for the scope and applicability of bootstrapping methods.

1.2 Objective

The iterative weighted LSQ algorithm for fitting a GLM is described in [6] and [9], but these textbooks generally assume that the reader is already familiar with the algorithms for performing regression fits. This paper seeks to explain at a practical level how to fit a GLM to a triangle of incremental development amounts. In particular, we address the issue of what happens if we either do not have complete information about the development history or want to exercise actuarial judgment about what data to include in our model. In addition, while there may be many advantages to using a fully fledged statistical package, we hope that interested readers who contact the author to request a copy of the companion MS Excel application will be able to explore the issues discussed and thus deepen their understanding of the process of fitting a regression model to a development triangle.

1.3 Outline

The remainder of the paper proceeds as follows. The second section starts with a visual description of the structure of a regression model for an incomplete development triangle. We then briefly discuss how we can link the discussed features of the model structure to aspects of the graph topology of an incomplete development triangle. Next we introduce an algorithm known as "breadth first search" which can be used to identify the situations previously described. We conclude by indicating how the information gathered can be used to specify the regression problem and deal with data points that require special attention. We believe that the application of graph theory to specifying a regression problem has not been previously discussed in the actuarial literature. The third section provides an overview of how to use Visual Basic to implement a maximum likelihood estimator based on iterative weighted least squares. The core algorithm here follows standard textbook treatment, but we make use of the graph topology of the incomplete development triangle to piece together the overall regression model from its subcomponents, if applicable. The fourth section deals with the standardization of residuals and plotting them for graphically evaluating goodness of fit. This section also walks through a number of the diagnostic exhibits using a concrete data set to demonstrate how an analyst may use them in practice. Finally we briefly discuss how the class of GLMs described in this paper compares to some other stochastic models for development triangles that are discussed in the actuarial literature.

2 SETTING UP THE MODEL SPECIFICATION

In this section we go into the details of how to set up the model specification that formally describes the regression problem corresponding to a multiplicative model for an incremental development triangle with separate parameters for rows and columns.

2.1 Notes on the structure of the regression model

To visualize what our set-up algorithm is trying to accomplish, we use the example of a five-by-five triangle. We have dispensed with row or column labels to reduce clutter. We follow the convention that rows denote exposure periods and columns development periods. With this said, a multiplicative model for expected incremental amounts looks something like this:

$$\begin{array}{ccccc} c & b_2c & b_3c & b_4c & b_5c \\ a_2c & a_2b_2c & a_2b_3c & a_2b_4c & \\ a_3c & a_3b_2c & a_2b_3c & & \\ a_4c & a_4b_2c & & & \\ a_5c & & & & \end{array} \tag{2.1}$$

This parameterization corresponds to a common method for dealing with extrinsic aliasing for factorial models: drop one level from each factor and replace them by one offset parameter common to all observations. For (2.1) we have dropped the first exposure and the first development period parameter and replaced them with an offset parameter. In this parameterization the offset parameter *c* denotes the value of a base (or reference) cell and the a_i and b_j parameters are relativities for exposure and development periods, respectively. Also note that the choice of reference cell generally does not affect the fitted values produced by the model. We could have equally chosen the following parameterization:

$$\begin{array}{ccccc} a_1b_1c & a_1b_2c & a_1c & a_1b_4c & a_1b_5c \\ a_2b_1c & a_2b_2c & a_2c & a_2b_4c & \\ b_1c & b_2c & c & & \\ a_4b_1c & a_4b_2c & & & \\ a_5b_1c & & & & \end{array} \tag{2.2}$$

For a complete triangle either of the above parameterizations can straightforwardly be translated into a standard regression problem and the fitted incremental amounts will be identical. When we start excluding data points from the analysis, we may encounter a number of issues that force us to pay closer attention the structure and parameterization of our regression model. The algorithm presented here deals with four specific issues relating to ensuring we are dealing with a well-defined regression problem, and to identifying triangle cells requiring special treatment in our subsequent goodness of fit analysis.

2.1.1 Not enough data points to estimate some parameters

$$\begin{array}{ccccc} c & b_2c & b_3c & \times & b_5c \\ a_2c & a_2b_2c & a_2b_3c & \times & \\ a_3c & a_3b_2c & a_2b_3c & & \\ a_4c & a_4b_2c & & & \\ a_5c & & & & \end{array} \tag{2.3}$$

The cross symbol here denotes data points that are missing or excluded by the analyst (e.g., truncated triangles or want to use last *n* diagonals). Clearly we have no information on the b_4 parameter and it therefore has to be dropped from the model. Note that despite the "gap" at development period 4, there is no issue with relating the top right corner (development period 5) with the rest of the triangle—we can compare this value to the first 3 incremental values for exposure period 1.

2.1.2 Choice of reference cell does matters after all

Assume we are trying to use our cell (3,1) as our reference cell for the following data set:

$$\begin{array}{ccccc} a_1b_1c & a_1b_2c & a_1c & a_1b_4c & a_1b_5c \\ a_2b_1c & a_2b_2c & a_2c & a_2b_4c & \\ \times & \times & \times & & \\ a_4b_1c & a_4b_2c & & & \\ a_5b_1c & & & & \end{array} \tag{2.4}$$

Clearly we have no information on a_3, and this situation could be remedied by dropping this parameter from the model. In this case, however, we cannot do this because a_3 has already been replaced by the common offset parameter *c*. Actually, our algorithm circumvents this problem altogether by first analyzing which rows and columns are part of the connected component of triangle cells for which fit a model before attempting to assign parameters to rows and columns.

2.1.3 Data splits into unrelated regions

$$\begin{array}{ccccc}
\times & \times & b_3c & b_4c & b_5c \\
\times & \times & a_2b_3c & a_2b_4c & \\
\times & \times & a_2b_3c & & \\
a_4c & a_4b_2c & & & \\
a_5c & & & &
\end{array} \tag{2.5}$$

In this situation, there is no information on how to relate the upper right sub-triangle to the lower triangle. This is an issue that cannot be fixed in a meaningful way, as far as predicting future values for all exposure and development periods is concerned. We handle this issue by using graph theory, noting that two triangle cells can be regarded as connected if they are either in the same row or in the same column. We determine what is called the maximal connected components of the triangle viewed as a graph. Further information on graph theory will be provided in section 2.2, below. One could fit a separate regression model for each of the connected components, but this is not useful for projecting future development amounts. Generally we hope that there is only one connected component. If not, we continue with the connected component that has the maximum number of triangle cells. If the number of triangle cells does not uniquely determine which component to pick, we take the component with the left-most column.

2.1.4 Exact fit cells

$$\begin{array}{ccccc}
\times & \times & b_3c & b_4c & \underline{b_5c} \\
\times & \times & a_2b_3c & a_2b_4c & \\
a_3c & a_3b_2c & \underline{\underline{a_3b_3c}} & & \\
a_4c & a_4b_2c & & & \\
\underline{a_5c} & & & &
\end{array} \tag{2.6}$$

One of the general goals in stochastic model fitting is to assess goodness-of-fit and measure the variability inherent in the observed process. To this end residuals (actual value less fitted value) need to be analyzed. This analysis can be distorted by triangle cells where the fitted value will always be exactly the same as the actual value. For a complete triangle this will always be the case for the top right and the bottom left corner, but when there are missing data points the same may be true for other cells. In the above example parameters a_5 and b_5 each appear in exactly one triangle cell, so they will always take values that ensure a perfect fit. What may be less obvious is that there will always be an exact fit for the cell in row 3, column 3. The reason for this is that removing this cell would split the incomplete triangle into two unrelated regions (as discussed under 2.1.2 above). Our algorithm for analyzing the model structure identifies exact fit cells by looping over all cells in the selected connected component and checking for each cell whether the removing the cell from the model changes to model structure by either dropping a row or column, or by splitting the model into two unrelated regions.

2.1.5 Further remarks on the model structure

Until now our discussion on the model structure preserved the shape of the triangle because the distinction between exposure and development periods is meaningful to us as P&C actuaries. Algorithms for fitting a simple GLM model as described above, however, are indifferent to how we perceive the various triangle cells as data points that are somehow ordered by exposure and development periods. Consider the following sparse data set for a hypothetical ratemaking problem with a multiplicative model with two classification dimensions, namely group and territory:

$$\begin{array}{ccccc} t_1g_1b & t_1g_2b & b & t_1g_4b & t_1g_5b \\ \times & \times & t_2b & \times & \times \\ \times & t_3g_2b & t_3b & \times & t_3g_5b \\ \times & \times & t_4b & \times & t_4g_5b \\ t_5g_1b & t_5g_2b & t_5b & \times & t_5g_5b \end{array} \tag{2.7}$$

Group 3 in territory 1 corresponds to the base rate *b*, while the g_i and t_i parameters represent group and territory relativities. This may not look like a development triangle, but this data set is structurally identical to multiplicative model for a complete triangle of incremental development amounts (see our original example at the beginning of section 2).

Hence, when using a model based on distinct parameters for each exposure and development period, a triangle is just an unordered list of data points, and the only relationships between data points are defined by certain parameters simultaneously affecting the fitted values of multiple triangle cells. For the purpose of implementing a concrete algorithm, however, we do need to find a way of listing all triangle cells. To keep things simple our default is to loop over rows, then columns, resulting in the following order of processing cells:

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & \times & 8 & \\ \times & 9 & 10 & & \\ 11 & 12 & & & \\ 13 & & & & \end{array} \tag{2.8}$$

Note how excluded cells are skipped. There will be occasions when this order changes, so when interpreting the Visual Basic code, the reader should generally not rely on triangle cells being ordered in this way.

Another point, that is worth understanding in translating between triangles and the common representation of regression problems (e.g., following McCullagh and Nelder), is that all explanatory variables are on the same footing after aliasing has been taking care of—the regression algorithm does not distinguish between exposure or development period parameters (or the offset parameter).

2.1.6 Moving to GLMs—taking the log transform

The above discussion on the model parameterization is generic in the sense that it applies to any multiplicative regression model for an incremental development triangle that is restricted to distinct, unordered parameters for exposure and development periods. This paper is more specifically about fitting a GLM to a development triangle. To linearize the multiplicative model we need to choose the logarithm as a link function. This results in the following additive model structure for the logarithms of the expected incremental amounts:

$$\begin{array}{lllll} \gamma & \beta_2+\gamma & \beta_3+\gamma & \beta_4+\gamma & \beta_5+\gamma \\ \alpha_2+\gamma & \alpha_2+\beta_2+\gamma & \alpha_2+\beta_3+\gamma & \alpha_2+\beta_4+\gamma & \\ \alpha_3+\gamma & \alpha_3+\beta_2+\gamma & \alpha_2+\beta_3+\gamma & & \\ \alpha_4+\gamma & \alpha_4+\beta_2+\gamma & & & \\ \alpha_5+\gamma & & & & \end{array} \tag{2.9}$$

We are emphasizing this step here for two reasons. Firstly, the use of a logarithmic link function restricts the model to positive incremental values. Secondly, understanding the connection between this additive model and the more generic multiplicative model is crucial to interpreting the output from GLM packages.

2.2 Graph topology of an incomplete development triangle

We noted above the cells of an incomplete development triangle can be thought of as forming a mathematical structure know as a graph. Generally a graph is collection of nodes (or vertices) that are connected to each other by edges. Two nodes are considered neighbors if there is an edge that directly joins them. We can also define an equivalence relationship among nodes called connectedness. Two nodes are connected if there is a path (or sequence) of neighboring nodes that leads from one node to the other. Two nodes are disconnected if there is no way of getting from one to the other by passing from neighbor to neighbor. This equivalence relationship of connectedness defines equivalence classes of nodes that are called maximal connected components. Note that in this paper we often refer to maximal connected components as connected components since repeating "maximal" becomes cumbersome.

If we think of the triangle cells as nodes and define two triangle cells as being neighbors if they are in the same row or column of the triangle, the collection of cells from an incomplete development triangle can be seen to form a graph. We will now briefly outline how graph theory can be used to handle the issues regarding model parameterization and exact fit cells identified above.

2.2.1 What parameters are needed for the model?

We use an algorithm known as "breadth first search" (described in detail in section 2.4) to first identify the maximal connected components of the incomplete triangle. As explained above in section 2.1.3 we can only "complete the triangle" for projection purposes if the given triangle cells form a connected component. If there is more than one connected component our algorithm proceeds by picking the largest (most triangle cells) connected component. If there is more than one largest connected component the algorithm chooses the component that has the left most column. Once we have identified the connected component for which we will fit a regression model we analyze which rows and columns are covered by the connected component. We parameterize our regression model by choosing the cell corresponding to the top row and left most column as our reference cell, with separate parameters for each other row and column. This takes care of the issues identified in section 2.1.1 and 2.1.2.

2.2.2 Which cells are exact fit cells?

As indicated above there are two circumstances under which the fitted value for a particular triangle cell will always match the given data point. Both situations can be identified by eliminating a particular cell from the regression model and seeing how the elimination affects the structure of the model. Technically we do this by looping over all cells in the selected connected component, remove each cell in turn and then run the "breath first search" algorithm also used in section 2.2.1 to analyze the structure of the remaining model. This allows us to identify three different types of cells:

Single parameter cells: when this cell is eliminated from the model, we lose one row or column and the corresponding parameter. Since this is therefore the only data point for that parameter, the parameter will always take a value that produces an exact fit for this cell.

Critical connector cells: eliminating this cell from the model, splits it into exactly two disconnected components (proof left as an easy exercise for the reader[1]). The issue of aliasing now affects both disconnected components separately, so we lose a parameter. The details of how this parameter disappears are more subtle than for single parameter cells, but the bottom line is that there is some "slack" in the parameterization and we always get an exact fit for a critical connector cell.

Regression cells: when eliminating this cell, the model structure is not affected in a significant way (same number of rows and columns covered, same number of parameters needed to parameterize the model).

Both types of exact-fit cells need to be excluded when analyzing standardized residuals and measuring the inherent uncertainty of the underlying stochastic process. The impact of the critical connector cells is more far reaching. Despite having a valid regression model for the entire connected component, the critical connector cells (if they exist) split the incomplete triangle into regions for which the regression fit is performed without any influence from the other regions. Our algorithm for fitting the GLM powerfully demonstrates this feature by literally applying the iterated weighted least square procedure to these separate regression components. To get the final parameterization for the overall model we then perform a single-weighted least square fit based on the fitted values separately obtained for regression regions and the actual data points for the identified exact fit cells. Note that it is not necessary to perform the fit separately for the regression regions, but when it comes to actual computations, it is usually more efficient to split a larger problem into separate smaller problems, especially when the computational cost scales non-linearly.

[1] Hint: if another cell, C, is connected to the critical connector cell, there has to be at least one cell that is connected to C, which either shares a row or column with the critical connector cell.

2.3 Formal set-up

To formally establish the relationship between the data for which we want to find a model (i.e., the cells of the triangle) and the explanatory variables (i.e., the exposure and development periods) we set up a data structure known as the model matrix, **X**, the columns of which represent the explanatory variables, and a corresponding column vector, **y**, which represents the data. Interested readers are referred to the CAS Practitioners' Guide to GLMs ([1]) for a general introduction to setting up information matrices. Here we will simply show an example of how a complete 5 x 5 triangle can be represented:

	Parameters **P**	γ	β_2	α_2	β_3	α_3	β_4	α_4	β_5	α_5	
	Data					*Model matrix*					*Fitted Values*
Unit #	**Y**					**X**					Exp(**X.p**)
1	inc(1,1)	1	0	0	0	0	0	0	0	0	Exp(γ)
2	inc(1,2)	1	1	0	0	0	0	0	0	0	Exp($\beta_2+\gamma$)
3	inc(2,1)	1	0	1	0	0	0	0	0	0	Exp($\alpha_2+\gamma$)
4	inc(2,2)	1	1	1	0	0	0	0	0	0	Exp($\alpha_2+\beta_2+\gamma$)
5	inc(1,3)	1	0	0	1	0	0	0	0	0	Exp($\beta_3+\gamma$)
6	inc(3,1)	1	0	0	0	1	0	0	0	0	Exp($\alpha_3+\gamma$)
7	inc(2,3)	1	0	1	1	0	0	0	0	0	Exp($\alpha_2+\beta_3+\gamma$)
8	inc(3,2)	1	1	0	0	1	0	0	0	0	Exp($\alpha_3+\beta_2+\gamma$)
9	inc(1,4)	1	0	0	0	0	1	0	0	0	Exp($\beta_4+\gamma$)
10	inc(4,1)	1	0	0	0	0	0	1	0	0	Exp($\alpha_4+\gamma$)
11	inc(3,3)	1	0	0	1	1	0	0	0	0	Exp($\alpha_3+\beta_3+\gamma$)
12	inc(2,4)	1	0	1	0	0	1	0	0	0	Exp($\alpha_2+\beta_4+\gamma$)
13	inc(4,2)	1	1	0	0	0	0	1	0	0	Exp($\alpha_4+\beta_2+\gamma$)
14	inc(1,5)	1	0	0	0	0	0	0	1	0	Exp($\beta_5+\gamma$)
15	inc(5,1)	1	0	0	0	0	0	0	0	1	Exp($\alpha_5+\gamma$)

Note that the unit #s are simply for referencing values stored in arrays and that inc(*i,j*) denotes the incremental amount for accident period *i* and development period *j*. The model matrix, **X**, has one row for each triangle cell and one column for each parameter of the model. All entries of **X** are either 0 or 1. A value of 1 simply means that the parameter corresponding to the respective column contributes to the fitted value for the triangle cell corresponding to the respective row; a value of 0 implies the converse. In the above table we have introduced the parameter vector, **p**. If $\mathbf{\hat{y}}$ denotes the vector of fitted values, the systematic part of our GLM model can neatly be summarized by the following equation:

$$\mathit{log}(\mathbf{\hat{y}}) = \mathbf{X.p} \tag{2.10}$$

2.4 The algorithm for setting up the model specification

Setting up the actual model matrix is straightforward. The real challenge here is to algorithmically analyze the incomplete development triangle to come up with a valid parameterization and to identify the exact fit cells. Before we introduce the pseudo-code for the "breadth first search" algorithm that is at the core of this undertaking, we briefly describe some auxiliary data elements. Readers in a hurry can jump right to section 2.4.1 where the pseudo-code is presented.

In the following we use some name ranges and variable names specific to the MS Excel application available from the author at request. We hope that their mention here does provide the reader with an idea of what is involved from a practical implementation point of view. We start with a complete triangle in a range called data_incremental. To exclude triangle cells from the analysis, we use a second triangle of 0s and 1s in a range called data_excluded to mask the corresponding cells in the triangle of incremental amounts.

For determining the maximal connected components of the selected triangle cells viewed as a graph, we use a "breadth first search"-type algorithm (see [5]) adapted to the fact that our graph edges (the links between cells) come in two "flavors:" shared row or shared column. To this end we need to maintain four lists of cells: (connected component) assigned, rows tested, columns tested and untested. To facilitate moving cells from and to these lists we use an array called UnitIndex that has a row for each selected triangle cell and columns for storing the cells predecessor and successor in its current list. In addition, for each list we maintain a special pointer to the first element in the list. This data structure allows us to store all four lists in parallel in the same array. The array UnitIndex also has additional columns for identifying a cell's row and column in the triangle. At times, we use a separate triangle of unit numbers to efficiently locate triangle cells in the UnitIndex array based on their position in the triangle.

There are also a number of arrays to facilitate moving from the model matrix to the actual triangle and vice versa:

UnitIndex() has fields for mapping a row of the model matrix to the corresponding triangle cell

Data_Selected() or Data_Regression() are used to either mark some specific triangle cells for further processing or to store pointers from the triangle cells to the corresponding row in UnitIndex()

GLM_Par_To_Triangle() maps each column of the model matrix (or the parameter row vector) to the corresponding row or columns of the triangle

ExporsurePeriod_To_GLM_Paramter() maps rows of the triangle to columns of the model matrix

DevelopmentPeriod_To_GLM_Paramter() maps columns of the triangle to columns of the model matrix

2.4.1 Determining connected components of the incomplete development triangles

Below we outline pseudo-code for the "breadth first search"-type algorithm for an incomplete development triangle. Here is how the general "breadth first search" algorithm works. It is called "breadth first" because while we are trying to find all nodes connected a particular untested node we first mark all its immediate neighbors before trying to find neighbors of neighbors. Once all the immediate neighbors of a node have been identified we are done with that node. What is left to do is to loop over all the immediate neighbors previously identified and check whether they have any neighbors we have not looked at, yet. For each node we are done once we have marked all of its immediate neighbors. This process continues until we cannot find any further new neighbors. We have now identified the maximal connected component of the original untested node. If there are untested nodes left, we know that our graph has a further connected component and we start the process all over to find all the nodes belonging to the next connected component.

For a development triangle the algorithm generally proceeds exactly the same way except that we are now done with a particular cell when we have identified all its row and all its column neighbors. In order to efficiently do this, we introduce two lists of cells with an intermediate testing status: all row neighbors identified (i.e., still need to check column neighbors) and all column neighbors identified (i.e., still need to check row neighbors). Hence there are two possible processing paths for a particular triangle cell: untested to column tested to component assigned, or untested to row tested to component assigned.

Here is the pseudo-code:

Input:	DataSelected....................	two-dimensional array indicating which triangle cells are included in model
	UnitIndex.........................	array for storing information about data points and maintaining list structures utilized by algorithm
	ConnectedComponent...	array for keeping track of maximal connected components and some key properties
Goal:	Assign component sequence number to each selected triangle cell and gather summary information about connected components	

BreadthFirst 1) Initialize UnitIndex and associated data structures needed for determining maximal connected components of selected triangle cells. In particular the list of untested cells contains all cells included by user, while the list of assigned, row tested, or column tested cells are empty.

BreadthFirst 2) Beginning of outer loop - keep going while the list of untested cells is empty.

BreadthFirst 3) Increment component counter; get column of last untested cell; loop over untested rows in this column; assign component counter to untested cells found and move them from the untested list to the column tested list.

BreadthFirst 4) Beginning of inner loop - keep going while there are newly identified neighbors for the current connected component.

BreadthFirst 5) Loop over cells in column tested list; get row of current column tested cell; move current cell from column tested list to assigned list; loop over untested columns in current row; assign component counter to untested cells found and move them from the untested list to the row tested list.

BreadthFirst 6) Loop over cells in row tested list; get column of current row tested cell; move current cell from row tested list to assigned list; loop over untested rows in current column; assign component counter to untested cells found and move them from the untested list to the column tested list.

BreadthFirst 7) End of inner loop - if list of column tested cells is not empty, execution will continue with BreadthFirst 5).

BreadthFirst 8) End of outer loop - if list of untested cells in non-empty, execution will continue with BreadthFirst 3).

2.4.2 Selecting connected component for the subsequent model fit

While it would be possible to fit a regression model for each separate connected component, there is no way of using these disconnected models for projecting future development for all periods. Hence our algorithm proceeds by selecting the component with the most triangle cells for further processing. If there is more than one largest connected component the algorithm simply picks the component with the left most column. Note that in the companion spreadsheet, which is available from the author at request, there is an exhibit that shows all connected components. If the user does not like the default choice imposed by the algorithm, they can change the selected data points to only include the connected component that they are interested in.

2.4.3 Determining the cell types for the selected connected component

As described in section 2.2.2, above, there are three types of cells within each connected component: single parameter cells, critical connector cells, and regression cells. For reference let NoUnits stand for the number of cells in the selected connected component. We test for the cell types by looping over all the cells in the selected connected component and remove each cell in turn from the selected component. After removing the cell we analyze the structure of the remaining cells. In order to do so it is sufficient to start with any of the remaining cells and then run the inner loop of the "breadth first search" algorithm described above. If the maximal connected component associated with that cell has less than NoUnits - 1 cells, we know that the removed cell must be a critical connector cell. Otherwise we need to check whether we lost a row or column by removing the cell. If we did lose a row or column, then the removed cell is a single parameter cell. If the latter is not the case, we know by elimination that the removed cell is a regression cell. The above-mentioned exhibit for the connected components also visualized the cell types for cells in the selected components by formatting; single parameter cells have a border, critical connector cells are crossed out, and regression cells have a grey fill.

2.4.4 Determining the regression components within the selected connected component

We now run the full "breadth first search" algorithm again, but only on the regression cells within the selected connected component. This allows us to identify decoupled areas of connected regression cells if they exist. Gathering this information allows us to split the fitting of the overall model into smaller pieces, each of which purely consists of regression cells.

2.4.5 Setting up the model matrix and associated data vector

Having done all the preparatory work of analyzing the graph topology of the incomplete triangle the task of setting up the model matrix and data vector is trivial. We set up separate model matrices and data vectors for each of the regression regions identified above. We also set up one overall model matrix and data vector. Note however, that we will run the full-fitting algorithm only on the model matrices for the regression regions. The model matrix for the overall model will be used to for a single iteration weighted least square fit based on the fitted values obtained for the regression regions and the actual data points for the exact fit cells. This last step is needed to obtain a convenient parameterization for the overall model that can be used for projection purposes. Note, however, that the model matrix for the overall model is perfectly valid and that feeding it into a GLM fitting algorithm should produce the same parameter values (give or take some rounding) as the approach we are taking. Keeping track of the exact fit cells and regression regions, however, is computationally advantageous and provides useful information for the subsequent residual analysis.

3 MAXIMUM LIKELIHOOD ESTIMATION USING ITERATED WEIGHTED LEAST SQUARES

In section 2 we went into considerable detail of how our algorithm for setting up the model matrix works, because we are not aware of such a step-by-step description in the actuarial literature. Algorithms for fitting a generalized linear model (GLM) by using a maximum likelihood estimator, however, are described in many text books (e.g., chapter 2.5 in [6] or chapter 6.4.2 in [9]). We will therefore concentrate on how to implement such an approach relying on standard linear algebra routines available in open source form—we have used code from the ALGLIB project available for download at www.alglib.net. Other than demonstrating that Visual Basic for MS Excel is well capable of fitting a GLM to triangles of considerable size, we also want to show that in the process of performing the calculations we can extract useful information[2] other than the fitted parameter values.

3.1 Notation

Before continuing we need to introduce some standard notation for describing a GLM model. Often a GLM model is specified by assuming a specific distribution from the exponential family for the observations. In practice the algorithm for fitting a GLM works just as well under the weaker assumption that the second moment of the distribution is a function of the expected mean (see chapter 9 on pseudo-likelihood functions in [6]). Hence we regard a GLM type problem to be fully specified by the following elements:

- Model matrix, $\mathbf{X}$
- Data vector, $\mathbf{y}$
- Fitted values vector, $\hat{\mathbf{y}}$
- Link function, g—note that here we only will use the logarithm
- Variance function, V

Additionally, the following notation is useful in describing the algorithm and the computations:

- Parameter vector, $\mathbf{p}$
- Linear estimator, $\boldsymbol{\eta}$
- Vector of quadratic weights for weighted LSQ regression, $\mathbf{w}$
- Vector of linearized data, $\mathbf{z}$
- Vector of expected variance for data points, $\mathbf{v}$

[2] Specifically we are referring to the diagonal elements of the hat matrix, which can be used to standardize residuals. This will be discussed in more detail in section 4.

The algorithm described here follows the approach outlined in chapter 2.5 of [6]—we perform iterated weighted least square regressions. This procedure can be understood as an adaption of the multi-dimensional Newton-Raphson method to the problem of solving the maximum likelihood equations for a GLM. In effect we are repeatedly solving a linearized regression problem until the successive solutions have converged to a sufficient degree. In general such numerical procedures can be sensitive to the starting point chosen. Here we can use the actually observed values for the data points (triangle cells), which makes the practical implementation easy.

3.2 Pseudo-code for MLE algorithm

Deferring our discussion of how to perform the weighted least squares (LSQ) regression, here is how the MLE algorithm implemented here works:

Step 0) Based on the actual data points, initialize the fitted data points, the linear estimator, and the expected variances: $\hat{\mathbf{y}}_0 = \mathbf{y}$, $\boldsymbol{\eta}_0 = g(\mathbf{y}) = log(\mathbf{y})$, and $\mathbf{v}_0 = V(\mathbf{y})$.

Step 1) Based on a current estimator, $\boldsymbol{\eta}_i$, determine the vector of linearized data using the following formula:

$$\mathbf{z}_i = \boldsymbol{\eta}_i + (\mathbf{y} - \hat{\mathbf{y}}_i)\left(\frac{d\boldsymbol{\eta}}{d\hat{\mathbf{y}}}\right). \tag{3.1}$$

Step 2) Based on a current estimator, $\boldsymbol{\eta}_i$, the expected variances, $\mathbf{v}_i$, calculate the vector of weights for weighted LSQ regression:

$$\mathbf{w}_i = \left(\left(\frac{d\boldsymbol{\eta}}{d\hat{\mathbf{y}}}\right)^2 \mathbf{v}_i\right)^{-1}. \tag{3.2}$$

Step 3) Perform weighted LSQ regression of $\mathbf{z}_i$ on $\mathbf{X}$ subject to $\mathbf{w}_i$ to obtain new set of parameters, $\mathbf{p}_{i+1}$, leading to a new liner estimator, $\boldsymbol{\eta}_{i+1}$, and fitted values, $\hat{\mathbf{y}}_{i+1}$.

Step 4) Compare $\mathbf{p}_i$ with $\mathbf{p}_{i+1}$ to determine whether convergence is satisfactory. If not, repeat from Step 1).

Step 5) Extract diagonal elements of hat matrix, **H**, calculate deleveraged residuals, estimate dispersion parameter, and calculate standardized residuals.

Step 6) Loop over exact-fit cells and recursively determine exact-fit parameters based on parameters obtained with MLE algorithm.

Note that with the logarithm as our link function we get $\boldsymbol{\eta} = g(\hat{\mathbf{y}}) = log(\hat{\mathbf{y}})$, and $d\boldsymbol{\eta}/d\hat{\mathbf{y}} = 1/\hat{\mathbf{y}}$. Readers who are interested in why this procedure (specifically steps 0 to 4) works are referred to chapter 2.5.1 in [6]. Step 5) will be discussed in detail in section 3. Step 6) utilizes the information gathered by the algorithm for setting up the model matrix, **X**, as described in section 2. While this also reduces the computational cost, the main purpose is to keep track for which data points we can calculate standardized residuals.

3.3 Notes on implementation issues

Implementing this general algorithm in Visual Basic is pretty straight forward. To make the implementation flexible regarding specific choices of variance and link functions we are using a number of generic functions that such as Calc_dL (for $d\boldsymbol{\eta}/d\hat{\mathbf{y}}$), Calc_Variance (for $\mathbf{v}$), and Calc_Weights (for $\mathbf{w}$), that are simply passing through a symbolic parameter indicating the current choice of link and variance function to lower level functions that compute those values for specific data points (Link_Scalar, LinkInv_Scalar, Var_Scalar, dL_Scalar). Note that despite this flexibility in design, the template does only implement the logarithmic link function.

For those readers who want to go over the Visual Basic code in detail, please note that we are making extensive use of passing data by reference to provide functions with input and to return the results. The formal function results are mainly used for debugging and error handling. Passing by reference means that subroutines are given direct access to data structures (variables and/or arrays) defined at a higher level rather than having their own copy of the data passed in. So the main point of a statement like

```
If Not Calc_dL(dL, Y_, LinkFunc) Then Stop
```

is not to stop execution if the function Calc_dL returns FALSE. Rather we want to calculate the vector of derivatives for the link function, which is accomplished as a side effect to the function call by updating the values of the vector (1-d array) dL that is passed to the function by reference. The "If … Then Stop" statement is simply a way of pointing the developer to the right position in the code in case something were to go wrong.

Performing the weighted LSQ regression is mainly an exercise in linear algebra. Built-in Excel functions only provide limited support for matrix operations and the matrices the built in functions can deal with is limited. We are therefore providing some auxiliary routines for simple matrix operations. A list with short descriptions follows:

- MClearUpper.. clears the upper right triangle of a square matrix **A**; needed because the Cholesky transform routine from ALGLIB assumes that symmetric matrix is represented in triangular format
- MTrans............ populates matrix transpose $\mathbf{A}^T$ of matrix **A**
- MMult............. populates matrix **R** with the product of matrices **A** and **B**

- MDiagRMult... left multiplies matrix **A** with a diagonal matrix that is represented as a vector **diag**
- MDiagLMult... right multiplies matrix **A** with a diagonal matrix that is represented as a vector **diag**
- MSet................ populates matrix **R** by copying entries of matrix **A**
- MSwapCol....... swap column c1 and c2 of matrix **A**
- MSwapRow..... swap row r1 and r2 of matrix **A**
- MQuickMult... populates vector **r** with the product of matrix **A** and vector **b**
- VectorSqrt....... populate vector **sqrt_w** with square root of elements of vector **w**

As indicated above, we are also using code from the open source ALGLIB project (www.alglib.net). The following functions are Visual Basic implementations of LAPACK routines (www.netlib.org/lapack/):

- SPDMatrixCholesky.perform Cholesky decomposition of matrix A which is assumed to be a symmetric matrix stored in triangular format.
- RMatrixTRInverse....perform matrix inversion of triangular matrix A.

We will not explain the algorithms in detail, but we do want to briefly outline why these routines are useful for our purposes. The key computational step in performing the weighted LSQ regression is the inversion of the matrix $\mathbf{X}^T.\mathbf{W}.\mathbf{X}$, where **W** is the diagonal matrix of weights represented by the vector **w**. If the regression problem is well defined, this matrix is symmetric and positive-definite. Now, if you have a symmetric, positive-definite matrix **A**, then there is a lower triangular matrix **L** such that $\mathbf{A}=\mathbf{L}.\mathbf{L}^T$. This representation of **A** is called its Cholesky decomposition. It follows from basic matrix algebra that $\mathbf{A}^{-1}=(\mathbf{L}^{-1})^T.(\mathbf{L}^{-1})$. Hence we can see how the inversion of $\mathbf{X}^T.\mathbf{W}.\mathbf{X}$. can be accomplished by first finding its Cholesky decomposition and then inverting the resulting lower triangular matrix.

In symbolic form the weighted LSQ regression estimate for the parameters based on the linearized data is given by:

$$\mathbf{p} = (\mathbf{X}^T.\mathbf{W}.\mathbf{X})^{-1}.\mathbf{X}^T.\mathbf{W}.\mathbf{z}. \tag{3.3}$$

3.4 Pseudo-code for weighted LSQ regression

We now present an outline of the code for performing the weighted LSQ regression. Note that in essence we are building up three matrices: $\mathbf{X}^{\mathbf{T}}.\mathbf{W}$, $(\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X})^{-1}$, and $(\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X})^{-1}.\mathbf{X}^{\mathbf{T}}.\mathbf{W}$. The corresponding variables are called XtW, XtWXinv, and XtWXinvXtW:

```
wLSQ  1)  MTrans_C(XtW, X)—i.e., XtW = $\mathbf{X}^{\mathbf{T}}$
wLSQ  2)  MDiagRMult_C(XtW, w)—now XtW = $\mathbf{X}^{\mathbf{T}}.\mathbf{W}$
wLSQ  3)  MMult_C(XtWXinv, XtW, X)—i.e., XtWXinv = $\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X}$
wLSQ  4)  MSet_C(M1, XtWXinv)—i.e., M1 = $\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X}$
wLSQ  5)  MClearUpper_C(M1)—prepare M1 for Cholesky routine
wLSQ  6)  MCholesky_C(M1)—calculate the $\mathbf{L}$ for $\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X}$
wLSQ  7)  MTriInv_C(M1, t)—i.e., M1 = $\mathbf{L}^{-1}$
wLSQ  8)  MTrans_C(M2, tmpM1)—i.e., M2 = $(\mathbf{L}^{-1})^{\mathbf{T}}$
wLSQ  9)  MMult_C(XtWXinv, M2, M1)—i.e., XtWXinv = $(\mathbf{L}^{-1})^{\mathbf{T}}.(\mathbf{L}^{-1})$
wLSQ 10)  MMult_C(XtWXInvXtW, XtWXinv, XtW)—used for calculating $\mathbf{h}$
```

The rest of the tasks associated with Step 3) of the <u>iterated</u> weighted LSQ algorithm is accomplished with the following statements:

```
wLSQ 11)  MQuickMult(p, XtWXInvXtW, z)—i.e., $\mathbf{p}_1 = (\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X})^{-1}.\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{z}_0$
wLSQ 12)  MQuickMult(z_, X, p)—i.e., $\boldsymbol{\eta}_1 = \mathbf{X}.\ \mathbf{p}_1$
wLSQ 13)  LinkInv(y_, z_, LinkFunc)—i.e., $\hat{\mathbf{y}}_1 = g^{-1}(\boldsymbol{\eta}_1)$
```

3.5 Concluding remarks

We encourage interested readers to contact the author and request a copy of the accompanying MS Excel application, so they can study the commented source code for further implementation details.

One note on performance: the computational cost of many sub-tasks except for the matrix operations varies linearly with the number of triangle cells included in the analysis. For fitting a single model the performance of the Visual Basic code seems satisfactory, even for larger triangles (the author tested up to 40 by 40). For repeated applications (such as bootstrapping) execution time can become an issue. Without changing the logic, significant gains in performance can be gained from porting the weighted LSQ routines to C++ and compiling them into a dll, which is then loaded by the Visual Basic code. In this paper we do not discuss the issues of interfacing Visual Basic with a dll and/or how to write code in C++ that can be compiled into a dll. For readers interested in those issues we recommend Steve Dalton's *Financial Applications using Excel Add-in Development in C/C++* (different versions exist for MS Excel 2003 and MS Excel 2007).

4 STANDARDIZED RESIDUALS AND THEIR APPLICATION

In this section, we introduce standardized residuals and how to compute them. After briefly describing how to create diagnostic plots based on these standardized residuals we present a practical example (based on a data set also used in a number of other papers on triangle-based stochastic reserving) of how these plots can be used to assess goodness-of-fit and make decisions about the model structure.

4.1 Standardized Residuals

Residuals are simply the difference between the actual data points and their fitted values based on a concrete model specification. Suitably standardized, so comparisons for residuals corresponding to different data points can be made, residuals can be a powerful tool for visually analyzing the goodness of fit of a model. Standardizing residuals is also a crucial step in bootstrapping a GLM. The approach here follows chapter 12.5 and 12.7 in [6], but similar descriptions of the standardization procedure can also be found in [7] and in chapter 7.2 of [3].

For a GLM, there are two adjustments to residuals that need to be made in order get residuals that are approximately identically distributed. Firstly we need to adjust for the differences in expected variances based on the relationship imposed by the variance function. This motivates the following definition of Pearson residuals:

$$\bar{\mathbf{r}} = \frac{\mathbf{y} - \hat{\mathbf{y}}}{\sqrt{V(\hat{\mathbf{y}})}} . \tag{4.1}$$

In addition we also need to adjust for the leverage that individual data points exert on their corresponding fitted value. At the extreme there are the exact-fit cells where the fitted value will always identically match the observed data point. For other points the observed value will still exert a pull on the fitted value that biases the residuals as a measure of variability inherent in the data. For a detailed discussion of the concept of leverage, we refer the reader to chapter 12.7 in [6]. A useful measure of leverage can be obtained by the diagonal values of the hat matrix, H, which for a GLM is defined as (equation 12.3 in [6]):

$$\mathbf{H} = \mathbf{W}^{½}.\mathbf{X}.(\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X})^{-1}.\mathbf{X}^{\mathbf{T}}.\mathbf{W}^{½}. \tag{4.2}$$

Note that since we are only interested in the vector **h** of diagonal elements we can also use the following formula:

$$\mathbf{h} = \mathrm{diag}(\mathbf{X}.(\mathbf{X}^{\mathbf{T}}.\mathbf{W}.\mathbf{X})^{-1}.\mathbf{X}^{\mathbf{T}}.\mathbf{W}). \tag{4.3}$$

With this we can now introduce the following definition of deleveraged Pearson residuals:

$$\tilde{\mathbf{r}} = \frac{\mathbf{y} - \hat{\mathbf{y}}}{\sqrt{V(\hat{\mathbf{y}}) \cdot (1 - \mathbf{h})}} . \tag{4.4}$$

Note that the deleveraged Pearson residuals are not defined for exact fit cells. Also note that this definition is still missing the estimated dispersion factor as a normalizing constant. The reason for this is that we are proposing the following estimator for the dispersion factor:

$$\hat{\phi} = \mathbf{Var}(\tilde{\mathbf{r}}) \tag{4.5}$$

Compare this to the conventional estimator (unnumbered formula on page 328 in [6]), based on an after-the-fact degree of freedom adjustment:

$$\hat{\phi} = \mathbf{Var}(\bar{\mathbf{r}}) \cdot \frac{n}{n - p} \tag{4.6}$$

where *n* is the number of data points and *p* is the number of estimated parameters. Both estimators for the dispersion parameter are ad hoc and are trying to adjust for the leverage effect. Equation 4.5 relies on the bias correction applied to each individual residual. Equation 4.6 does not distinguish between the difference in leverage for individual data points. Further research is required for assessing the relative performance (e.g., in terms of bias or standard error) of these two estimators. For the purposes of assessing goodness of fit and for bootstrapping applications, deleveraged residuals are the better choice since the assumption that they are approximately identically distributed is more likely to be true. For this reason, we continue with using equation 4.5, but the accompanying excel application displays both versions of the estimate of the dispersion factor.

With this we derive at our final definition of standardized Pearson residuals:

$$\mathbf{r}^* = \frac{\mathbf{y} - \hat{\mathbf{y}}}{\sqrt{\hat{\phi} \cdot V(\hat{\mathbf{y}}) \cdot (1 - \mathbf{h})}} . \tag{4.7}$$

This concludes our discussion of the subtasks summarized as step 5 in MLE algorithm outlined in section 2. For convenience, we are repeating the step:

Step 5) Extract diagonal elements of hat matrix, **H**, calculate deleveraged residuals, estimate dispersion parameter, and calculate standardized residuals.

4.2 Graphical Representation

Now that we have defined standardized residuals and know how to calculate them, the task of plotting them in MS Excel is straightforward. The main trick for getting pretty plots is to keep track of how many residuals we are actually trying to plot. We address this issue by having the MLE algorithm automatically update named ranges, which are used to specify the data source for the plots. We are graphing standardized residuals against exposure period, development period, calendar period, and size of fitted value. As an aid in the visual inspection we are adding a trend line to each of the plots. For the period-based plots this trend line simply is the average of all residuals for that period. For the plot against size of fitted value, we perform a linear regression of the residuals on the fitted values to plot the trend line.

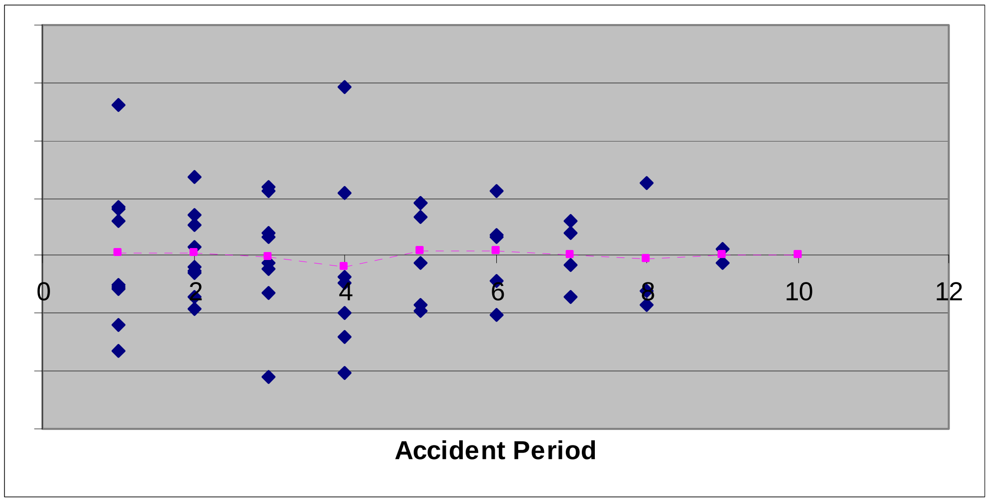
Plots of standardized residuals against various axis of interest are a standard tool for assessing goodness of fit in stochastic modeling. Barnett and Zenwirth's paper on "Best Estimates for Reserves" ([2]) has popularized the concept in the context of analyzing actuarial development triangles and they are now a staple of stochastic reserving packages. The idea is that standardized residuals should be randomly and identically distributed. If there are any obvious systematic trends visible in the plots, then the residuals are not random after all. In addition one can also detect extreme outliers and an experienced analyst may also infer other information that can be used to find an optimal model.

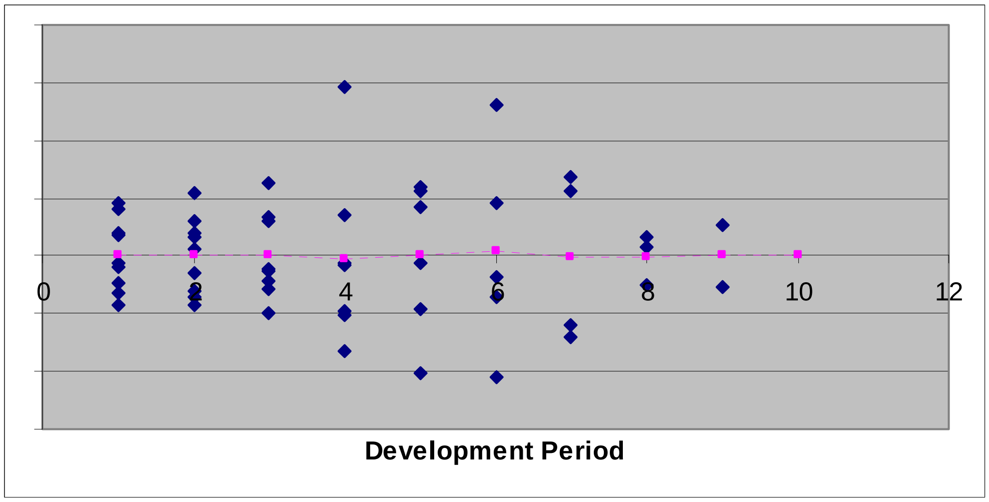
4.2 Example

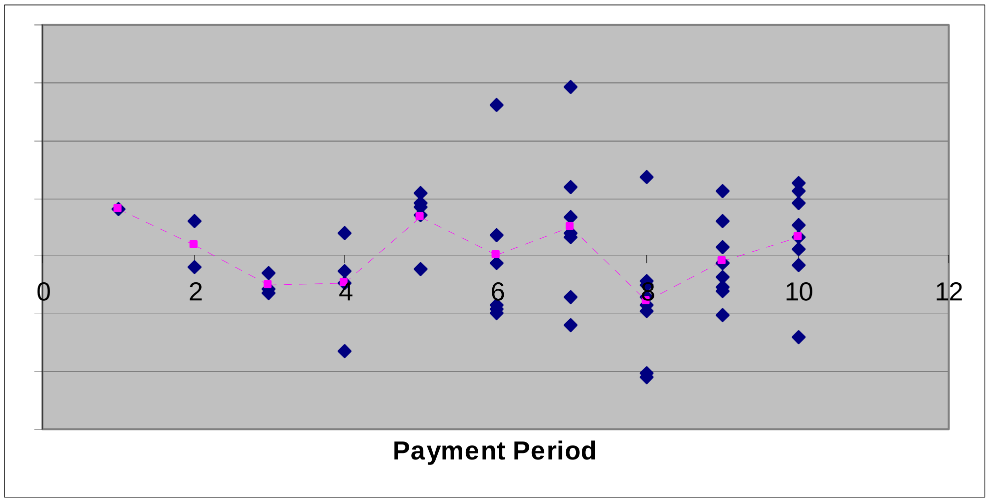
We are using a data set that the authors of [7] attribute to Taylor and Ashe (1983). Here is the data in incremental form:

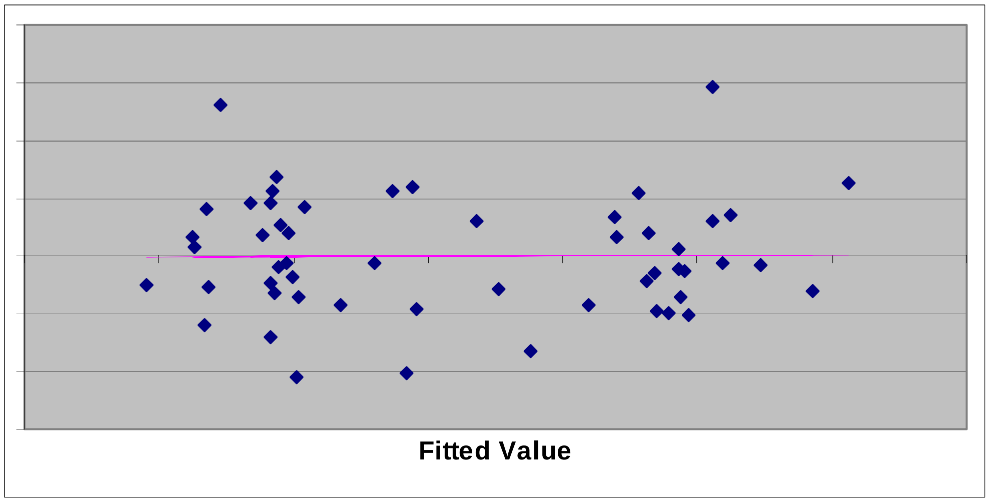
357,848	766,940	610,542	482,940	527,326	574,398	146,342	139,950	227,229	67,948
352,118	884,021	933,894	1,183,289	445,745	320,996	527,804	266,172	425,046	
290,507	1,001,799	926,219	1,016,654	750,816	146,923	495,992	280,405		
310,608	1,108,250	776,189	1,562,400	272,482	352,053	206,286			
443,160	693,190	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									

There are no non-positive or missing data points, so we can go ahead and fit a model based on the full triangle. To start with we chose the identity variance function, which produces a model that preserves the row and column sums of the triangle and is equivalent to the model obtained by developing the triangle using the all-year, volume-weighted link ratios. This results in the following residual plots:

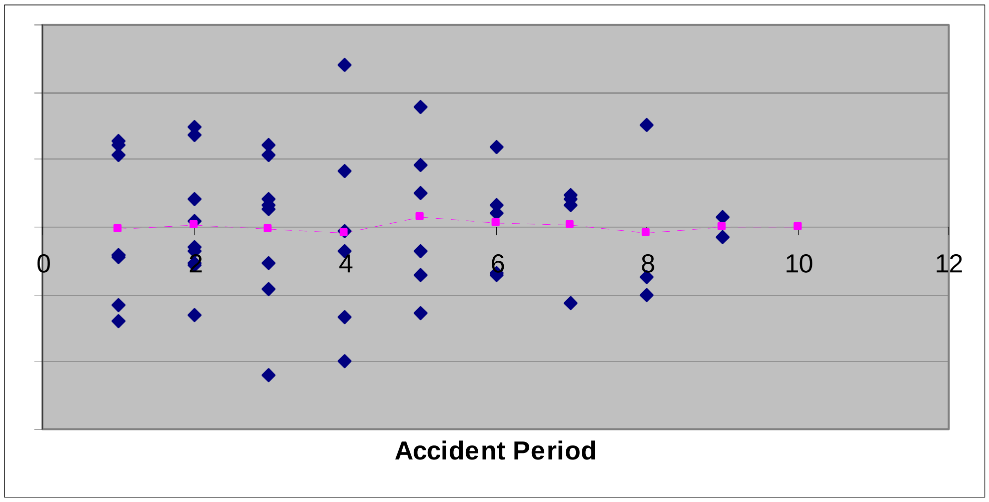


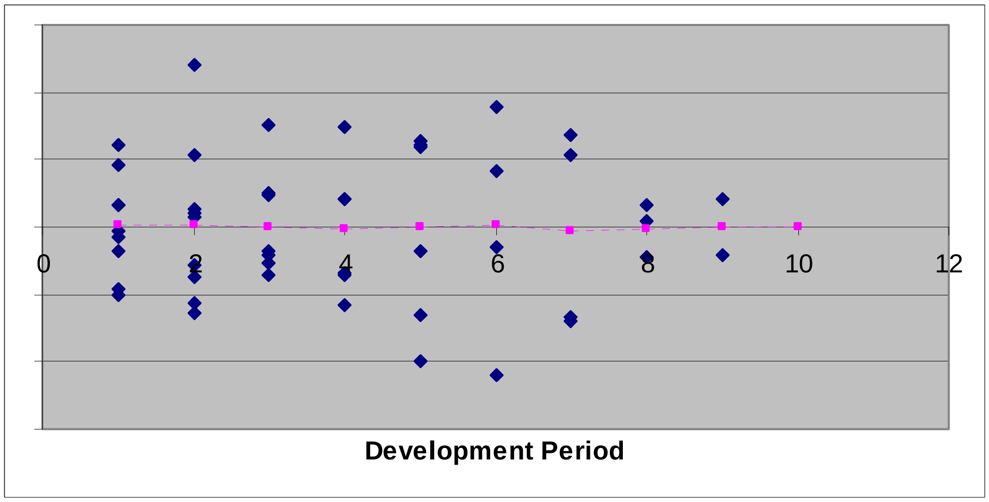

0
2
4
6
8
10
12
Development Period


0
2
4
6
8
10
12
Payment Period

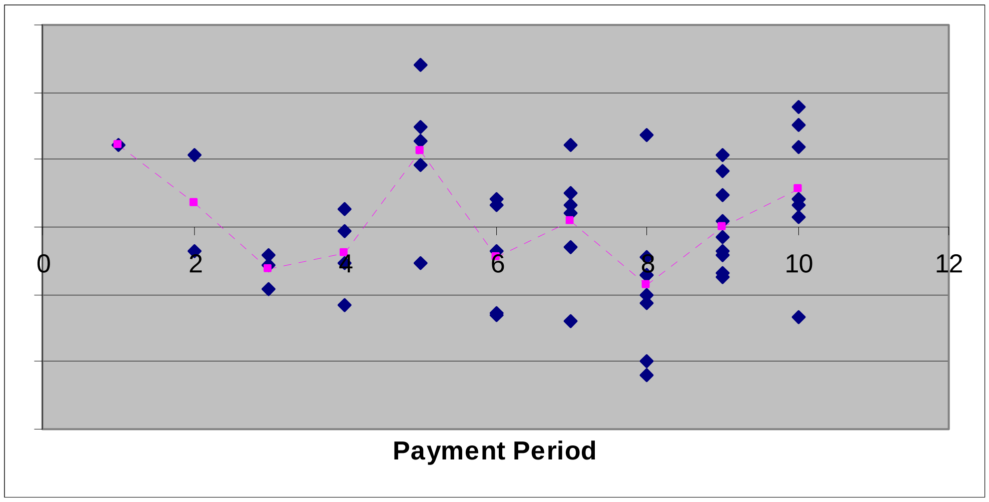

Fitted Value

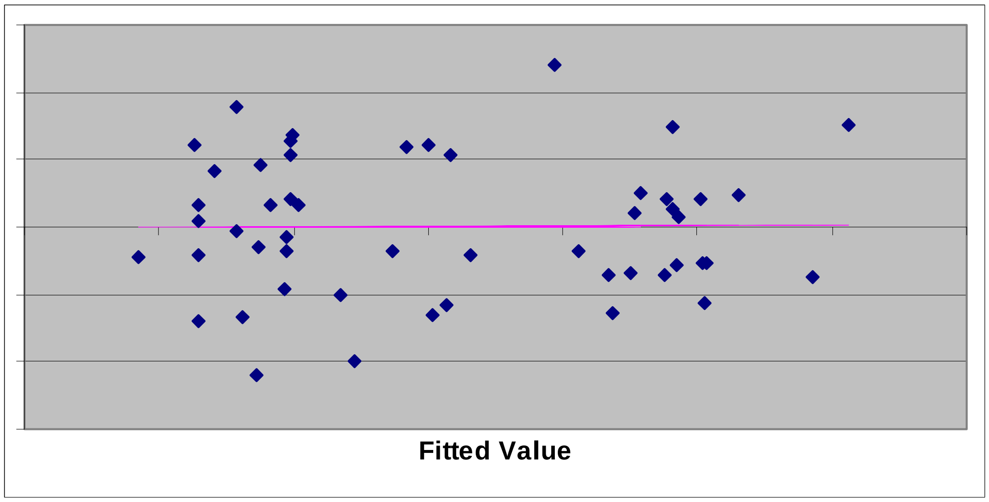
From the jagged trend line for calendar period plot we can see that there are calendar year effects that our model does not capture. From the exposure period and development period plots, we can see that there are two relatively large residuals for cells (4,4) and (1,6)—looking at the data set above we can see that the corresponding values are 1,562,400 and 574,398, respectively. Inspection of the corresponding columns in the triangle does confirm that these values appear unusually high. For demonstration purposes, we assume that theses values represent abnormal circumstances that should not be included in our model to predict future development. Hence we exclude these data points. Technically we have now parameterized a new model and the new residual plots look as follows:





Before looking at the remaining plots on the next page, please note that these plots should mainly be used for assessing the internal consistency of the current model. We emphasize that by excluding data points we end up with different models that cannot be directly compared. With that said, we note that while there are still ups and downs in the exposure and development period plots, the biggest and smallest residuals now do not look way out.



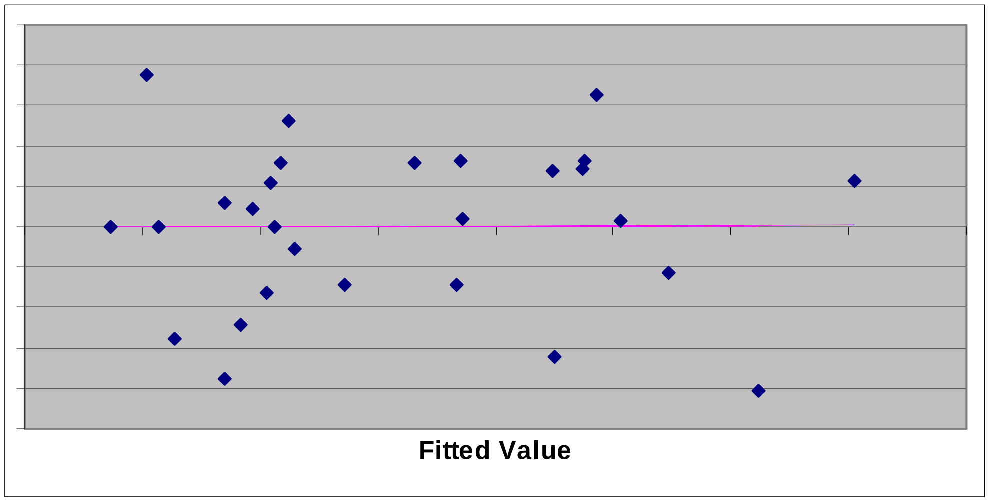


The calendar period plot looks no better than before. Since our model only has exposure and development period parameters, we have limited options for responding to these calendar period effects. For demonstration purposes we will try a model that only includes the latest five diagonals except for the diagonal for calendar period 8, which looks out of line, and cells (4,4) and (1,6), which we previously excluded.

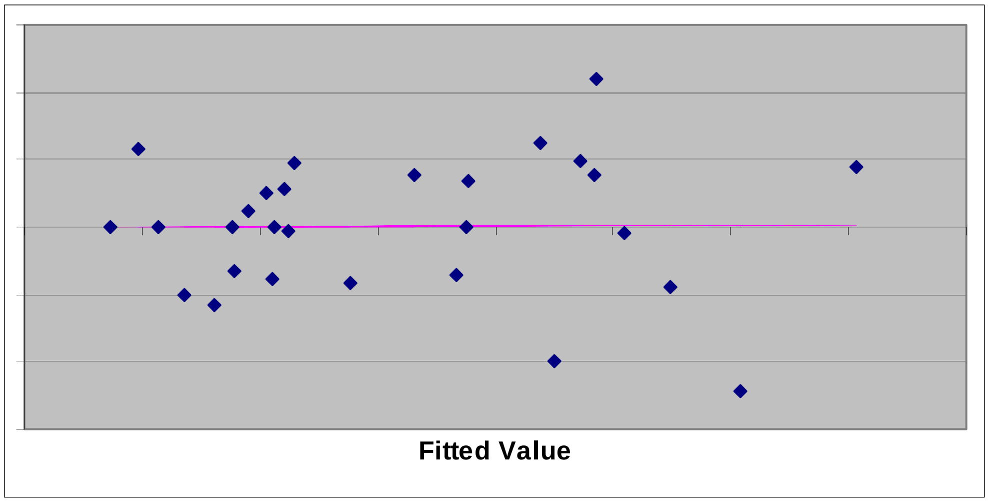
At this point, let us have a look at the output produced by our model. One feature of this type of model is that we cannot only project expected values for future triangle cells, but we can also extrapolate the expected values for all past triangle cells that were excluded from the analysis. In the table on the following page we show all fitted values that correspond to included data points in bold letters. All values in italics correspond to projections/extrapolations based on the fitted parameters for the model. Since we have chosen the identity variance function the reader can also verify that fitted values in bold preserve the row and column sums of the original data points for the included triangles cells along the last five diagonals.

142,392	*330,441*	*425,664*	*331,922*	*244,123*	*196,001*	**146,600**	*110,970*	**226,971**	**67,948**
266,817	*619,189*	*797,621*	*621,963*	**457,443**	**367,273**	*274,703*	**207,939**	**425,304**	*127,323*
434,523	*1,008,377*	*1,298,962*	**1,012,895**	**744,967**	*598,120*	**447,366**	**338,638**	*692,627*	*207,351*
247,343	*573,998*	**739,407**	*576,569*	*424,057*	**340,467**	**254,654**	*192,763*	*394,263*	*118,030*
316,708	**734,969**	**946,766**	*738,262*	**542,980**	**435,948**	*326,069*	*246,821*	*504,831*	*151,130*
386,120	**896,049**	*1,154,264*	**900,064**	**661,982**	*531,493*	*397,532*	*300,915*	*615,472*	*184,253*
416,980	*967,665*	**1,246,518**	**972,001**	*714,890*	*573,972*	*429,304*	*324,966*	*664,663*	*198,979*
471,751	**1,094,769**	**1,410,249**	*1,099,674*	*808,792*	*649,363*	*485,694*	*367,650*	*751,967*	*225,115*
410,550	**952,744**	*1,227,297*	*957,013*	*703,867*	*565,121*	*422,684*	*319,955*	*654,414*	*195,911*
344,014	*798,336*	*1,028,394*	*801,913*	*589,794*	*473,534*	*354,182*	*268,101*	*548,356*	*164,160*

There is one last feature of the MLE algorithm implemented here that we want to demonstrate: the user can choose from a number of pre-defined variance functions (identity, unity, square root, power 2, and power family with a specifiable positive exponent). To see the effect this may have on the model, consider the residuals versus fitted size plot for our latest model with the identity variance function:



For comparison we will also be fitting a model with the same data points but using the unity variance function. With the unity variance function all data points are assumed to have the same expected variance. Hence we would expect that, relative to the identity variance function, smaller fitted values are given more weight and therefore should have smaller residuals (relative to the residuals for larger fitted values). The reader can inspect the residual plot on the following page to see whether this expectation holds up.



This concludes our walk through on how to use the MLE template. We encourage readers to contact the author to request a copy of the companion MS Excel template and put it through its paces with data sets of your choice.

5. RESULTS AND DISCUSSION

In section 2.1, we visually presented the issues that need to be addressed when setting up a model matrix for a development triangle-assuming a multiplicative model which has distinct unordered parameters for exposure and development periods.

The following section 2.2 more formally introduces some concepts from graph theory that allow us to analyze the graph topology of an incomplete development triangle. This enables us to generate a valid model specification and to gather information that is important for interpreting the output of the fitting procedure. In particular we can identify exact fit cells and sub-regions of the incomplete development triangle for which the regression fit is performed without any influence from other regions.

Section 2.4 provided details for an original algorithm based on graph theory that generates a valid model specification for a regression model that can be used to project future development for an incomplete development triangle. If necessary the algorithm will restrict the model to a maximal connected component of the selected incomplete triangle. Note that with minor modifications the algorithm would also work for a rectangle of data points (still assuming a two-dimensional factorial model).

An outline of how to use iterated weighted least squares to implement a maximum likelihood estimator for a GLM was given in section 3.2. The algorithm (and its implementation in Visual Basic, which can be found in the accompanying MS Excel application) is generic and should work for any valid model matrix. An interested reader could easily extend the functionality by adding their own code for other link functions, the derivative of the link function, and/or code for other variance functions.

After providing further guidance on implementation issues in section 3.3, we explained the concrete steps required for performing a weighted least squares regression in section 3.4. This code utilized some simple routines for basic matrix operations tailored to our specific application and standard LAPACK routines implemented in Visual Basic from the open source ALGLIB project. We note in section 3.5 that performance of the MLE algorithm can be significantly improved by porting the matrix routines to C++ and compiling them into a dll that then can be accessed from Visual Basic.

In section 4.1 we presented an intuitive account of the concept of leverage and its role in the computation of standardized residuals that can be used for visual analysis of goodness of fit and for bootstrapping applications. We also proposed an alternative estimator for the dispersion parameter which is based on deleveraged residuals that have been individually adjusted for the bias introduced by leverage. The conventional estimator uses a degree of freedom adjustment that in effect is uniformly applied to all Pearson residuals.

After providing a brief outline on how to create plots of standardized residuals we presented a "walk through" of how these plots can be used in assessing goodness of fit and making informed choices about which data points should be included for a concrete triangle that has also been used in other papers on stochastic reserving. Note that the intention of the walk through is not to provide an optimal model for the specific data set, but to showcase the kind of judgments an analyst can make in the context of the type of GLM model presented here.

The material in this paper is based on standard GLM theory and standard numerical methods. We hope, that by presenting a complete open-source implementation, we can contribute to making more actuaries aware of how these powerful methods can be used in the context of stochastic reserving, and that interacting with the accompanying MS Excel template will prove a useful aid to gaining a deeper understanding of what regression models can accomplish in the context of development triangles.

Interested readers are encouraged to contact the author and request a copy of the companion MS Excel application to further explore the concepts and algorithms presented in this paper.

We want to conclude this section by comparing the model presented in this paper to some other stochastic models for incremental development triangles discussed in the actuarial literature. The type of model described in this paper (multiplicative effects with discrete parameters for exposure and development periods) is the same as the model described in [7], except that we provide the full apparatus that allows an analyst to exclude arbitrary triangle cells from the analysis. By allowing for excluded data points and different choices of variance functions we somewhat extend the simple bootstrapping models discussed in [4] and [7]. As noted also in these papers a GLM with the identity variance function produces the same estimated reserve as the all-year, volume-weighted link ratio method, when applied to a complete development triangle. But, as observed in [9], such a GLM is more like the traditional BF or Cape Cod method than the link ratio method, in the sense that generally we derive a development pattern and an estimate of exposure by exposure period and then calculate future development by multiplying the two.

Models such as those proposed in [2] and [8] differ from the model presented here in at least two important ways. The first difference is that the models in [2] and [8] effectively group development (or even exposure) periods together and parameterize them using either parametric curves or forms non-parametric smoothing (also discussed in [4]). The second difference is that these models add the calendar period as an additional dimension to the analysis. The model proposed in [8] and some models discussed in [4] furthermore use a Bayesian framework utilizing Monte Carlo Markov Chain simulation techniques.

One straight-forward extension for the model presented here is to allow for arbitrary prior weights for the various triangle cells. Some grouping of development periods (or accident periods) based on parametric curves should not be too difficult to implement either.

A persistent nuisance of most stochastic reserving models for development triangles is that they do not work for negative incremental values. As we have noted in the case of our model in section 2, this seems to be due to the seemingly inevitable choice of a logarithmic link function or a similar transformation involving taking a logarithm. Given how often we encounter triangles with negative incremental values as practicing U.S. P&C reserving actuaries, one would hope that a solution to this problem is found soon.

We conclude our discussion by reiterating that the model and material presented here is intended to introduce a wider audience of P&C actuaries to regression analysis for development triangles. The paper should aid practitioners in deepening their understanding of regression analysis, in general, and GLM analysis, in particular. We also hope that practitioners will start appreciating that development triangles represent a rather condensed form of data and that even the most sophisticated stochastic models cannot recover the information that was destroyed in the process of aggregating individual claims data.

6. CONCLUSIONS

We have demonstrated that an incomplete development history is no obstacle to projecting future development. Our analysis of the graph topology of an incomplete development triangle precisely describes to what extent such projections are possible based on the data points given. Understanding the nature and implications of exact fit cells and critical connector cells is crucial for assessing the goodness of fit of the model and for bootstrapping applications. To our knowledge the application of graph theory in this context has not previously been discussed in the actuarial literature. The companion MS Excel application, which is available from the author at request, demonstrates that performing a GLM-based regression fit to a development triangle is a tool within easy reach of any practitioner with access to a personal computer.

Acknowledgment

The companion MS Excel application to this paper is derived from a stochastic reserving model internally developed by PricewaterhouseCoopers' Actuarial and Insurance Management Solutions (AIMS) practice. The author thanks all PwC staff members who helped him with this development project. PwC AIMS also supported the author of this paper by providing him with writing time. Any errors, inaccuracies or views expressed in the paper are the author's responsibility alone. PwC's support for this project in no way represents an endorsement by PwC or PwC AIMS of any of the views expressed by the author or the methods presented in the paper.

7. REFERENCES

Anderson, D. et al., "A Practitioner's Guide to Generalized Linear Models—A CAS Study Note" (3rd ed.), **2007**, URL http://www.casact.org/library/studynotes/anderson9.pdf.

Barnett, G., and B. Zehnwirth, "Best Estimates for Reserves," *Proceedings of the Casualty Actuarial Society*, **2000**, Vol. 87, 245-303.

Davison, A.C., and D.V. Hinkley, *Bootstrap Methods and Their Application*, Cambridge University Press, **1997**.

England, P.D., and R.J. Verrall, "Predictive Distributions of Outstanding Liabilities in General Insurance," *Annals of Actuarial Science,* **2006**, Vol. 1, No 2, 221-270.

Hopcroft, J., and R. Tarjan, "Efficient algorithms for graph manipulation," *Communications of the ACM,* **1973**, Vol. 16, Iss. 6, 372-378.

McCullagh, P., and J.A. Nelder, *Generalized Linear Models* (2nd ed.), London: Chapman & Hall/CRC, **1989**.

Pinheiro, Paulo J.R., et al., "Bootstrap Methodology in Claim Reserving," *Journal of Risk and Insurance*, **2003**, Vol. 70, No. 4, 701-714.

Schmid, Frank A., "Robust Loss Development Using MCMC," **2009**, URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1501706.

Wüthrich, Mario V., and Michael Merz, *Stochastic Claims Reserving Methods in Insurance*, Chichester: Wiley, **2008**.

Abbreviations and notations

GLM, generalized linear model
LSQ, least squares
MLE, maximum likelihood estimator (or estimation)

Biography of the Author

Thomas Hartl is a Manager within PricewaterhouseCoopers' Actuarial Insurance Management Solutions (AIMS) practice. He provides consulting services to insurance companies, reinsurers, and regulators. His responsibilities include the design, validation and implementation of simulation models supporting statistical analysis for ERM, litigation support, predictive modeling and stochastic reserving. Thomas Hartl is an Associate of the Casualty Actuarial Society, a Member of the American Academy of Actuaries, and holds a PhD in Mathematics from the University of Glasgow, Scotland.

Contact: thomas.hartl@us.pwc.com; 617-530-7524